

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ



ЧАСТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МЕЖДУНАРОДНЫЙ ИНСТИТУТ РЫНКА»

СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ЭКОНОМИКЕ
Методические рекомендации по практикуму для магистрантов

Самара 2016

Современные информационные технологии в экономике: Методические рекомендации по практикуму. / Составитель: Д.Ф. Китаев - Самара, МИР, 2016. 67 с.

Методическое пособие содержит методические материалы по дисциплине «Современные информационные технологии в экономике». В пособии приведены подробные рекомендации по работе в пакете AnyLogic, примеры реализации в пакете конкретных моделей, варианты для самостоятельного моделирования в среде AnyLogic, а также список литературы по предмету.

Пособие предназначено для обучающихся по направлению «Экономика» очной формы обучения.

Составитель: Д.Ф. Китаев

Рецензент:

*Печатается по решению редакционно-издательского совета
Международного института рынка*

© Составление Д.Ф. Китаев, 2016
© Международный институт рынка, 2016

Содержание

Оглавление

Практикум 1. Знакомство с пакетом AnyLogic	5
1.1.1. Структурная диаграмма	7
1.1.2. Окна свойств объектов модели	8
1.1.3. Окно поведения активного объекта	8
1.1.4. Окно редактора анимации активного объекта	9
1.2. Режим выполнения модели	10
1.2.1. Запуск модели	10
1.2.2. Эксперименты с моделью	11
1.2.3. Управление скоростью выполнения модели и изображением	11
1.2.4. Предварительно определенные эксперименты с моделью	12
1.2.5. Работа с окнами	12
1.3. Доработка модели BALLS	13
1.3.1. Изменение цвета мяча в анимации при отскоке	13
1.3.2. Введение второго мяча в модель	15
1.3.3. Произвольные перемещения мяча	16
1.4. Самостоятельная работа по вариантам	18
Практикум 2. Построение динамических моделей.	19
2.1. Постановка задачи	19
2.2. Построение модели	19
2.3. Запуск модели	22
2.4. Анимация модели	24
2.5 Самостоятельная работа по вариантам	28
Практикум 3. Системы массового обслуживания: часть 1: использование библиотеки стандартных объектов	28
3.1. Постановка задачи	28
3.2. Построение модели	29
3.3. Самостоятельная работа по вариантам	32
Работа 4. Оптимизация: модель сервиса мобильной связи	33
4.1. Постановка задачи	33
4.2. Построение модели	34
Параметры модели	35
Проблема оптимизации. Целевая функция	36
Штраф за необслуженные вызовы	38
Приведенная стоимость оборудования	39
Целевая функция	40
Самостоятельная работа по вариантам	42
Работа 5. Агентное моделирование. Диффузия Басса	43
1. Создание модели	43
2. Моделирование продаж под влиянием рекламы	44
3. Усложнение модели: учет влияния общения людей и повторных покупок	49
Список литературы	56

1. ЦЕЛИ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ

Целями освоения учебной дисциплины являются: усвоение студентами основных понятий и парадигм математического и имитационного моделирования при изучении объектов различной природы.

Основными задачами курса являются: знакомство студентов с принципами математического и имитационного моделирования, с методами конструирования и исследования математических моделей, программно-языковыми средствами, используемыми при моделировании, а также с пакетами прикладных программ.

Учебный курс призван прививать навыки самостоятельного исследования сложных моделей математическими методами, а также производить экспериментальное исследование созданных моделей.

2. МЕСТО УЧЕБНОЙ ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО

Данная учебная дисциплина входит в обязательную часть вариационного компонента учебного плана. Для усвоения дисциплины необходимы знания, полученные в средней общеобразовательной школе, а также в вузе в результате освоения дисциплин «Математика», «Информатика», «Моделирование бизнес-процессов».

Знания и умения, усвоенные студентами в процессе изучения дисциплины, необходимы в качестве методологической основы для исследования моделей сложных систем.

3. КОМПЕТЕНЦИИ СТУДЕНТА, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ (МОДУЛЯ)

Дисциплина «Современные информационные технологии в экономике» способствует формированию следующих компетенций, предусмотренных ФГОС-3 по направлению подготовки ВПО «Экономика»:

профессиональные (ПК):

- способностью руководить экономическими службами и подразделениями на предприятиях и организациях различных форм собственности, в органах государственной и муниципальной власти (ПК-11);
- способностью разрабатывать варианты управленческих решений и обосновывать их выбор на основе критериев социально-экономической эффективности (ПК-12)

В результате освоения дисциплины студент должен:

Знать:

- основные положения теории математического и имитационного моделирования, особенности методов и этапов;
- модели систем массового обслуживания;
- понятие о технологиях моделирования случайных факторов в различных системах;
- методики планирования и реализации экспериментов с моделями.

Уметь:

- ставить и решать конкретные задачи по разработке математических и имитационных моделей;
- правильно выбирать критерии и показатели адекватности моделей;
- осуществлять анализ и интерпретацию результатов моделирования

Владеть:

- приемами конструирования математических и имитационных моделей в различных областях человеческой деятельности;
- навыками проведения экспериментальных исследований созданных моделей с оценкой их адекватности исходным прототипам.

4. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ К ЛАБОРАТОРНОМУ ПРАКТИКУМУ

Практикум 1. Знакомство с пакетом AnyLogic

1.1. Интерфейс программы.

При запуске AnyLogic отображается стартовая страница. Со стартовой страницы можно создать новый проект, открыть проект, с которым недавно работали, или открыть один из уже разработанных примеров моделей AnyLogic. AnyLogic при открытии проекта всегда открывает среду разработки проекта - графический редактор модели. На рис 1.1 показаны основные составляющие пользовательского интерфейса этого редактора. *Рассмотрим их поочередно на примере модели **Balls**, которую нужно загрузить из директории, указанной преподавателем.*

Окно **Проект** обеспечивает навигацию по элементам проекта (рис. 1.2 для модели **Balls**). Проект всегда организуется иерархически, поэтому он отображается в виде дерева: сам проект (с именем *balls*) образует корень дерева рабочего проекта, классы активных объектов и сообщений - следующий уровень, в классах активных объектов могут быть включены анимация, функции и т. д. Активный объект является основным структурным элементом модели в AnyLogic. Активным объектом называется сущность, которая инкапсулирует (включает в себя) данные (атрибуты объекта), функции (методы) 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*.

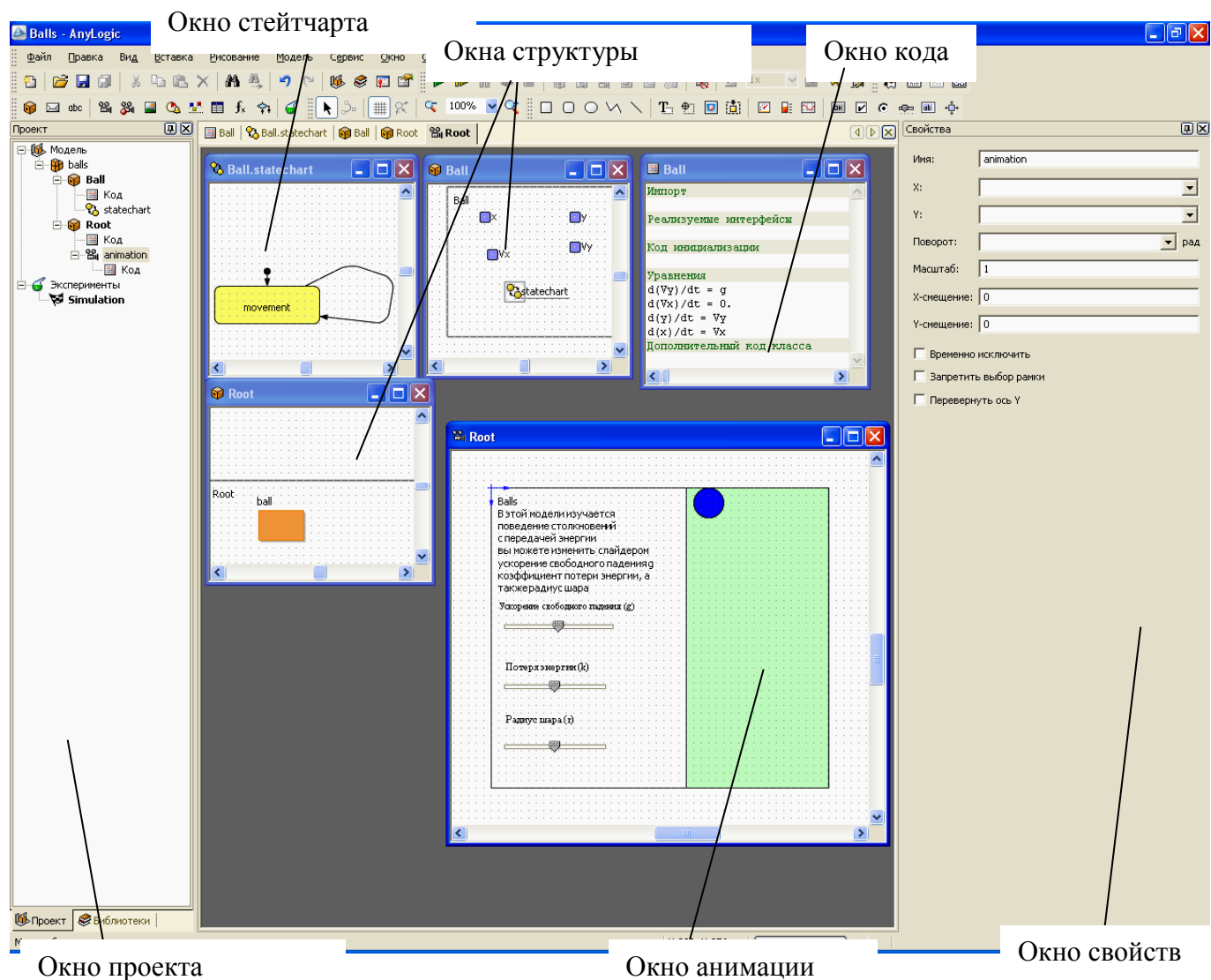
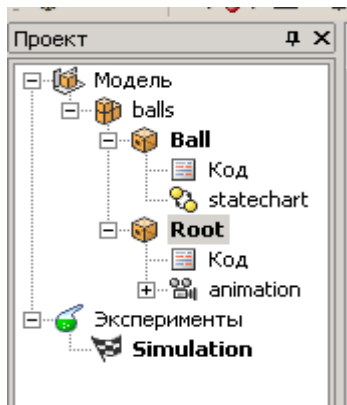


Рис. 1.1

На дереве проекта (рис. 1.2) как составные элементы класса *Ball* показаны *Код* и стейтчарт с именем *statechart*, у класса *Root* составными его элементами показаны *Код* и анимация с именем *animation*.

Одна из ветвей в дереве проекта имеет название *Эксперименты*, этот объект объединяет группу экспериментов, которые могут быть выполнены с моделью. В открытом нами проекте в группу экспериментов входит только один эксперимент с именем ***Simulation***, выделенный жирным шрифтом - это **текущий** эксперимент, именно в соответствии с установленными в текущем эксперименте параметрами будет происходить выполнение модели после ее компиляции.

Открыть окно проекта, если оно закрыто, можно с помощью щелчка мышью по кнопке **Проект** панели инструментов или командой **Вид / Проект** из главного меню. Окно любого элемента дерева проекта (классов активных объектов, стейтчартов, анимации и т. п.) можно открыть двойным щелчком мышью по имени этого элемента в дереве проекта. Открытая нами среда создания проекта (рис. 1.1) содержит несколько окон. Структура активного объекта задается графически в специальном окне редактора - **структурной диаграмме**. В окне **Ball.statechart** представлен **стейтчарт** (или карта состояний), который определяет реакции активного объекта на внешние события - логику его действий во времени. Стейтчарт является удобным расширением



графического представления классического конечного автомата, он состоит из состояний и переходов между состояниями. В окне анимации можно построить анимированное поведение активного объекта.

Поле в любом окне можно двигать, нажав на нем правой кнопкой мыши. При этом также можно менять масштаб изображения в этом поле с помощью двух кнопок и окна масштаба, если окно активно.

Закройте и откройте окно проекта *balls*, а также каждое из окон элементов, включенных в дерево проекта. Измените масштаб

изображений в окнах.

Рис. 1.2

1.1.1. Структурная диаграмма

При построении любой модели задается ее структура (т. е. ее компоненты и связи этих компонентов) и поведение отдельных компонентов. В AnyLogic активный объект имеет структуру и поведение. Структуру активного объекта составляют его параметры, переменные, стейтчарты, а также экземпляры других активных объектов, включенные как компоненты в данный активный объект (и, возможно, их связи).

На рис. 1.1 для нашего простого примера структурная диаграмма активного объекта - мяча - задается в окне с именем **Ball** прямоугольником, внутри которого содержатся его переменные (координаты мяча x , y и его скорости V_x и V_y) и иконка поведения с именем *statechart*. Поскольку этот простой объект (*Ball*) не содержит экземпляров других объектов, то в прямоугольнике с именем *Ball* нет других вложенных прямоугольников и связей (рис. 1.3).

Связь переменных друг с другом обеспечивается уравнениями:

$$V_x = \frac{dx}{dt}, \quad V_y = \frac{dy}{dt}, \quad \frac{d(V_x)}{dt} = 0, \quad \frac{d(V_y)}{dt} = g$$

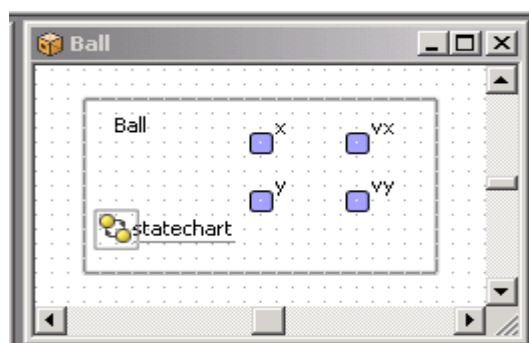


Рис. 1.3

Структура корневого активного объекта *Root* задана в окне с именем **Root**. Именно корневой активный объект является верхним уровнем структурной иерархии в нашем проекте. В модели (рис. 1.1) активный объект *Root* содержит один прямоугольник с именем *ball* - один экземпляр активного объекта *Ball*.

1.1.2. Окна свойств объектов модели

В редакторе AnyLogic для каждого элемента модели существует свое окно свойств, в котором указываются свойства (параметры) этого элемента. При выделении какого-либо элемента в любом из окон редактора (в окне структуры, окне поведения, окне анимации или в окне проекта) справа появляется окно свойств именно этого выделенного элемента. Выделить элемент можно щелчком левой кнопки мыши на этом элементе. Окно **Свойства** (рис. 1.5) используется для просмотра и изменения свойств элементов. Открыть это окно, если оно закрыто, можно кнопкой **Свойства** панели инструментов.

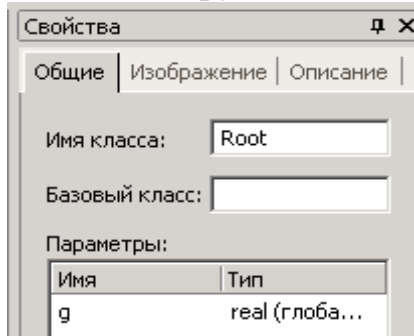


Рис. 1.4

Например, при выделенном окне редактора структуры объекта *Ball* справа появится окно свойств, которое содержит 3 вкладки: **Общие**, **Изображение** и **Описание**. На вкладке **Общие** кроме имени этого объекта указаны его параметры. Смысл параметров очевиден: g – ускорение свободного падения, r – радиус мяча и k – коэффициент, определяющий уменьшение скорости мяча при каждом отскоке, $x0$ и $y0$ – начальные значения координат.

1.1.3. Окно поведения активного объекта

Поведение мяча представлено в окне **Ball.statechart**, содержащем простейший стейтchart (карту состояний), который можно считать расширенным графом переходов конечного автомата.

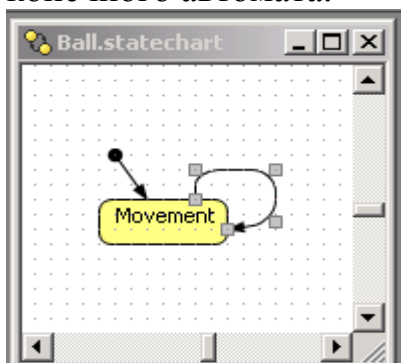


Рис. 1.5

Стейтchart модели прыгающего мяча (рис. 1.5) состоит из одного состояния с именем *Movement* и одного перехода. Переход срабатывает при наступлении события касания поверхности земли при движении мяча вниз. Условие наступления этого события можно записать выражением:

$$y \geq y0 - r \ \&\& \ Vy > 0$$

Это выражение означает, что вертикальная координата y центра мяча с радиусом r отстоит от поверхности (определяемой координатой $y0$) на r и при этом скорость мяча

V_y направлена вниз, в сторону увеличения координаты y . Именно это выражение записано в поле **Событие** окна свойств перехода (рис. 1.6), которое открывается при выделенном переходе стейтчарта. При наступлении данного события мяч отскакивает, т. е. его скорость меняет свой знак, уменьшаясь при этом на долю k , показывающую потерю энергии при отскоке. Это отражено в поле **Действие** окна свойств перехода.

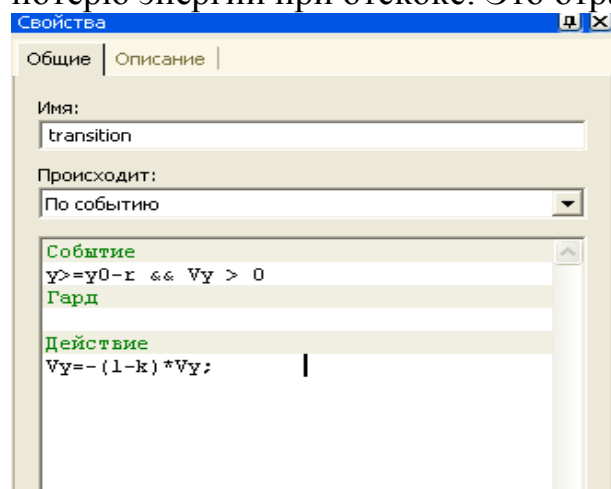


Рис. 1.6

Таким образом, стейтчарт в этой модели следит за событиями. При наступлении нужного события выполняется необходимое действие. И условие наступления события, и само действие, меняющее переменные модели, записываются здесь обычными алгебраическими соотношениями.

В настоящее время стейтчарты широко используются как удобное средство визуального описания поведения сложных систем. В частности, стейтчарты являются основным средством представления поведения в языке UML (Unified Modeling Language).

1.1.4. Окно редактора анимации активного объекта

В этом окне для модели строится двумерное анимационное представление, которое показывает, что происходит с моделью с течением времени. Именно здесь визуально представляется имитация поведения моделируемой системы. Для модели *Balls* в окне анимации построено изображение мяча, представленного закрашенным кругом. В другом окне можно построить трехмерное анимационное представление.

Анимация в AnyLogic создается в виде динамических графических объектов, которые дают возможность наглядно представить динамику моделируемой системы, т. е. поведение ее во времени. Основная идея здесь состоит в том, что параметры элементов анимационной картинки (для круга это его координаты центра, радиус, цвет и т. п.) связываются с переменными и параметрами модели. Изменение переменных модели во времени заставляет изменяться во времени графический образ, что позволяет наглядно представить динамику моделируемой системы с помощью динамически меняющейся графики. На рис. 1.7 показано, что когда в нашей модели в окне анимации выделен синий круг (щелчком правой кнопкой мыши), то в окне свойств представлены свойства этого круга. А именно координаты X и Y центра круга анимационной картинки имеют статические значения (значения этих параметров в редакторе) 90 и 400, а в поле динамических значений этих параметров они связаны с переменными x и y экземпляра *ball* активного объекта *Ball*. Таким образом, изменение данных переменных будет вызывать перемещение центра графического образа мяча на

анимационной картинке при работе модели.

Щелкните мышью на нескольких элементах окон редактора (на переменных и на поле окна редактора структуры, на переходе и состоянии окна стейтчарта, на графических элементах окна редактора анимации, на объекте *Simulation* дерева проекта и т. п.). Вы увидите, что для каждого элемента модели окно свойств имеет свою структуру и содержит специфическую информацию и параметры, характеризующие именно данный элемент. Например, щелкните мышью в окне анимации на поясняющем тексте. В окне свойств на вкладке **Текст** будет представлен только один параметр - сам текст. Его можно редактировать и наблюдать, как в поле текста окна анимации он изменяется.

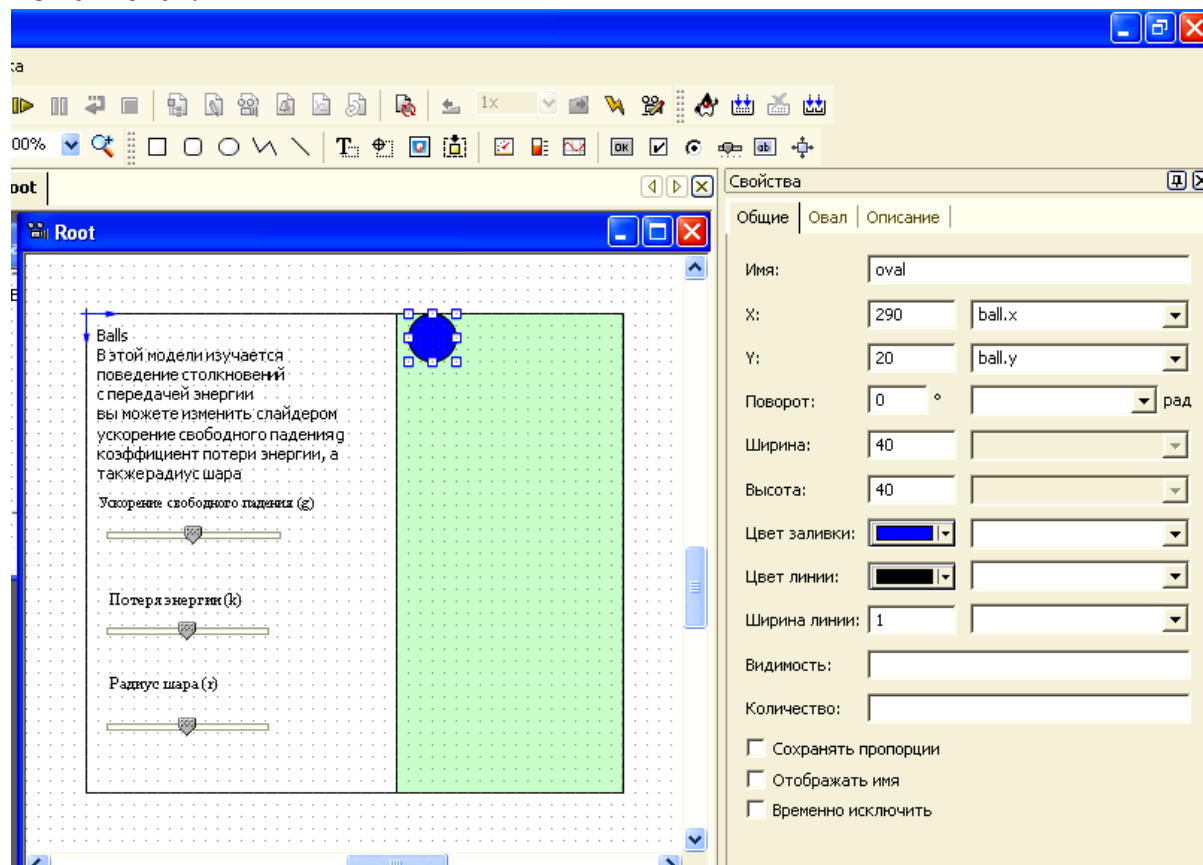


Рис. 1.7

1.2. Режим выполнения модели

1.2.1. Запуск модели

Для запуска модели щелкните кнопку **Запустить** на панели инструментов. Этим действием запустится компилятор, который построит исполняемый код проекта на языке Java, оттранслирует его и затем запустит модель на исполнение. При этом откроется окно анимации с прыгающим мячом;

Кроме окна наблюдения при работе модели могут быть открыты и другие окна, например, кнопкой **События** панели инструментов можно открыть окно событий (в котором фиксируются будущие события, запланированные к данному моменту исполнительной системой), кнопкой **Общий журнал** журнал (Log), куда попадают комментарии компилятора и куда разработчик может вывести отладочную информацию.

1.2.2. Эксперименты с моделью

В данном примере в окне анимации кроме движущегося изображения мяча можно видеть текстовый комментарий и так называемые "слайдеры" или "бегунки" - подвижные указатели для изменения параметров модели во время ее выполнения. Двигая слайдеры, можно менять в этой модели два параметра - ускорение свободного падения g , долю k потери скорости прыгающим мячом при каждом отскоке радиус шарика. Изменение параметров позволяет исследовать поведение модели в различных условиях - это и есть компьютерный эксперимент. Изменение параметров и переменных возможно также и без слайдеров. Двойной щелчок мыши на переменной или параметре в окне с именем **Root** останавливает выполнение модели и открывает диалог с текущим значением этой характеристики, которое можно редактировать. Последующий запуск приведет к продолжению выполнения модели уже с измененным значением.

Для проведения компьютерных экспериментов необходимо использовать (кроме уже известной кнопки **Запустить** компиляции и запуска модели на выполнение) также следующие кнопки:

- запуска выполнения модели по шагам – **Выполнить шаг**;
- паузы – **Пауза**;
- повторного запуска с исходными начальными условиями – **Перезапустить**;
- разрушения скомпилированной модели и возврата в редактор – **Остановить**.

Вместо этих кнопок можно в основном меню **Модель** выполнить команды **Запустить**, **Выполнить шаг** и т. п.

Активность кнопки **Остановить** показывает, что исполнительная система запущена (модель работает либо приостановлена в паузе), пассивная (негорящая) кнопка показывает, что вы находитесь в редакторе.

Проведите несколько экспериментов с моделью, изменяя параметры и переменные модели либо слайдерами, либо вызывая эти параметры двойным щелчком для изменения в окне **Root**.

1.2.3. Управление скоростью выполнения модели и изображением

В AnyLogic скорость выполнения модели может быть установлена максимальной (и модель будет выполняться в режиме виртуального времени с максимально возможной скоростью выполнения соответствующего программного кода) либо соответствующей (при возможности) реальному физическому времени с некоторым коэффициентом. Единица модельного времени при этом равна одной секунде физического времени. Переключение между виртуальным и реальным временем исполнения модели осуществляется кнопкой **Виртуальное время** панели инструментов, а уменьшение масштаба времени (масштаба модельного времени относительно физического только в режиме реального времени) выполняется с помощью двух кнопок **Уменьшить скорость**, **Увеличить скорость** и расположенного между ними поля. Это поле указывает коэффициент ускорения модельного времени относительно физического (здесь 1х означает единичный коэффициент ускорения).

Выполните несколько экспериментов с различными скоростями выполнения модели, используя кнопки останова, рестарта, запуска.

1.2.4. Предварительно определенные эксперименты с моделью

Запуск модели на выполнение производится в AnyLogic в соответствии с определенным набором значений параметров модели, а также с некоторыми дополнительными установками (например, точность, шаг численных методов и т. п.). Совокупность всех установок для проведения компьютерного эксперимента с моделью называется в AnyLogic *экспериментом*. Все эксперименты, возможные для выполнения в данном проекте, представлены как элементы группы (корня) с именем *Эксперименты* в окне проекта. Один такой эксперимент с названием *Simulation* уже построен при создании нового проекта с установками по умолчанию, он и выбран в качестве текущего (название текущего эксперимента показано жирным шрифтом).

Значения параметров, режим реального либо виртуального времени при выполнении эксперимента, условие прекращения выполнения эксперимента и многое другое, относящееся к проведению эксперимента, можно до запуска модели установить в окне свойств объекта *Simulation*, являющегося в данном проекте единственным элементом группы *Эксперименты* в дереве классов модели. В окне **Свойства** объекта *Simulation* вы можете увидеть эти параметры и поменять установки, прежде чем запустить модель. Например, на вкладке **Дополнительные** можно определить условие остановки выполнения модели по времени как целое число единиц модельного времени и запустить модель. Полный набор свойств экспериментов описан в руководстве пользователя AnyLogic.

Для одной и той же модели в AnyLogic можно определить несколько различных экспериментов на этапе построения модели.

1.2.5. Работа с окнами

Работать с окнами при разработке и исследовании модели требуется постоянно. Поэтому необходимо уметь открывать нужные окна, изменять их размеры, закрывать, сворачивать и разворачивать.

Вновь откройте редактор проекта *Balls*. В редакторе и при работе модели открытие и закрытие окна проекта и окна свойств выполняется соответственно кнопками **Проект** и **Свойства** панели инструментов. Кроме того, открыть эти окна можно, выбрав в главном меню команды **Вид / Модель** и **Вид / Свойства** соответственно. Откройте и закройте окна несколько раз в редакторе и в окне наблюдения при выполнении модели.

Окна структуры, поведения или анимации можно открыть двойным щелчком мыши на именах соответствующих объектов дерева проекта, если они закрыты. Попробуйте закрыть и открыть несколько раз каждое из окон редактора. Для удобства редактирования любое окно может быть максимально увеличено так, чтобы оно заняло все поле редактора. В этом случае кнопки управления таким окном размещаются в правом верхнем углу. С любым объектом (графиком, переменной, активным объектом в окне редактора и т. п.) в редакторе AnyLogic связано контекстное меню, которое появляется при выделении этого объекта. Размеры окон можно менять, как и в любом Windows-приложении. Для помещения изображения в центр окна после изменения размеров окна в его контекстном меню выберите команду **Перейти в центр**.

Каждое окно редактора сделайте активным, щелкнув на нем мышью, измените его размеры, поместите изображение в центр нового окна, измените масштаб изображения

в ту и другую сторону.

Запустите модель на выполнение по шагам. В появившемся окне наблюдения можно закрыть и затем открыть окно (с предопределенным именем **Root**) структуры модели, показывающее текущие значения всех переменных и параметров модели (командой **Вид | Корневой объект модели** или кнопкой **Дерево объектов** панели инструментов). Двойной щелчок мыши на переменной или параметре в этом окне вызовет появление диалога для модификации данного объекта. Тот же эффект вызовет и команда **Изменить** контекстного меню этого объекта (контекстное меню выделенного объекта всегда вызывается правой кнопкой мыши). Команда **Вид / Анимация** откроет окно анимации модели, если оно закрыто.


Заметьте, что закрытие окон графиков уничтожает их, они не сохраняются и их нужно будет снова создавать. Если вы удалили график, можно закрыть проект (не сохраняя сделанные изменения) и снова его открыть. Введение и исключение переменных из графика выполняется выбором команды **Содержимое диаграммы** контекстного меню графика.

1.3 Доработка модели BALLS

Выполним некоторые упражнения с моделью *Balls*, которые дадут некоторое представление о средствах разработки моделей в AnyLogic.

1.3.1 Изменение цвета мяча в анимации при отскоке

Для большей наглядности дополним анимационное представление поведения мяча так, чтобы при отскоке мяча его цвет на мгновение изменялся на красный. Для этого нужно зафиксировать момент отскока (запомнить значение момента времени наступления этого события) и установить красным цвет шара в анимации на небольшой интервал времени, следующий за этим моментом.

Введите сначала переменную *tBounce*, которая будет фиксировать момент отскока. Для этого сделайте активным окно структуры активного объекта *Ball* и перенесите внутрь ограничивающего прямоугольника иконку  (**Переменная**), щелкнув левой кнопкой мыши сначала по этой иконке на инструментальной панели, а потом в окне. В поле **Имя** открывшегося окна свойств этой переменной введите *tBounce*, а в поле **Начальное значение** введите -1 (рис.1.8).

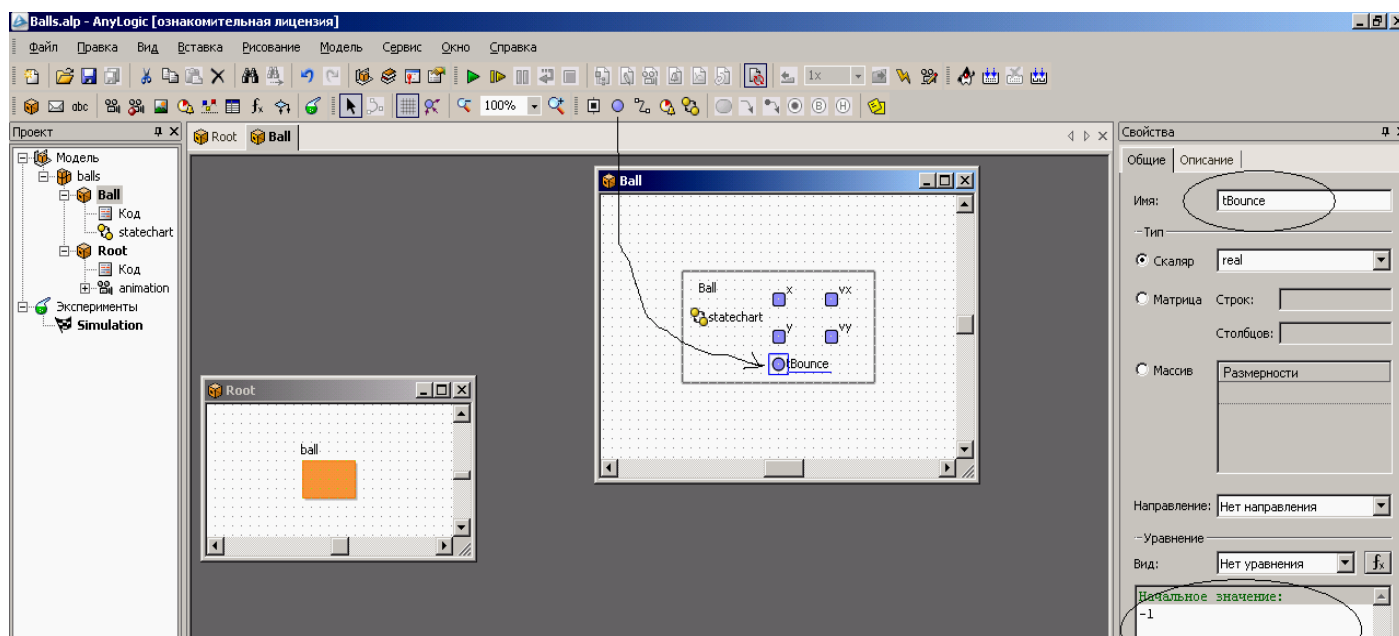


Рис. 1.8 Введение новой переменной в активный объект Ball

Для того чтобы переменная *tBounce* фиксировала момент отскока, нужно значение текущего времени в модели при наступлении события "отскок" запомнить в этой переменной. За наступлением данного события следит стейтchart, поэтому сделайте активным окно стейтchartа (рис. 1.9), в нем выберите переход и в поле **Действие** [перехода] добавьте действие:

$$tBounce = getTime();$$

При каждом вызове функция *getTime* дает текущее значение модельного времени.

Переменная *tBounce* имеет начальное значение -1 и при работе модели хранит значение момента времени последнего отскока. Для того чтобы каждый раз при отскоке мяча его цвет изменялся на красный (скажем, в течение 0.5 с), нужно установить в поле **Цвет заливки** графического изображения мяча в окне анимации динамическое значение цвета (рис. 1.10):

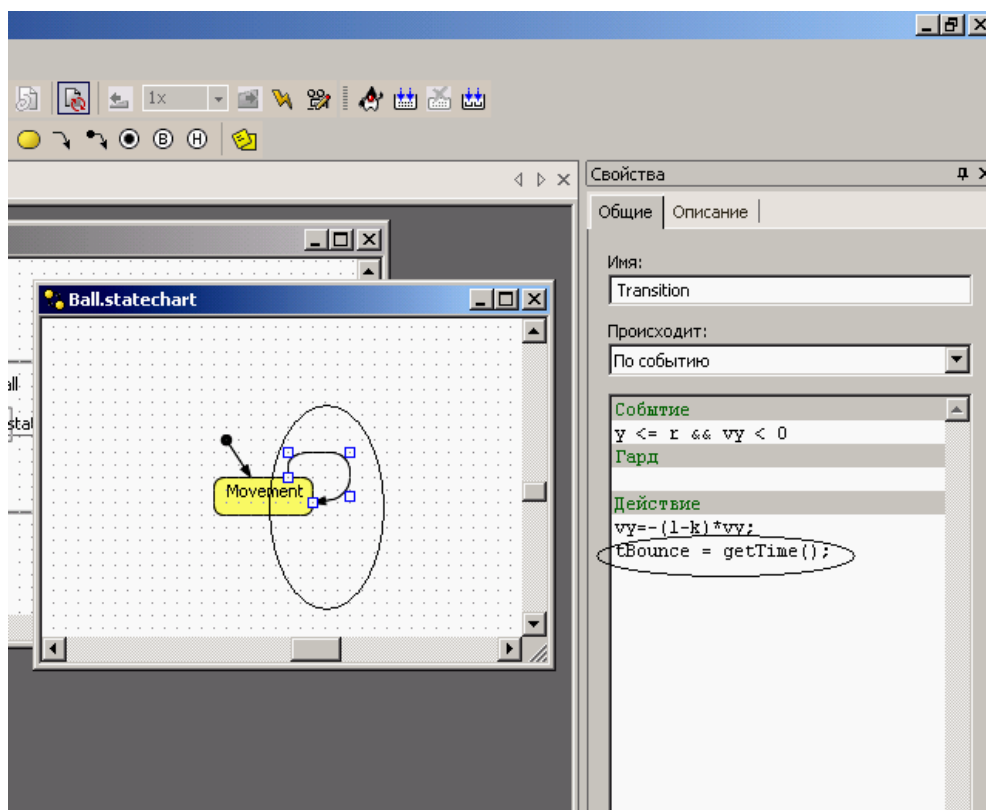


Рис. 1.9

$getTime() < ball.tBounce + 0.5$? Color.red: Color.blue

Это условное выражение устанавливает цвет заливки изображения мяча *ball* красным в течение 0.1 с после каждого отскока.

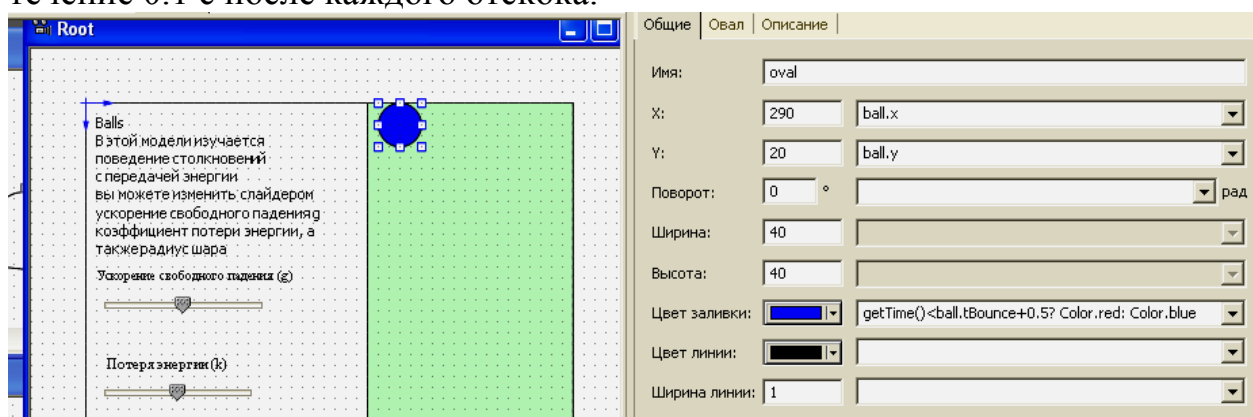


Рис. 1.10. Запоминание времени в момент отскока

1.3.2. Введение второго мяча в модель

Введите в модель второй мяч. Для этого сделайте активным окно структурной диаграммы *Root* и перенесите в него еще один экземпляр мяча, щелкнув левой кнопкой мыши сначала по имени *Ball* в окне проекта, а потом в окне структурной диаграммы *Root*. Появившийся объект автоматически получит имя *ball1* (рис. 1.11). При этом справа откроется окно свойств выделенного объекта (нового экземпляра мяча), в котором установлены те значения параметров мяча, которые были определены для активного объекта *Ball*.

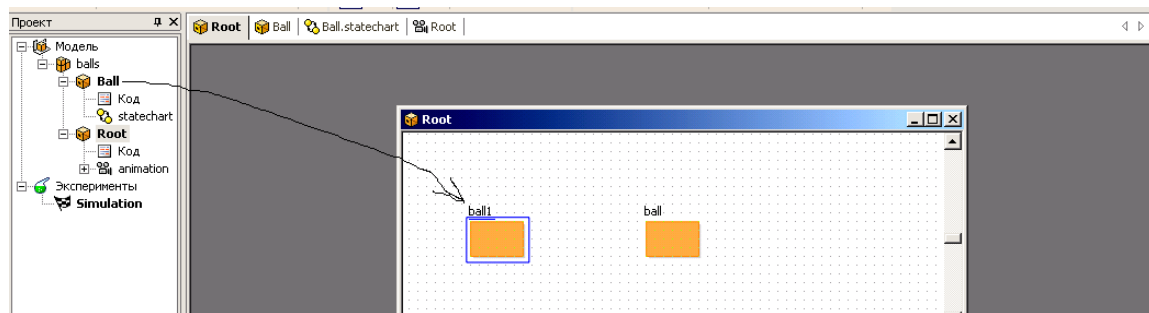


Рис. 1.11. Введение нового экземпляра мяча в модель

Установите начальные значения x_0 и y_0 координат x и y нового мяча равными 400 и 400. Остальные значения – таких же как для первого шара. Чтобы движение нового мяча отобразить в анимации, в окне анимации продублируйте изображение мяча. Для чего нажмите левой кнопкой мыши на изображении мяча и при нажатой клавише <Ctrl> перенесите это изображение в другое место поля анимации. Для нового изображения круга в правой части окна редактора появится окно свойств, в котором динамические значения параметров (координат и цвета) связаны с характеристиками шара *ball*.

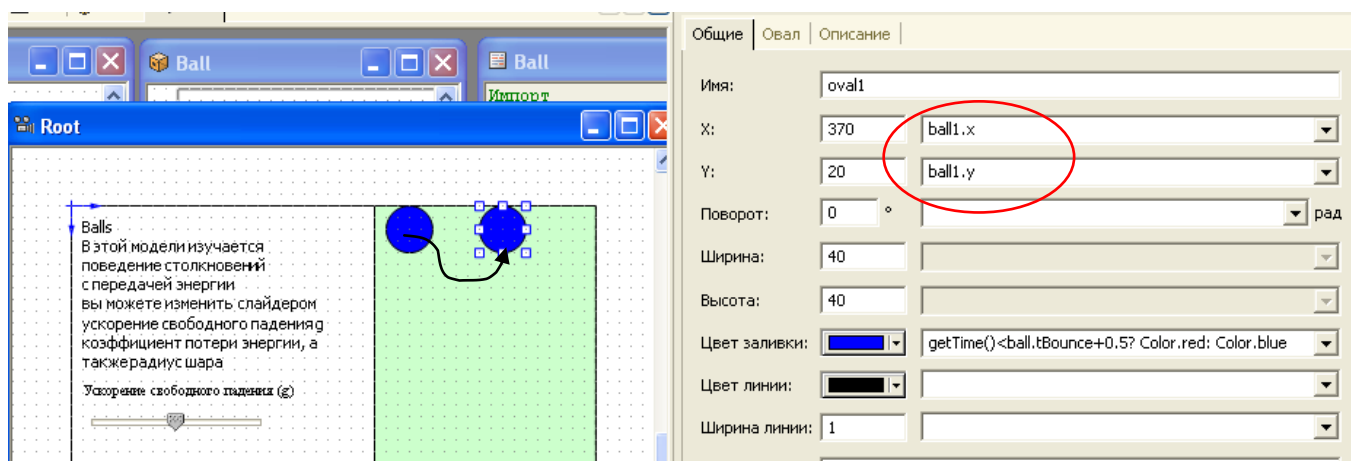


Рис. 1.12. Введение изображения нового экземпляра мяча копированием существующего изображения

Их нужно связать с новым объектом - шаром с именем *Ball1*. Иными словами, вместо *Ball.x*, *Ball.y* и *Ball.tBounce* в соответствующих полях нужно поставить *Ball1.x*, *Ball1.y* и *Ball1.tBounce* (рис. 1.12). Точно также на вкладке **Овал** этого окна для значения радиусов *Радиус 1* и *Радиус 2* нужно установить *Ball1.r* вместо *Ball.r*.

Запустите модель. В модели теперь будут имитироваться независимые движения двух шаров.

Продемонстрируйте свою модель преподавателю.

1.3.3. Произвольные перемещения мяча

В данной модели мячи движутся вертикально, отталкиваясь от поверхности с координатой y_0 . Это происходит потому, что начальная скорость мячей по координате x равна 0. Если мы изменим начальные скорости мячей по этой координате, сделав их, например, случайными, нам необходимо задать, как мячи будут себя вести при встрече с потолком и вертикальными стенками.

Вернемся снова к экспериментам с одним мячом. Удалите из окна структурной

диаграммы объекта *Root* объект *Ball1*, а из окна анимации удалите шар с именем *oval1*, отображавший движение шара *ball1*.

Сделаем сначала случайными начальными значения скоростей V_x и V_y мяча. Активизируйте окно структурной диаграммы активного объекта *Ball*, выделите переменную V_x и в поле **Начальное значение** этой переменной замените значение 0 на значение *uniform(-100, 100)*. Тем самым начальная скорость по координате x у различных экземпляров активного объекта *Ball* будет выбираться случайно из диапазона (-100, +100) метров в секунду как реализация случайной величины, равномерно распределенной в этом диапазоне. То же самое сделайте для переменной V_y .

Для учета отталкивания мяча от потолка нужно событие встречи препятствия мячом на переходе стейтчарта изменить. Размеры поля, в котором двигаются мячи, установлены 200x400, начиная с координаты $x = 250$. В поле **Событие** окна свойств перехода стейтчарта активного объекта *Ball* выражение:

$$y \geq y0 - r \ \&\& \ V_y > 0$$

следует дополнить:

$$y \leq r \ \&\& \ V_y < 0 \ || \ y \geq 400 - r \ \&\& \ V_y > 0$$

Действие, которое выполняется и в том, и в другом случае, будет тем же: изменение направления скорости V_y с частичной ее потерей.

Для того чтобы учесть отталкивание мяча от вертикальных стен, нужно учесть это событие в стейтчарте введением дополнительного перехода. Сделайте окно стейтчарта активным и добавьте к состоянию *Movement* дополнительный переход, используя иконку панели инструментов. Щелкнув мышью на этой иконке, поместите указатель мыши на границе состояния и продвигайтесь по полю окна стейтчарта, щелкая кнопкой мыши в тех местах, где должны располагаться точки изгиба линии перехода. Двойной щелчок мыши заканчивает рисование перехода стрелкой (рис. 1.13).

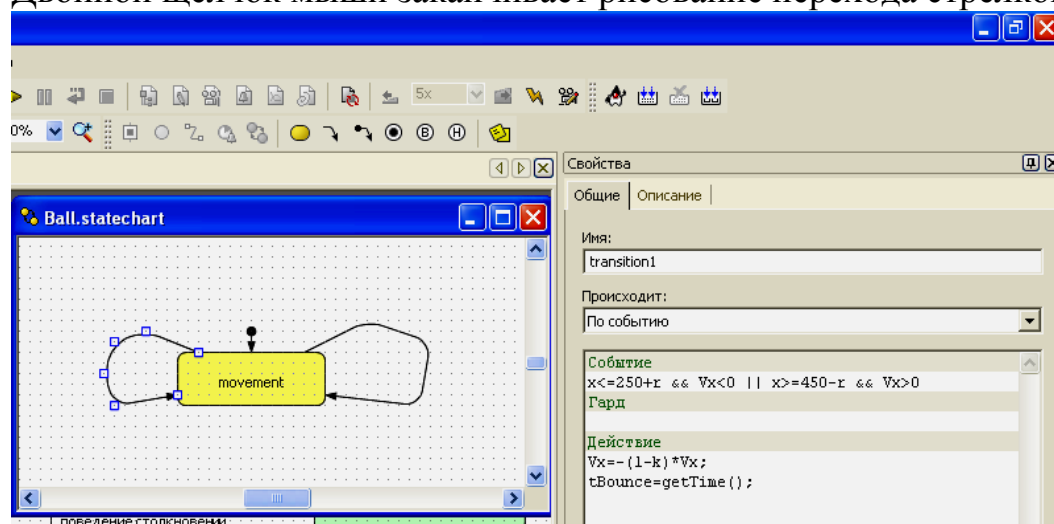


Рис. 1.13. Рисование перехода

В появившемся справа окне свойств этого перехода в поле **Происходит** нужно выбрать вариант **По событию**, в поле **Событие** следует вставить условие наступления события касания мяча о вертикальную стенку:

$$x \leq 250 + r \ \&\& \ V_x < 0 \ || \ x \geq 450 - r \ \&\& \ V_x > 0$$

а в поле **Действие** установить действия изменения направления составляющей V_x скорости мяча и моментальное изменение цвета мяча при столкновении (рис.1.13):

$$V_x = -(1 - k) * V_x;$$

$$tBounce = getTime();$$

Запустите модель на исполнение. Используя слайдеры, можно изменять радиус мяча, коэффициент потери скорости при встрече с препятствием, ускорение свободного падения, превращая мяч в воздушный шар при $g < 0$ или в бильярдный шар при $g = 0$. Продемонстрируйте свою модель преподавателю.

1.4. Самостоятельная работа по вариантам

1. Измените модель таким образом, чтобы при каждом отскоке изменялся диаметр мяча на 20%.
2. Измените модель таким образом, чтобы один из мячей притягивался вниз, а второй вверх.
3. Измените модель таким образом, чтобы подсчитывалось количество отскоков мяча.
4. Измените модель таким образом, чтобы подсчитывалось время, прошедшее с момента последнего отскока мяча.
5. Измените модель таким образом, чтобы при полете вверх мяч был красного цвета, а при полете вниз - голубого.
6. Измените модель таким образом, чтобы на один мяч сила тяжести действовала по оси X, а на другой по оси Y.
7. Измените модель таким образом, чтобы при отскоке от горизонтальных стенок мяч становился желтым на 0,5 сек., а при отскоке от вертикальных – зеленым.
8. Измените модель таким образом, чтобы мяч №1 менял свой цвет тогда, когда мяч №2 соударяется со стенками.
9. Измените модель таким образом, чтобы моделировалось движение мяча в пространстве в форме окружности (вариант для продвинутых студентов) .

Практикум 2. Построение динамических моделей.

2.1. Постановка задачи

Мы рассмотрим простейшую динамическую модель на примере простой модели, описывающей процессы, которые похожи на биение сердца. Модель эта задается парой дифференциальных уравнений первого порядка:

$$\begin{aligned}dx/dt &= (x - x^3 - b)/eps, \\db/dt &= x - x0,\end{aligned}$$

где x представляет радиус сердца, x_0 - его значение в начальный момент, b – переменная состояния, а eps - параметр. Эта модель - одна из простейших, описывающих динамику работы сердца.



Целью построения данной модели является исследование характера зависимостей переменных состояния x и b от времени при разных значениях параметра eps , а также анализ фазовой диаграммы изменения радиуса x от значения b . Очевидно, что все эти величины вещественные (с плавающей точкой). Для построения модели необходимо задать начальные значения переменных x и b , а также значение параметра eps .

2.2. Построение модели

Для хранения ваших проектов необходимо создать новую папку. Создайте, папку: MyModel на Рабочем столе. Запустите AnyLogic. Для построения нового проекта в вашей папке щелкните кнопку **Создать** панели инструментов, либо щелкните по кнопке **Новый проект** слева внизу окна, либо выберите в основном меню **Файл / Создать**.

В появившемся диалоговом окне установите нужную рабочую папку, в ней наберите *heart* как имя проекта (это имя нового файла, в котором будет храниться ваш новый проект) и щелкните ОК. Новый проект под названием *heart* будет создан (обратите внимание, что название проекта должно быть написано со строчной буквы). Измените имя корневого объекта нашей модели, назвав его *Heart* (вместо установленного по умолчанию имени *Main*). Для этого в поле **Имя класса** вкладки **Общие** окна **Свойства** корневого объекта введите *Heart* вместо *Main*. В окне классов имя корневого объекта сразу изменится.

Наша задача - построение модели, в которой присутствуют две переменные состояния, x и b , и два параметра – x_0 и eps , где x_0 - начальное значение x . Начальное значение переменной b зададим константой.

Для введения первой переменной x щелкните мышью на кнопке переменной  панели инструментов, после чего щелкните мышью в каком-либо месте поля окна редактора структуры объекта Heart. Пиктограмма  появится в поле редактора с именем var. Одновременно справа вместо окна свойств объекта Heart появится окно свойств переменной (именно эта переменная сейчас выделена). В это окно в поле имени (Имя) вместо предопределенного имени var введите x (рис. 2.1) и нажмите клавишу < Enter >. При выделенной пиктограмме переменной ее имя можно перемещать по полю окна структуры и изменять. Саму пиктограмму переменной также можно перемещать по полю при нажатой на ней левой кнопке мыши.

В модели переменная x определяется дифференциальным уравнением

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

с начальным значением x , равным x_0 , и с параметром eps . В AnyLogic можно подобные зависимости задавать именно в таком аналитическом виде. Для того чтобы определить переменную x , в поле **Вид** окна ее свойств выберите вариант **Интеграл или накопитель** в выпадающем меню и определите значение в этом поле в строке ниже уже установленного $dx/dt =$ как: $(x - x^3 - b)/eps$ (рис. 2.1). Заметьте, что переменная, определенная как **Интеграл или накопитель**, в поле структуры модели будет изображаться фиолетовым прямоугольником со скругленными углами.

В поле начального значения запишите x_0 . Вторая переменная b определяется дифференциальным уравнением $db/dt = x - x_0$. Действия по ее введению в модель аналогичны тем, которые были сделаны ранее для переменной x . Пусть начальное значение b равно 0. В поле **Начальное значение** окна свойств переменной b величину 0 можно не записывать: если это поле пусто, по умолчанию значение переменной считается нулевым.

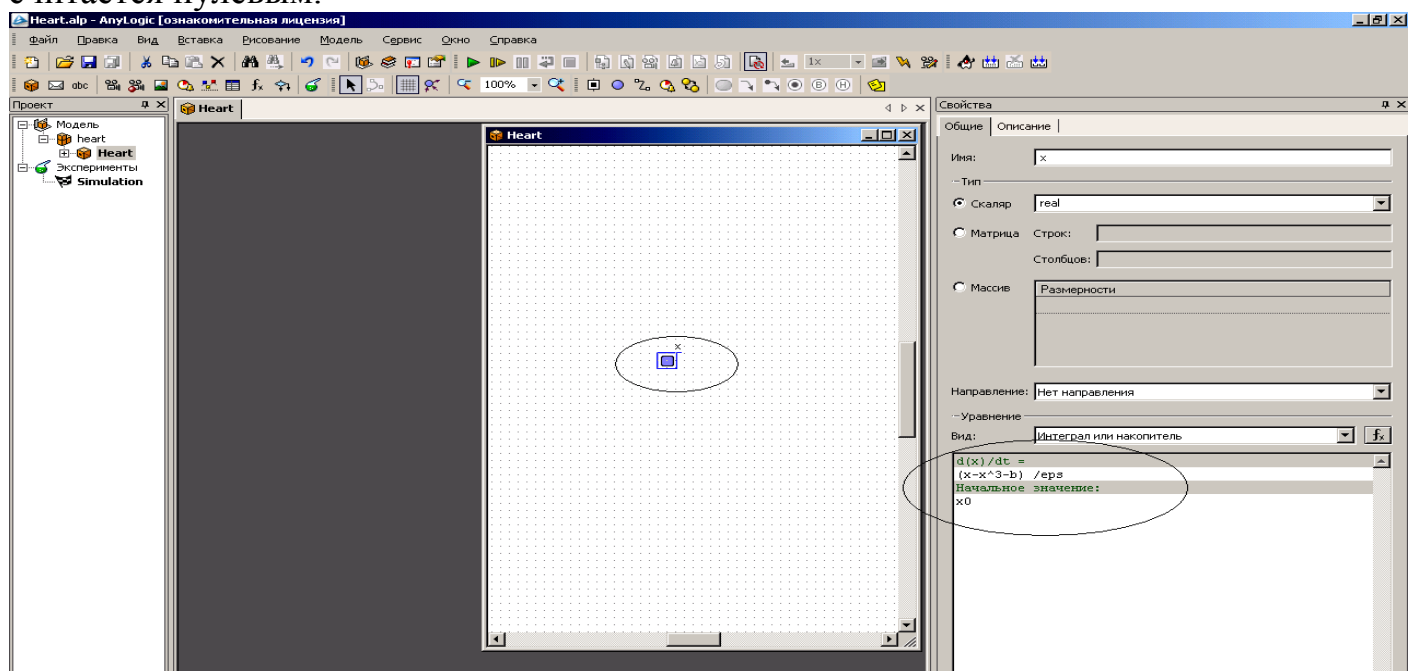


Рис. 2.1 Введение переменной в активный объект

Для проверки правильности синтаксиса (формальных правил) модели в любой момент при ее построении можно использовать кнопку **Построить** панели инструментов. Если щелкнуть на этой кнопке, то выполнится компиляция разрабатываемой модели в программный код на языке Java. Щелкните по кнопке **Построить**. В нашем примере обнаружили ошибки (рис. 2.2): действительно, нами не определены параметры x_0 и eps .

На наличие ошибки указывает появившийся символ [X] строке статуса окна редактора. Если при трансляции проекта в нем найдены ошибки, то построение программы на Java не завершается, и в появившемся окне **Вывод** внизу экрана будет представлен список обнаруженных ошибок с информацией о них. Двойной щелчок мыши по строке, в которой указана информация об ошибке, открывает окно и место в нем, где компилятор обнаружил эту ошибку.

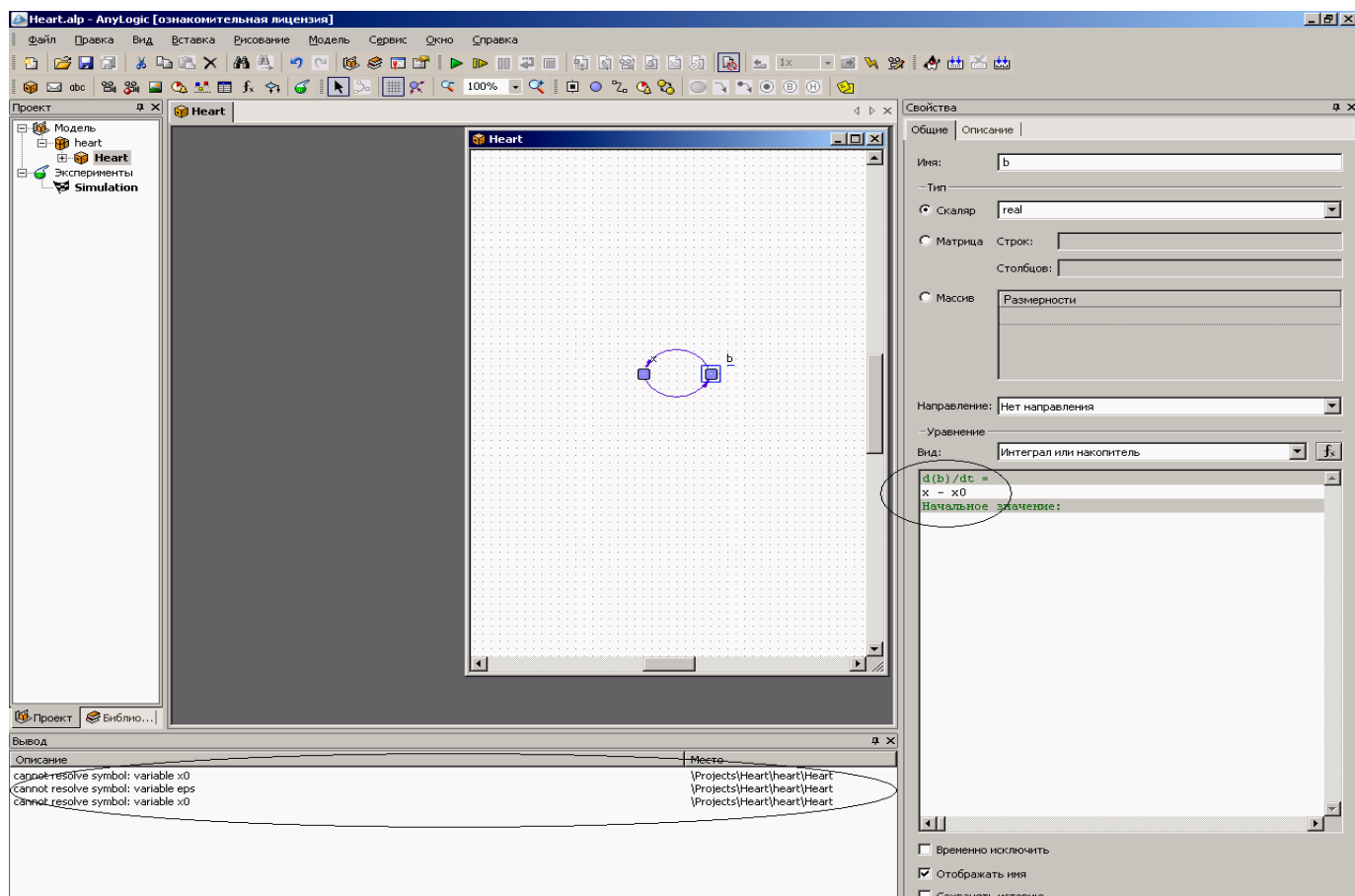


Рис. 2.2. Ошибка компиляции

Для завершения построения модели указанные параметры нужно задать. Пусть $x_0 = 0.5$, $eps = 0.01$. Параметры эти являются параметрами активного объекта *Heart*, потому они вводятся в окне свойств данного объекта. Для задания x_0 сделайте активным окно редактора структуры объекта *Heart*. Появится окно **Свойства** этого объекта.

В данном окне для задания параметров следует щелкнуть мышью на кнопке **Новый параметр** расположенной под полем **Параметры**, либо дважды щелкнуть левой кнопкой мыши в любом месте этого поля. Появится диалог **Параметры** для определения нового параметра. У него уже predetermined имя *param*, тип *real* и пустое поле с именем **По умолчанию** (следовательно, по умолчанию это будет вещественная переменная и ее значение будет нулевым). Замените имя параметра на x_0 , в поле значения установите 0.5 , а тип оставьте *real*. Остальные поля также оставьте без изменения. При нажатии кнопки **ОК** этого окна в поле **Параметры** окна свойств нашего активного объекта *Heart* появится строчка **Имя:** x_0 **Тип:** *real*. Переменная eps со значением 0.01 задается так же. Снова выполните проверку синтаксиса: нажмите кнопку **Построить**. В результате на экране вы получите следующее - рис. 2.3.

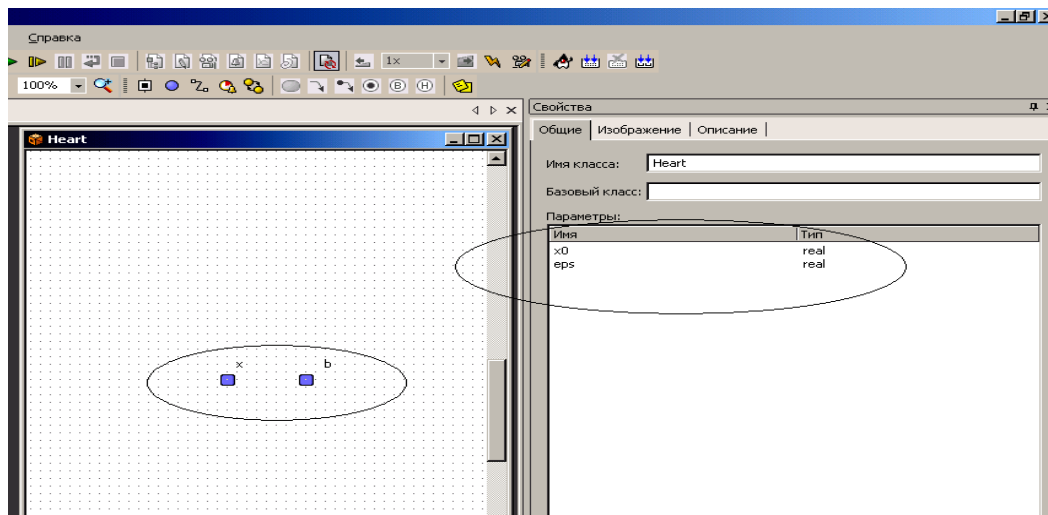


Рис. 2.3. Переменные и параметры модели Heart

Заметьте, что внизу под полем с именем **Параметры** для удобства манипулирования в этом поле со списками параметров помещены шесть кнопок со следующей функциональностью, Рис.2.3:


- Введение нового параметра
- Редактирование выделенного параметра
- Удаление выделенного параметра
- Перемещение выделенного параметра
- Дублирование выделенного параметра



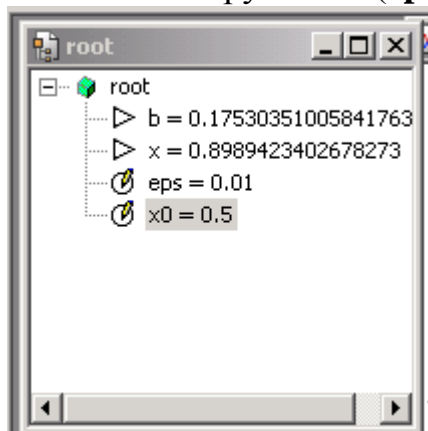
Рис. 2.3. Кнопки


Построение модели закончено.
управления параметрами

2.3. Запуск модели

Щелкнув на кнопке  запуска выполнения модели по шагам, после компиляции в окне наблюдения увидим открытым только окно **root** переменных и параметров с их начальными значениями (рис. 2.4).

Предопределенное имя *root* дано единственному экземпляру единственного корневого класса *Heart*. Переменные в этом дереве помечены треугольниками (**b** и **x**), а константы - кружками (**eps** и **x0**).





Запуск модели  приведет к тому, что переменные *b* и *x* в этом окне начнут изменяться в соответствии с определенными для них уравнениями. Системы дифференциальных и алгебраических уравнений, определенные в проектах AnyLogic, при выполнении модели решаются одним из встроенных численных методов. Сам метод и необходимая точность решения выбираются системой автоматически, если пользователь не изменит предварительные установки в окне свойств объекта *Simulation*. По умолчанию выполнение модели закончится, когда счетчик модельного времени дойдет до 100 (это условие остановки эксперимента также может быть изменено в окне свойств объекта *Simulation*).

закончится, когда счетчик модельного времени дойдет до 100 (это условие остановки эксперимента также может быть изменено в окне свойств объекта *Simulation*).

Рис. 2.4. Окно root модели Heart

AnyLogic позволяет наглядно представить поведение модели, в частности, представить

изменения во времени всех ее переменных. Введем графики изменения переменных x и b .

1. Запустите модель на выполнение по шагам  или нажмите кнопку рестарта модели.
2. Выберите в главном меню команду **Вид | Новая диаграмма** или щелкните кнопку  на панели инструментов.
3. В поле окна появившегося графика перетащите из окна **root** переменную x , нажав на этой переменной левой кнопкой мыши.
4. Второй график постройте так: в контекстном меню переменной b (в окне **root**) выберите вариант **Диаграмма**.
5. Введите еще один график и включите в него обе переменные поочередно.
6. Запустите модель на выполнение. В окнах будут рисоваться графики значений от времени соответствующих переменных.

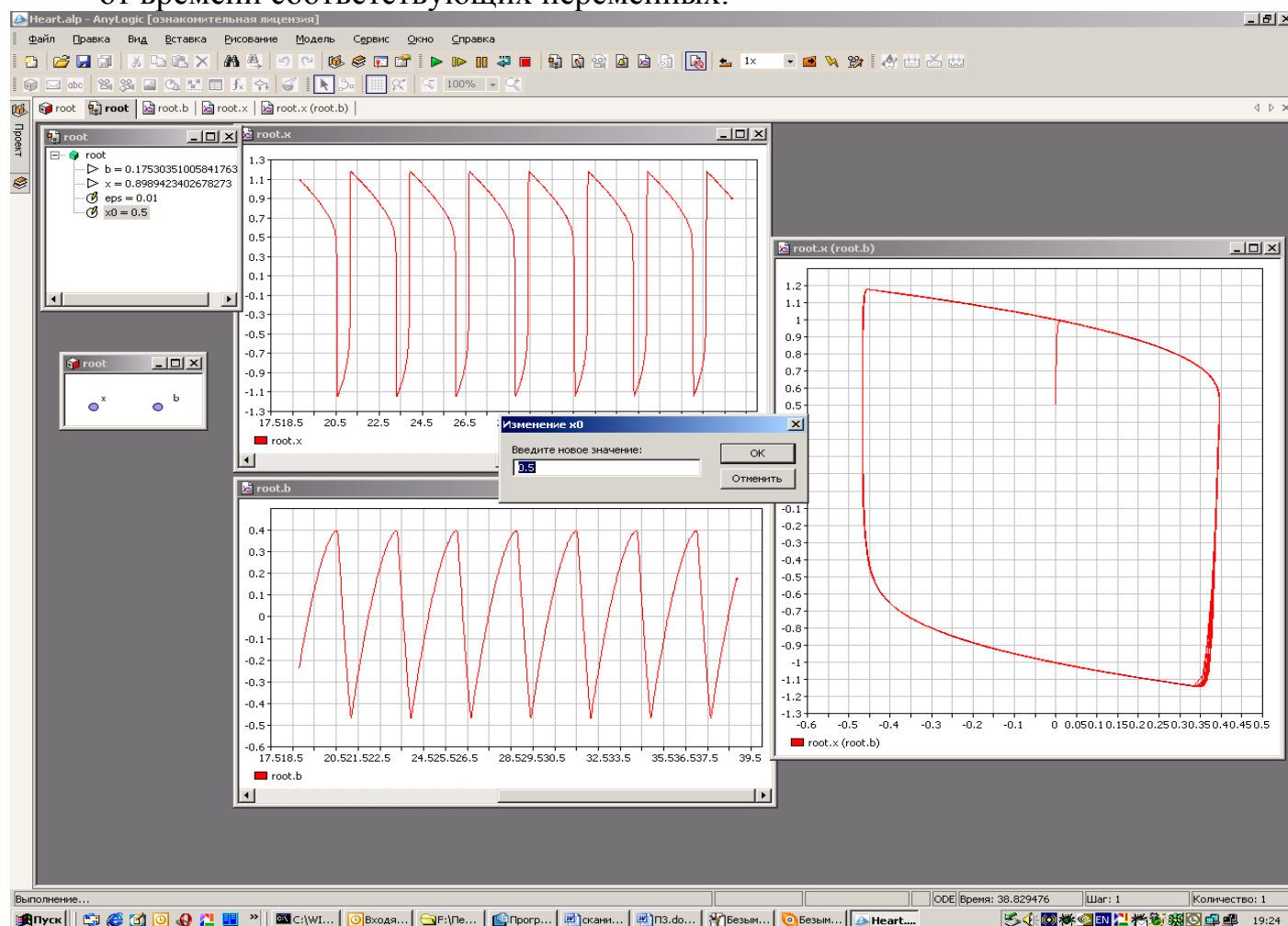


Рис. 2.5. Графики модели Heart


Настройка параметров графиков производится с помощью окна свойств, вызываемого командой **Параметры диаграммы** из контекстного меню данного графика. Контекстное меню графика, как и любого объекта модели AnyLogic, вызывается щелчком правой клавишей мыши, помещенной на этом объекте. Поэкспериментируйте с установкой цвета графиков, возможностью установки опций отображения каждого набора данных на своем графике, размерами отображаемого окна данных, с возможностью отображения фазовой диаграммы, когда по обеим осям графика откладываются значения переменных, и т. п.


Заметьте, что переменные и параметры нашей модели в окне **Содержимое**

диаграммы контекстного меню любого графика имеют имена вида: <имя объекта> и через точку <имя переменной в этом объекте>. Например, ссылка на переменную x здесь записана как *root.x*. Это стандартный прием объектно-ориентированной разработки. Переменные x , b и параметры $x0$ и eps определены как элементы корневого объекта со стандартным именем *root* (см. рис. 2.4), поэтому они имеют здесь соответствующие ссылки.

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

Изменять параметр, как уже говорилось, можно в окне этого параметра, появляющемся в результате двойного клика на нем в окне **root** или при выборе команды **Изменить** контекстного меню данного параметра (рис. 2.5).

Вернитесь в редактор, разрушив скомпилированную модель (кнопкой ). Дважды щелкнув на объекте *Simulation* окна классов проекта, в окне **Свойства** вы можете увидеть и изменить начальные установки компьютерного имитационного эксперимента, который можно выполнить с построенной моделью. На вкладке **Общие** этого окна можно выбрать общие установки - вариант выполнения модели в виртуальном, либо в реальном времени (с заданным соотношением модельного и физического времени), а также изменить именно для этого эксперимента параметры модели. На вкладке **Дополнительные** можно установить условие прекращения выполнения модели, выбрать численный метод решения системы обыкновенных дифференциальных уравнений и его параметры. По умолчанию метод будет выбираться автоматически на основе анализа системы уравнений.

Пользователь может сгенерировать несколько экспериментов различных типов, которые можно выполнить с одной и той же моделью. Щелкните на кнопке  панели инструментов или выберите команду **Новый эксперимент** в контекстном меню объекта *Эксперименты*. В появившемся окне можно выбрать любой из доступных типов экспериментов: **Простой эксперимент**, **Оптимизационный эксперимент** и т. п.

2.4. Анимация модели

Для лучшего понимания динамики модели и наблюдения, развивающихся во времени процессов, в AnyLogic предусмотрена возможность построения анимированного изображения, состоящего из динамических графических элементов. Графические элементы здесь 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.

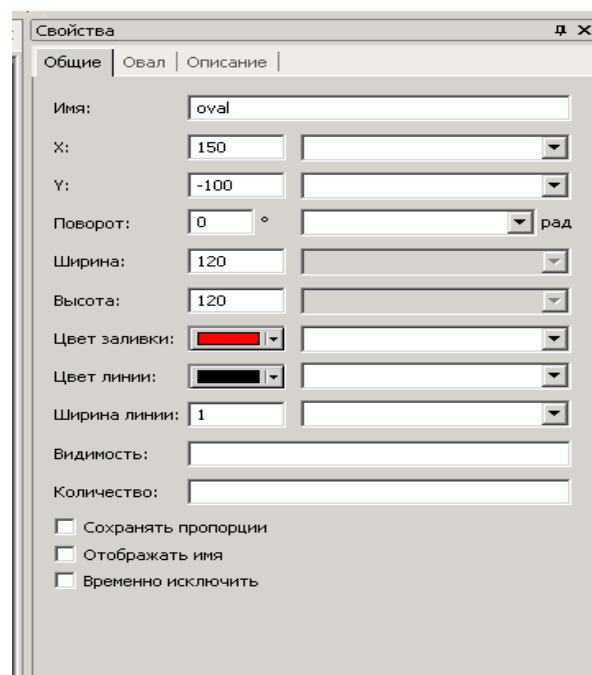
Вернитесь в редактор, разрушив скомпилированную модель и, щелкнув правой кнопкой мыши в окне классов на имени *Heart* в дереве проекта, выберите в появившемся контекстном меню этого объекта левой кнопкой мыши команду **Новая анимация**. После щелчка по кнопке **ОК** в появившемся диалоге, в окне редактора появится окно анимации. Заметьте, что этот объект появился с именем *animation* в дереве проекта как составная часть класса *Heart*. Если окно анимации будет закрыто, открыть его можно двойным щелчком мыши по этому имени в окне классов или выбором команды **Открыть** в его контекстном меню.

Окно анимации представляет собой плоскость с системой координат (X, Y) с шагом нанесенной сетки 10 пикселей. Так же, как единицу модельного времени можно считать любым интервалом реального времени, размер одного пикселя в окне анимации можно ассоциировать с любой единицей длины. Начало координат и их направления отмечены голубыми стрелками в центре поля; эти стрелки не будут видны при анимации. Заметьте, что ось Y направлена вниз.

Штриховой прямоугольник на поле показывает рамку – ту часть поля анимации, которая будет видима при работе модели. Размещение рамки относительно начала координат и ее размеры, можно изменять. Выделите границу рамки (штриховую линию), щелкнув на ней левой кнопкой мыши, и в появившемся окне свойств рамки установите следующие значения координат: $X = -300$ и $Y = -200$. Это координаты левого верхнего угла рамки относительно ее начала координат. Ширину и высоту рамки установите равными 600 и 400 в соответствующих полях окна ее свойств. Этому же положению и размеров можно добиться манипуляциями с рамкой с помощью мыши.

Будем строить анимацию динамики работы сердечной мышцы как изображения круга (овала), радиус которого будет меняться. Этот радиус является функцией от значения переменной x модели. Во-первых, построим изображение овала. Для этого, щелкнув мышью на панели инструментов по кнопке **Овал**, нарисуем в любом месте поля анимации какой-нибудь овал. Справа появится окно **Свойства** этого овала. Свойства овала уже установлены по умолчанию, и мы можем их редактировать. По умолчанию имя этого объекта будет *oval*, координаты X и Y соответствуют месту, куда мы поместили овал в поле анимации, а ширина и высота соответствуют тому, что мы нарисовали.

Если мы запустим модель на выполнение, мы увидим, что в окне анимации этот овал находится неподвижно именно в том месте, которое определено его заданными координатами, и с установленными нами размерами. Как уже говорилось, в AnyLogic



принята следующая концепция: каждая характеристика графического элемента, помещенного в окно анимации, имеет два значения: статическое и динамическое. Статическое значение определяет характеристику (координату, угол поворота, цвет и т. п.) объекта в окне анимации в статике как константу в процессе редактирования. Динамическое определяет значение этой характеристики в процессе выполнения модели и может быть определено как значение любой переменной модели и даже как любая функция от переменных модели. Поэтому обычно характеристика графического объекта имеет два поля в окне свойств объекта: левое поле для статического значения, правое – для динамического значения (рис. 2.6). Если

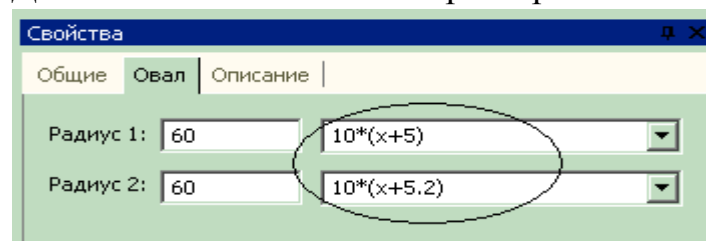
динамическое значение не определено, в динамике характеристика этого графического объекта сохраняет свое статическое значение. Например, выделите штриховую границу поля анимации и заметьте, что в окне ее свойств правые поля характеристик

задать нельзя. Это и понятно: размеры и координаты поля анимации не могут изменяться при выполнении модели.

Рис. 2.6. Окно свойств овала в окне анимации

Выделите овал, который будет представлять динамику бьющегося сердца. Статические значения его характеристик задайте так: в окне свойств овала в левых полях установите координаты центра X и Y равными 150 и -100 соответственно, ширину и высоту по 120, цвет заливки определите бордовым, цвет линии границы - красным; ширину линии границы установите 2.

Динамические значения характеристик овала должны отражать "жизнь" этого объекта




в процессе выполнения модели. Изменение объема сердца будем представлять меняющимся радиусом овала. Радиус овала задается в окне его свойств на вкладке **Овал**. Определите радиусы овала как меняющиеся в за-

висимости от переменной x так, как показано на рис. 2.7.

Рис. 2.7. Динамические свойства овала

Запустите модель. Проведите эксперименты с установкой различных коэффициентов между реальным и модельным временем. Кроме того, можно изменять параметры модели (x_0 и eps) и наблюдать, как изменяется характер сердцебиения.

Целью разработки этой модели является исследование влияния ее параметров на поведение системы. Введем график непосредственно в поле анимации. Щелкнув на

кнопке **Индикатор-диаграмма**  панели инструментов, поместите график в поле анимации и, изменяя мышью положение и размеры данного графического объекта, добейтесь, чтобы в открывшемся окне свойств этого нового объекта на вкладке **Общие** установились статические параметры **X** = 20, **Y** = 50, **Ширина** = 260, **Высота** = 100. Такого же эффекта можно добиться, если данные значения будут установлены непосредственно в соответствующих полях (статических значений параметров) окна свойств этого объекта.


Чтобы связать график с переменными модели, откройте вкладку **Индикатор-диаграмма** окна свойств этого графика и установите в поле **Отображает** имя x , в поле **Размер окна** значение 10, в полях **Минимум** и **Максимум** значения -1.5 и 1.5, Цвет графика (**Цвет индикации**) и цвет надписей у координатных линий (**Цвет шкалы**) выберите по желанию. Запустите модель на выполнение.

Введите в поле анимации прямоугольник так, чтобы координаты X и Y его верхнего левого угла были 0, -200, а ширина и высота были бы 300 и 400. Цвет заливки этого объекта выберите черный. Для того чтобы эта фигура была фоном и не закрывала другие изображения, в контекстном меню данного прямоугольника выполните команду **Перенести назад**.

Другую половину поля анимации залейте каким-либо фоном (введите прямоугольник и заполните его нужным цветом). В этой части поля мы поместим графические элементы управления экспериментом.

Щелкните мышью на кнопке слайдера (**бегунка**) панели инструментов и поместите его в поле анимации с координатами X = -270, Y = 80, с шириной и высотой 240 и 20, соответственно. В поле **Переменная** вкладки **Бегунок** выберите в выпадающем меню из возможных переменных имя eps , а минимальное и максимальное значения, которые можно регулировать слайдером, установите 0.01 и 0.5. Запустите модель.

Двигая слайдер, можно наблюдать изменение характера пульса на графике. В окне **root**, содержащем переменные и параметры модели, можно видеть, как значение параметра *eps* изменяется при движении слайдера. Конечно, имя параметра и его текущее значение удобно отобразить рядом со слайдером. Вернувшись в редактор,

щелкните мышью при активном окне анимации кнопку  на панели инструментов и затем щелкните у левого верхнего угла слайдера. В появившемся у слайдера поле наберите текст *eps* =. С этим текстом связано окно свойств. На вкладке **Текст** этого окна у параметра **Текст** есть два поля: верхнее (для введения статического текста) и нижнее (для введения динамического текста, появляющегося в процессе выполнения модели). В верхнем уже появилась строка *eps* =, набранная нами в поле анимации. Здесь, в окне, ее можно изменить, дополнить и т. п. Но это поле показывает лишь статический текст, который при пустом нижнем поле будет статически отображаться в окне анимации при прогоне модели. Оставьте данный текст без изменения, поле динамических значений также оставьте пустым.

Для представления значения параметра *eps* поместите рядом с текстом *eps* = в поле анимации еще один текст из нескольких символов, например: 123. Выберите его цвет синим. Это также статический текст, но в динамике он может отображать то, что записано в поле динамического значения этого объекта. Поместите туда имя параметра *eps* (рис. 2.8). В процессе выполнения модели на месте текста 123 теперь будет отображаться численное значение параметра *eps*.

Запустите модель и убедитесь, что значение параметра *eps* при изменении его слайдером можно видеть непосредственно у слайдера. Постройте подобный слайдер также для параметра *x0*. Пределы изменения регулирования установите как для *eps*, т.е., от 0.01 до 0.5

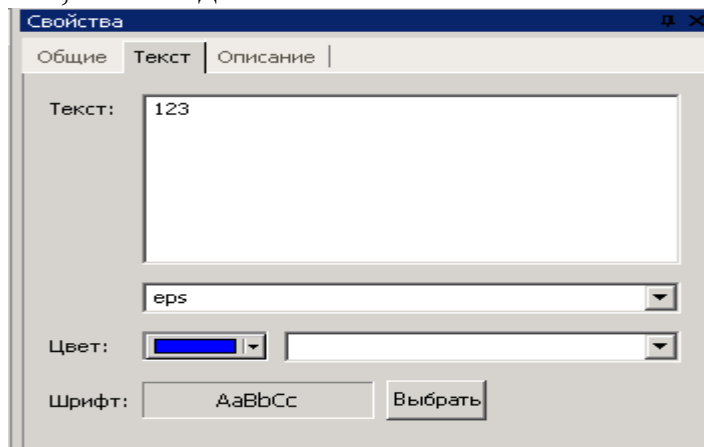



Рис. 2.8. Задание текста в поле анимации

В верхнюю левую часть поля анимации введите поясняющий текст. В редакторе при активном окне анимации щелкните мышью на кнопке  введения текста на панели инструментов и после этого щелкните в поле анимации в верхнем левом углу. Установите в появившемся окне свойств текста координаты (X, Y) этого объекта (-280, -170) на вкладке **Общие**. В поле **Текст** вкладки **Текст** данного окна зададим статический текст *Модель сердца*, который будет отображаться в этом месте поля анимации. Нажав кнопку **Выбрать** окна свойств этого объекта, можно выбрать параметры (стиль, размер) текста. Выберите шрифт **Century Gothic**, стиль **Bold**, размер 22. Следующий текст, содержащий пояснение к модели, введите в поле статического текста с координатами (-290, -120): *Это простая модель динамики*

сердечных сокращений. Модель описывается двумя дифференциальными уравнениями: $dx/dt = (x-x^3-b)/\epsilon$ и $db/dt = x-x_0$. Вы можете менять параметры и наблюдать изменение ритма и формы пульсаций. При значении $x_0 = 0.6$ сердце останавливается. Все другие параметры текста оставьте установленными по умолчанию. Для того чтобы текст уместился в окне анимации по ширине, в нужных местах следует поставить перевод строки.

2.5 Самостоятельная работа по вариантам

1. Поместите рядом с изображением сердца текст с динамическим значением переменной X .
2. Поместите в поле анимации текущее значение времени.
3. Измените анимацию сердца так, чтобы овал сжимался по оси Y и расширялся по оси X и наоборот.
4. Измените анимацию сердца так, чтобы овал, имитирующий сердце вращался вокруг вертикальной оси.
5. Измените анимацию сердца так, чтобы овал, имитирующий сердце вращался вокруг горизонтальной оси.
6. Измените анимацию так, что при сжатии сердце меняло цвет, например, на синий, а при расширении становилось опять красным.

Практикум 3. Системы массового обслуживания: часть 1: использование библиотеки стандартных объектов

3.1. Постановка задачи

Рассмотрим систему массового обслуживания (СМО), моделирующую операционный зал банка, она является типичной для широкого круга систем, моделирующих сервисное обслуживание. Другими примерами СМО являются парикмахерские, телефонные станции, магазины, бензоколонки и т.п. В моделях таких систем типичными объектами являются заявки и обслуживающие их приборы. **Заявки** моделируют клиентов, заказы на выполнение работ, телефонные вызовы, покупателей, автомашины и т. п. **Приборы обслуживания** моделируют кассиров, продавцов в магазине, лифты, линии передачи данных и т. п. Сам процесс **обслуживания** – с точки зрения модели – это просто временная задержка. Каждая СМО состоит из какого-то числа приборов обслуживания и имеет на входе потоки заявок, поступающих обычно в случайные моменты времени. Обслуживание заявки каждым прибором СМО происходит обычно также в течение случайного времени. Различные СМО отличаются характеристиками случайных входных потоков заявок, числом обслуживающих приборов и дисциплиной обслуживания. Системы массового обслуживания являются типичными системами событийно-дискретного типа. **Событием** в СМО является появление в системе очередной заявки, постановка заявки в очередь на обслуживание, начало и окончание обслуживания и т.п.

Из-за однотипности, похожести задач, решаемых моделями систем массового обслуживания, удобно отдельные блоки (генераторы заявок, обслуживающие приборы, очереди и т. п.) реализовать как набор (библиотеку) объектов, из которых может собираться структура конкретной модели и параметры которых можно настраивать в зависимости от характеристик моделируемой системы.

Именно для этих целей в AnyLogic создана библиотека Enterprise Library. Она предоставляет высокоуровневый интерфейс для быстрого создания дискретно-событийных моделей с помощью блок-схем. Построим с помощью элементов библиотеки модель системы, предоставляющей сервисы – операционный зал банка.

3.2. Построение модели

Рассмотрим простую систему, предоставляющую сервисное обслуживание - операционный зал банка. В банке есть два менеджера, отвечающие за два различных типа операций: выдачу инвестиций и работу со счетом. К менеджерам в очереди стоят посетители. После обслуживания менеджером каждый клиент идет в кассу, получая либо сдавая деньги. Очередь в кассу общая.

Клиенты приходят в банк обычно в случайные моменты времени. У каждого клиента свои вопросы по своему счету или по будущему кредиту, поэтому время обслуживания тоже случайно. Оплата в кассе занимает случайное время, поскольку один посетитель может приготовить точную сумму, а другому нужно дать сдачу. Операционный зал банка является типичной системой массового обслуживания (СМО). Такую систему можно представить моделью с небольшим числом типов абстрактных объектов: клиенты представляются заявками на обслуживание, а объекты, выполняющие обслуживание (менеджеры, кассиры), представляются приборами, обрабатывающими заявки. Объекты типа "очередь" имитируют ожидание без обработки.

Структура имитационной модели, которая даст ответ на поставленные вопросы, должна отражать структуру реальной системы массового обслуживания: заявки (клиенты банка) генерируются (входят в систему), становятся в очереди к обслуживающим приборам, а после полного обслуживания покидают систему. Характерной особенностью СМО является стохастическая природа описывающих эти системы характеристик. Структура модели операционного зала банка в данных терминах имеет вид рис. 3.1.

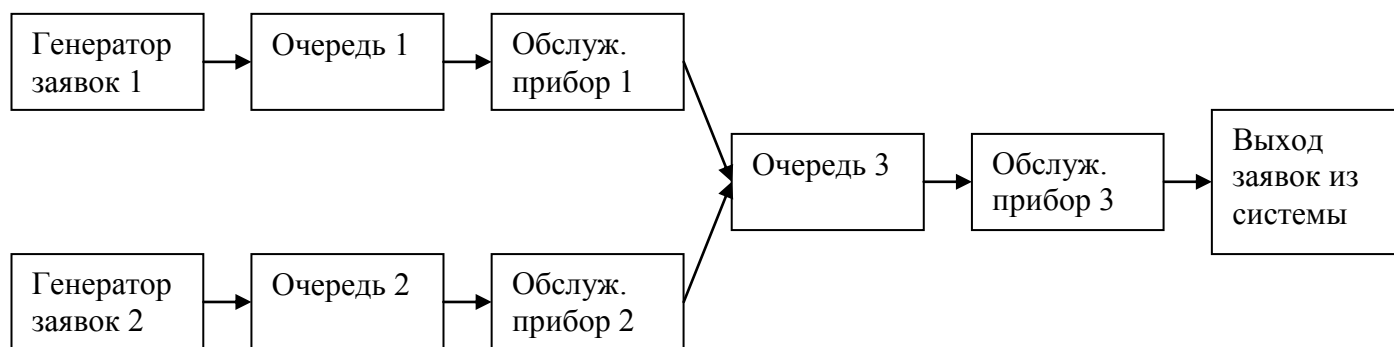


Рис. 3.1. Структура модели операционного зала банка

На рис. 4.1 генераторы заявок имитируют события прихода новых клиентов, которые для модели все одинаковы за исключением того, что каждый из клиентов будет иметь свою собственную историю обслуживания и ожидания в очередях. Поэтому имитирующие клиентов заявки могут хранить такую историю - моменты времени входа в систему, начала и конца обслуживания и т. п. Очереди имитируют реальные очереди людей. Обслуживающие приборы имитируют работу менеджеров и кассира. Ясно, что для оценки качества обслуживания в операционном зале банка не важны ни расположение обслуживаемого персонала, ни даже вид работы, которую выполняет

конкретный служащий. С точки зрения поставленной задачи (анализа эффективности) важно только время обслуживания, и именно этим отличаются различные обслуживающие приборы в этой модели. Поскольку время обслуживания случайно, приборы просто задерживают заявки на некоторый случайный период времени. Блок "Выход заявок из системы" получает заявки и может подсчитать интегральные временные характеристики, описывающие историю обслуживания заявок в системе.

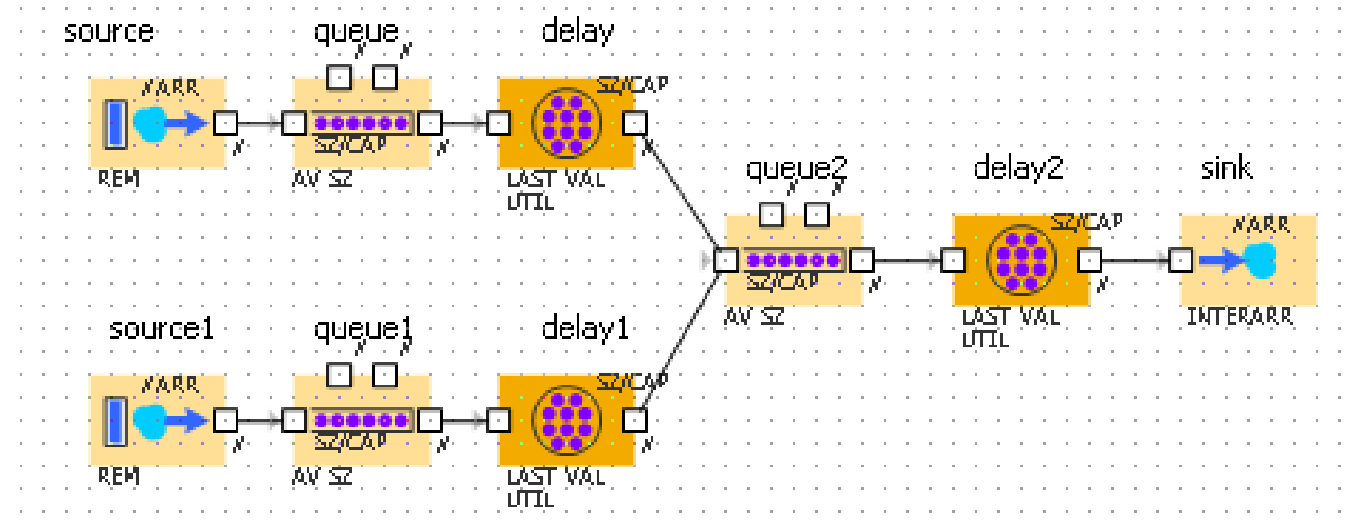
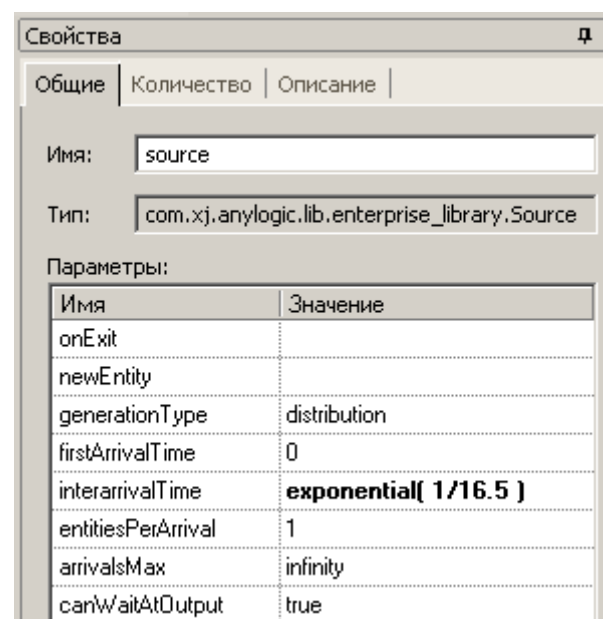


Рис. 3.2. Структура модели банка в среде AnyLogic

Постройте модель операционного зала банка. Создайте новый проект, назовите его, например *Ochered*, (обратите внимание на то, что в названии проекта, как и в именах объектов, можно использовать только латиницу и цифры). Откройте библиотеку *Enterprise Library* и в окно структуры корневого активного объекта методом drag-and-drop внесите из библиотеки следующие элементы – рис. 3.2. Блоки *serverA*, *serverB* и *serverC* - это экземпляры блока *Delay* библиотеки. Соедините блоки нужным образом и запустите модель на выполнение. (Соединяются блоки библиотеки так же, как интерфейсные переменные или порты.)

Настройка параметров включенных в модель блоков из библиотеки тоже не займет много времени. Пусть в соответствии с измерениями (или предположениями) поток клиентов к первому менеджеру является случайным, интервалы времени между клиентами в нем распределены экспоненциально со средним 16.5 мин., поток ко второму менеджеру – тоже экспоненциальный со средним интервалом времени между приходами клиентов 12.5 мин. Выделите блок *source*, и в окне свойств этого объекта в строчке параметра *interarrivalTime* запишите *exponential(1/16.5)*. Заметим, что параметром функции *exponential* (которая при обращении к ней генерирует реализацию случайной величины с экспоненциальным законом распределения) является



интенсивность потока событий, обратная среднему времени между событиями. Остальные параметры этого объекта не изменяйте. На рис. 7.3 видно, что измененные

значения параметров в соответствующих полях свойств блоков выделяются полужирным шрифтом.

Аналогично измените значение параметра *interarrivalTime* у другого генератора потока заявок *source1*.

Пусть время обработки заявок обоими менеджерами и кассиром тоже случайно, и пусть оно распределено экспоненциально со средними значениями 15, 10 и 6.5. Введите соответствующие значения параметра *delayTime* у экземпляров *delay*, *delay1* и *delay2* (рис. 4.4).

Запустите модель на выполнение. Целью моделирования подобной сервисной системы является, конечно, не просто имитация функционирования его служащих и поведения клиентов, а определение тех параметров, которые характеризуют качество сервиса и затраты на обеспечение этого сервиса при определенных характеристиках входного потока клиентов и структуры системы. В частности, средней длины очередей, занятость обслуживающего персонала и т.п.

Окно **root** модели банка после обработки примерно 33 тыс. заявок представлено на рис. 3.3. На нем видно, что 14 397 заявки прошли через первого менеджера (*delay*), 19 104 заявки - через второго менеджера (*delay1*) и 33 500 заявок обработано кассиром. Далее на этом же рисунке видно, что в данный момент времени до генерации очередной заявки объекту *source* осталось 12.64 мин., а *source1* - 6.84 мин., в очереди к первому менеджеру стоит 16 заявок, ко второму - 3 заявки, а к кассиру - 0 заявок. Все эти значения получены без каких-либо усилий со стороны разработчика модели. Они представляются при работе библиотечных блоков автоматически. Блоки библиотеки *Enterprise Library* имеют и другие встроенные средства сбора статистической информации, однако по умолчанию сбор части статистики отключен для повышения производительности.

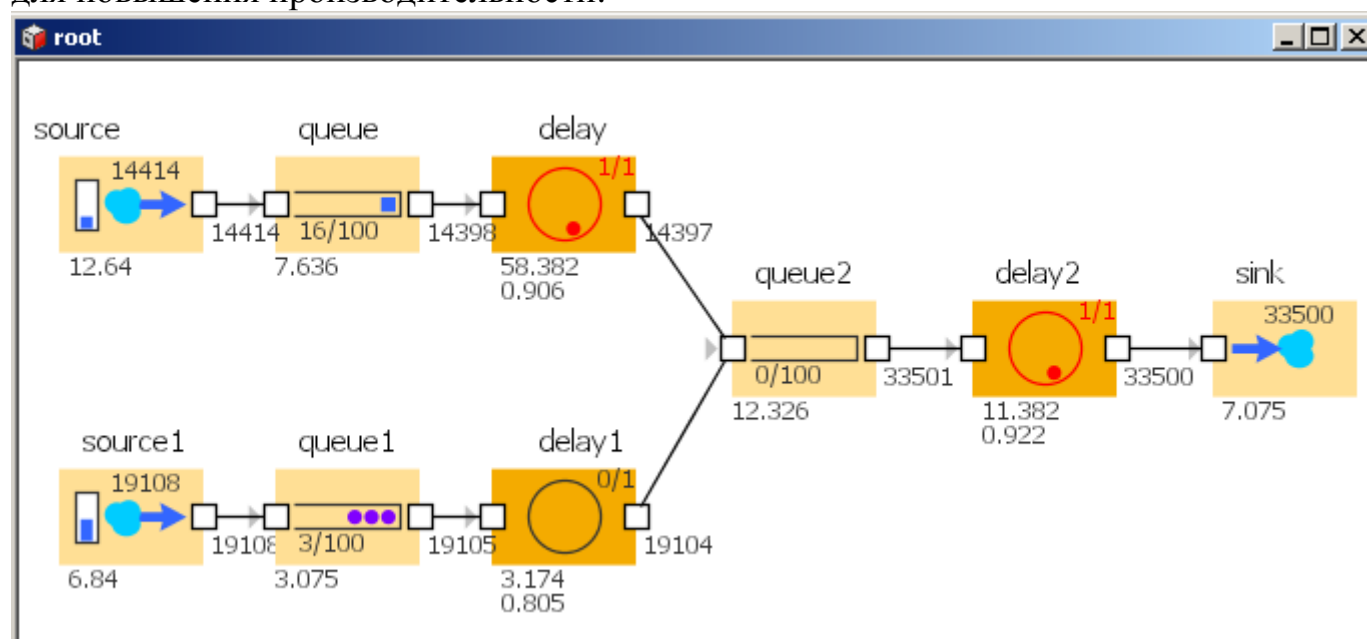


Рис. 3.3. Окно root модели банка

В окне свойств блоков *queue*, *queue1* и *queue2*, а также *delay*, *delay1* и *delay2* в поле значений параметра *statsEnabled* вместо *false* вставьте из выпадающего меню значение *true*. Запустите систему на выполнение опять.

На рис. 3.3 представлено окно **root** модели банка, в которой для экземпляров активных объектов *Queue* и *Delay* включен сбор статистики. В системе собирается статистика по длинам очередей для *Queue* и статистика по коэффициенту использования (доля времени занятости) для блоков *Delay*, представляющих в нашей модели менеджеров. В системе обработано более 33 тысяч заявок. Видно, что средняя длина очереди *queue* равна 7.636, очереди *queue1* равна 3.075, а очереди *queue2* равна 12.326. Коэффициент использования двух менеджеров и кассира соответственно 0.906, 0.805 и 0.922.

На основе этого анализа можно с уверенностью сказать, что при указанных ранее входных потоках клиентов и производительности обслуживания зафиксированная в модели структура операционного зала банка неудовлетворительна. Первый менеджер (которого моделирует блок *delay*) и кассир (блок *delay2*) перегружены, средняя длина очередей к ним 8-12 человек. Клиенты банка будут не удовлетворены.

4.3. Самостоятельная работа по вариантам

1. Перераспределите клиентов между двумя менеджерами более оптимально. Интенсивности входных потоков оставьте прежними, а для их разделения используйте объект *selectOutput*, т.е. от обоих источников заявки должны попасть на вход этого объекта, а с его выходов заявки должны поступить в очереди к менеджерам. Условие разделения: клиент должен идти к тому менеджеру, очередь к которому меньше. Размер очереди можно узнать из переменных *queue.size()* и *queue1.size()*. О правилах использования объекта *selectOutput* прочтите в документации к **EnterpriseLibrary**.
2. То же, что в варианте 2, но пропишите условие разделения потоков клиентов в объекте *selectOutput*, таким образом, чтобы очередь ко второму менеджеру была в 2 раза меньше, чем к первому.
3. Добавьте в модель второго кассира со средним временем обработки заявок 10 минут, минимальным 7 минут, а максимальным 20 минут (для этого используйте треугольное распределение *triangular(7., 10., 20.)*) и доработайте модель, введя объект *selectOutput*, для оптимального распределения клиентов между кассирами. Условие разделения: клиент должен идти к тому кассиру, очередь к которому меньше. Размер очереди можно узнать из переменных *queue.size()* и *queue1.size()*. О правилах использования объекта *selectOutput* прочтите в документации к **EnterpriseLibrary**.
4. То же, что в варианте 3, но пропишите условие разделения потоков клиентов в объекте *selectOutput*, таким образом, чтобы очередь ко второму кассиру была в 3 раза меньше, чем к первому.

Работа 4. Оптимизация: модель сервиса мобильной связи

4.1. Постановка задачи

В этой работе мы будем решать следующую проблему: поставщик сервиса для мобильной связи выбирает оборудование автоматической телефонной станции и задается вопросом, сколько потребуется телефонных каналов для получения максимальной прибыли? Если поставщик сервиса имеет телефонную станцию с малым числом каналов, то у него будет большое число неудовлетворенных абонентов. Напротив, если телефонная станция оборудована слишком большим числом каналов, то стоимость ее обслуживания может перекрыть всю прибыль. Очевидно, есть некоторое оптимальное количество телефонных каналов, зависящее от множества различных факторов.

Сформулируем нашу проблему в случае поставщика сервиса более точно. Пусть правила тарифного плана таковы, что каждый обслуженный телефонный вызов приносит некоторый доход, а за каждый отклоненный вызов поставщик сервиса должен заплатить штраф. Покупка оборудования АТС и содержание каждого канала обходятся в некоторую сумму, зависящую от числа каналов АТС. Варьируя число каналов, можно найти то оптимальное их число, которое принесет максимальный доход. Аналитически проблему эту решить нельзя, поскольку все функции здесь неаналитические. Решение проблемы возможно с помощью оптимизации, которая использует имитационную модель для нахождения значения заданного функционала при конкретных значениях параметров.

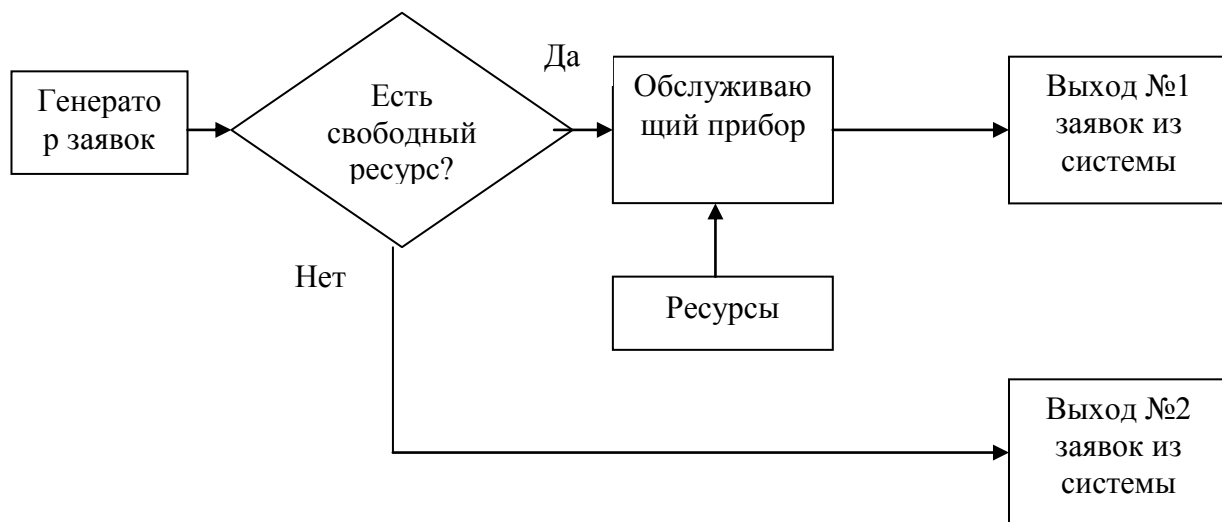


Рис. 4.1. Структура модели оценки прибыли при предоставлении сервиса мобильной связи

На рис. 4.1 представлена структура имитационной модели, с помощью которой решается эта проблема. Генератор заявок имитирует приход вызовов. Обслуживающий прибор в данном случае может обслужить вызов, только если есть свободный ресурс (канал соединения). Блок "Ресурсы" имитирует наличие ограниченного числа ресурсов, это число можно задать параметром. Блок "Анализ" направляет поступившие вызовы либо на обслуживающий прибор, если есть

свободный ресурс (канал), либо на выход из системы при отсутствии свободного канала.

Для того чтобы определить зависимость прибыли от числа N каналов, нужно подсчитать стоимость покупки станции и содержания каналов, при моделировании подсчитать доход за все обслуженные вызовы в течение некоторого времени и штраф за все необслуженные вызовы за это время. Изменяя параметр N имитационной модели, можно для каждого его значения получить возможную прибыль и выбрать наилучшее, оптимальное значение параметра. В AnyLogic встроен оптимизатор OptQuest - алгоритм поиска оптимума неаналитических функций, который автоматизирует описываемую процедуру.

Чтобы настроить оптимизацию в AnyLogic, нужно выполнить следующие шаги:

1. Создать в разработанной модели оптимизационный эксперимент.
2. Задать оптимизационные параметры и области их изменения.
3. Задать условие остановки модели после каждого прогона. Это может быть либо остановка по времени выполнения прогона, либо остановка по условиям, накладываемым на переменные модели.
4. Задать целевую функцию - ту функцию, значение которой отражает предпочтительность вектора исходных факторов. Значение целевой функции должно быть доступно в конце каждого прогона модели, оно будет использоваться оптимизатором.
5. Задать ограничения, если необходимо, которые в конце каждого прогона определяют, допустимо ли значение вектора исходных факторов.
6. Задать условия прекращения оптимизации.

В следующем разделе мы рассмотрим пример оптимизации, в котором все эти шаги конкретизированы.

Будем считать, что за каждый отвергнутый вызов платится штраф (усредненные потери от неудовлетворенности клиента), а обслуженные вызовы приносят доход. Имея стоимость поддерживаемых каналов связи, можно в качестве целевой функции в этой модели выбрать прибыль, определяемую как разность доходов за обслуживание телефонных вызовов и расходов на оборудование, обслуживание станции и штрафы. Прибыль будем подсчитывать в среднем за единицу времени, например за минуту. Она равна доходу минус расходы на оборудование (вместе со стоимостью обслуживания) и штрафы за отказ в обслуживании (все приведенное к стоимости в течение одной минуты).

4.2. Построение модели

Структура модели представлена на рис. 4.1.

Откройте новый проект, назовите его **MobileCom**. Из палитры **Основная библиотека** с помощью мыши перенесите блоки структуры системы в поле редактора структуры модели и соедините их в соответствии с рис. 4.1. Нам нужен один источник заявок (блок source), блок, решающий, отказать или нет пришедшей заявке в обслуживании в

соответствии с некоторым условием (`selectOutput`), блок обработки входных заявок с использованием ресурсов – в нашем случае каналов связи (`service`), блок ресурсов (`resourcePool`) и два блока стока (`sink` и `sink1`).

Очевидно, что поток поступающих заявок и поток обслуживания имеют вероятностную природу. Для простоты будем считать, что они экспоненциальные. Введем в модель три параметра: целый N (число линий связи) и два вещественных λ (интенсивность поступления заявок) и μ (интенсивность их обслуживания каждым каналом). Установим начальные значения для них $N=3$, $\lambda=1.5$ и $\mu=0.5$.

Параметры модели

Для определения характеристик эффективности системы установим следующие параметры для блоков системы.

- **Блок `source`.** У параметра *Заявки прибывают согласно* оставим (по умолчанию), значение *Интенсивности* (в этом случае заявки генерируются в соответствии с экспоненциальным распределением), но установим у этого распределения параметр *Интенсивность прибытия* – λ .
- **Блок `selectOutput`** посылает принятые заявки на свой выход при условии, истинности параметра *Условие*. В нашей модели это условие должно быть истинно только в том случае, когда процессор `service` имеет свободные линии. Число обрабатываемых в этом блоке заявок можно получить, обратившись к функции `busy()`. Итак, в поле параметра *Условие* блока `selectOutput` нужно установить $resourcePool.busy() < N$.
- **Блок `service`.** Здесь параметр *Время задержки* установите как $exponential(\mu)$. Остальные параметры оставьте неизменными. В частности, параметр **Количество ресурсов** определяет число ресурсов на одну входную заявку, он уже установлен в 1, параметр *Вместимость очереди* определяет длину очереди в процессоре, но у нас длина очереди не будет влиять на работу блока (мы определили, что в блок приходит заявка, если только для нее есть ресурс).
- **Блок `resourcePool`** требует установки только одного параметра – **Количество ресурсов**, установите для него значение N .

Запустите модель в режиме виртуального времени. Через несколько секунд некоторые из требуемых характеристик уже появятся в анимированном окне структуры.

Из статистики модели видно, что 65% вызовов обслужено, остальным отказано. Таким образом, относительная пропускная способность системы, т. е. средняя доля пришедших заявок, обслуживаемых системой ~ 65%, а вероятность отказа заявке в обслуживании ~ 35 %.

Однако статистические характеристики системы играют лишь вспомогательную роль при анализе систем. Действительной целью моделирования является анализ возможной прибыли при различных значениях вектора входных параметров системы.

В нашей задаче единственным таким параметром является N - число поддерживаемых каналов связи.

Проблема оптимизации. Целевая функция

Введем понятие *Тарифного плана* - аналога *Соглашения о качестве сервиса* (Service Level Agreement), которое широко используется в современных системах предоставления сервиса для расчета стоимости сервиса. В таком соглашении обычно оговариваются как стоимость сервиса при его надлежащем качестве, так и штраф поставщика сервиса за низкое качество обслуживания. Пусть в соответствии с *Тарифным планом* каждая обслуженная заявка приносит поставщику сервиса некоторый доход в соответствии с правилом посекундной тарификации после первой минуты, а за каждую отклоненную заявку поставщик сервиса должен заплатить штраф. Покупка оборудования АТС и содержание каждого канала обходятся в некоторую сумму, зависящую от максимального числа каналов АТС. Прибыль, полученную владельцем АТС, можно вычислить как доход за обслуживание вызовов минус штраф за отвергнутые вызовы и расходы на оборудование:

$$\text{прибыль} = \text{ДОХОД} - \text{штраф} - \text{стоимость оборудования}$$

Увеличение числа каналов, обеспечивающих связь, уменьшает штраф и повышает доход, но при этом растут и расходы на оборудование. Варьируя число каналов при конкретных значениях остальных параметров системы, можно найти то оптимальное число каналов, которое принесет максимальную прибыль. Именно значение прибыли в этой задаче играет роль значения функции полезности – целевой функции.

Доход от каждой обслуженной заявки зависит от времени соединения (например, поставщик сервиса хочет объявить повременную тарификацию после первой минуты). Отнеся суммарный доход (назовем его в модели *Income*) к временному периоду моделирования, получим доход в единицу времени (назовем его *gain*). Доход в единицу времени является случайной величиной, мы будем использовать его среднее значение.

Рассмотрим теперь расходы. Они состоят из штрафов за отвергнутые вызовы и затрат на оборудование. Выплачиваемый штраф нужно подсчитывать для всех отвергнутых вызовов. Пусть *penalty* - средний штраф в единицу времени по всем отвергнутым вызовам. Это также случайная величина, среднее значение которой будет использоваться при поиске минимума целевой функции.

Наборы данных

В нашей задаче полученный в каждый момент времени средний доход *gain* так же, как и многие другие значения в моделях систем массового обслуживания (длина очереди и время нахождения каждой заявки в очереди в некоторый момент времени и т. п.), будет только конкретной реализацией соответствующих величин, имеющих стохастическую природу. В общем случае все параметры производительности систем, функционирующих в условиях неопределенности, являются стохастическими. При анализе таких систем представляют интерес не конкретные значения этих параметров,

которые являются случайными величинами, а статистические характеристики всего набора реализаций таких случайных величин - среднее, максимальное и минимальное значения, доверительный интервал и т. п.

Для удобства работы с такими наборами данных AnyLogic имеет ряд инструментов, которые собраны в палитре **Статистика**. Понятно, что случайные величины gain и penalty в нашей модели должны быть вневременными (не зависящими от времени) дискретными наборами данных.

Для сбора реализаций случайных величин gain и penalty введите два элемента **Статистика** из палитры с тем же названием. Кроме имени никаких других параметров в окне свойств этих объектов менять не надо (по умолчанию они дискретны).

В нашей модели сделаем следующие добавления. Все цены будем считать в у. е., единица модельного времени будет соответствовать одной минуте реального времени.

Для подсчета длительности состоявшегося телефонного разговора введем новый Java-класс сообщений, который назовем **Call**. Этот тип будет наследоваться от базового класса Entity и иметь два вещественных поля: **tStart** и **tFinish** (рис. 4.2.). Для создания класса щелкните правой кнопкой по окну **Проекты** и выпадающем меню выберите > **Создать** > **Java-класс**. В окне нового класса в поле **Имя** введите **Call**, в поле **Базовый Класс** выберите **Entity**. Щелкните по полю **Далее** и введите два параметра типа **double**: **tStart** **tFinish** (см. рис.4.2)

В блоке **service** параметр **Действие при входе** определим как:

```
((Call)entity).tStart=time();
```

а, параметр **Действие при выходе** как:

```
((Call)entity).tFinish=time();
```

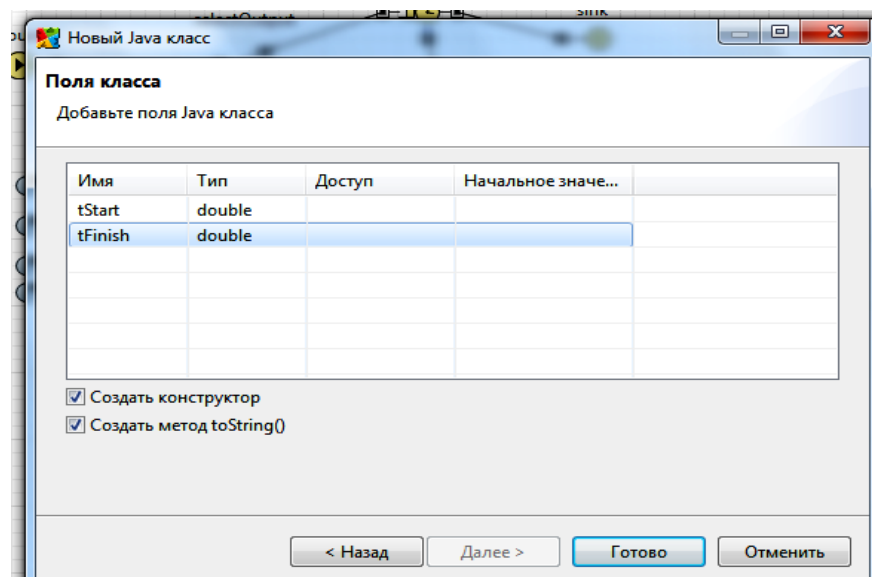


Рис. 4.2. Новый тип сообщения

Во всех блоках Enterprise Library имя текущего объекта (сообщения) по умолчанию установлено как *entity*. Для того чтобы пользоваться элементами структуры текущего сообщения (здесь - данными *tStart* и *tFinish*), необходима операция явной конверсии типов (*Call*)*entity*), указывающая, что этот объект имеет тип *Call*. Операция кастинга имеет меньший приоритет, чем точка, поэтому выражение это нужно взять в скобки, чтобы затем использовать "точечную" операцию доступа к данным этого объекта.

Теперь после обработки каждая заявка будет хранить момент начала и момент конца разговора. Эти значения нужны для того, чтобы начислить плату за обслуживание заявки.

Подсчет прибыли

Зададим теперь правило расчета за состоявшиеся телефонные переговоры. Его удобно определить алгоритмической функцией. Перетащите кнопку **F (Функция)** из основной палитры в модель и назовите функцию *callPrice*. У нее должен быть один вещественный параметр типа *double*, который мы назовем *t* (вкладка: *Аргументы функции*). Саму функцию на вкладке *Код* определите так:

```
return t<=1? minPrice : t*minPrice;
```

Функция *callPrice(t)* определяет стоимость разговора так: если длительность *t* разговора не больше одной минуты, то взимается плата за одну минуту в размере *minPrice*, если больше, то *t*minPrice*. Параметр *minPrice* (цена минуты разговора) задайте как вещественный параметр модели со значением 0.12.

Введите в модель вещественную переменную *Income*, суммирующую полученный с течением времени доход от предоставления соединения (с начальным значением 0). Каждый раз, как обслуженная заявка попадает в блок *sink*, величина *Income* должна возрастать на величину:

```
callPrice(((Call)entity).tFinish - ((Call)entity).tStart );
```

Как только получено новое значение суммарной величины дохода *Income*, можно вычислить новую реализацию среднего дохода *gain*, которая равна *Income/time()*. Поэтому в поле с именем *Действие при входе* вкладки *Основные* блока *sink* следует вставить следующие операторы:

```
Income += callPrice( ((Call)entity).tFinish - ((Call)entity).tStart ); gain.add (Income/time() );
```

Штраф за необслуженные вызовы

В соответствии с Тарифным планом за каждый необслуженный вызов поставщик сервиса платит фиксированный штраф. Примем значение этого штрафа 1 у. е. Подсчитаем сумму штрафа за все пропущенные вызовы. Введите новый параметр модели *penaltyPercall* (штраф за необслуженный вызов) со значением 1.0. Кроме того, введем в модель переменную *Penalties* для подсчета полной суммы штрафа с

начальным значением 0. Для получения всей суммы штрафа каждый раз, как только заявка отбрасывается, следует увеличить суммарное значение штрафа **Penalties** на величину **penaltyPercall**, а также добавить в набор данных **penalty** ее новую реализацию, равную: **Penalties /time()** .

Все необслуженные заявки в модели направляются в блок **sink1**. Для подсчета штрафа нам не нужны никакие параметры новой пришедшей в этот блок заявки, важен лишь сам факт прихода заявки. Поэтому для учета штрафа в поле с именем *Действие при входе* вкладки **Основные** **sink1** вставьте операторы:

Penalties += penaltyPercall; penalty.add(Penalties/time());

Напомним, что переменная **Penalties** принимает значение общего штрафа, в то время как набор данных **penalty** – это случайная величина, равная среднему штрафу за единицу времени.

Приведенная стоимость оборудования

Определим теперь функцию зависимости затрат на оборудование от числа каналов. Отнеся затраты на оборудование к периоду морального устаревания оборудования (например, 5 лет), получим значение приведенных капитальных затрат.

Пусть для организации АТС может быть закуплено оборудование разных типов, так что стоимость поддержания одного канала зависит от числа каналов сложным образом. Пусть простая АТС, обслуживающая до 10 каналов одновременно, вместе с затратами на поддержку каналов в течение 5 лет стоит 0.012 у.е. в пересчете на 1 минуту, если использовать все 10 каналов, но только 0.01 у.е, если использовать 1 канал. Более продвинутая АТС, обслуживающая до 50 каналов, стоит 0.05 у.е. (в пересчете на 1 минуту обслуживания в течение 5 лет), но если использовать только 11 ее каналов, стоит 0.013 у.е. и т. п.

Учет затрат на оборудование и его обслуживание, зависящих нелинейным образом от числа каналов **N**, выполним с помощью табличной функции. Назовем функцию, дающую значение затрат на оборудование от числа каналов, приведенное к минуте времени, **equipmentPrice**. Зададим ее таблично. Для этого выберите из палитры **Основная** вкладку **Табличная функция** и назовите ее ***equipmentPrice***.

В появившемся окне **Основные** этого объекта ***equipmentPrice***. определите функцию **equipmentPrice** согласно рис. 4.3.

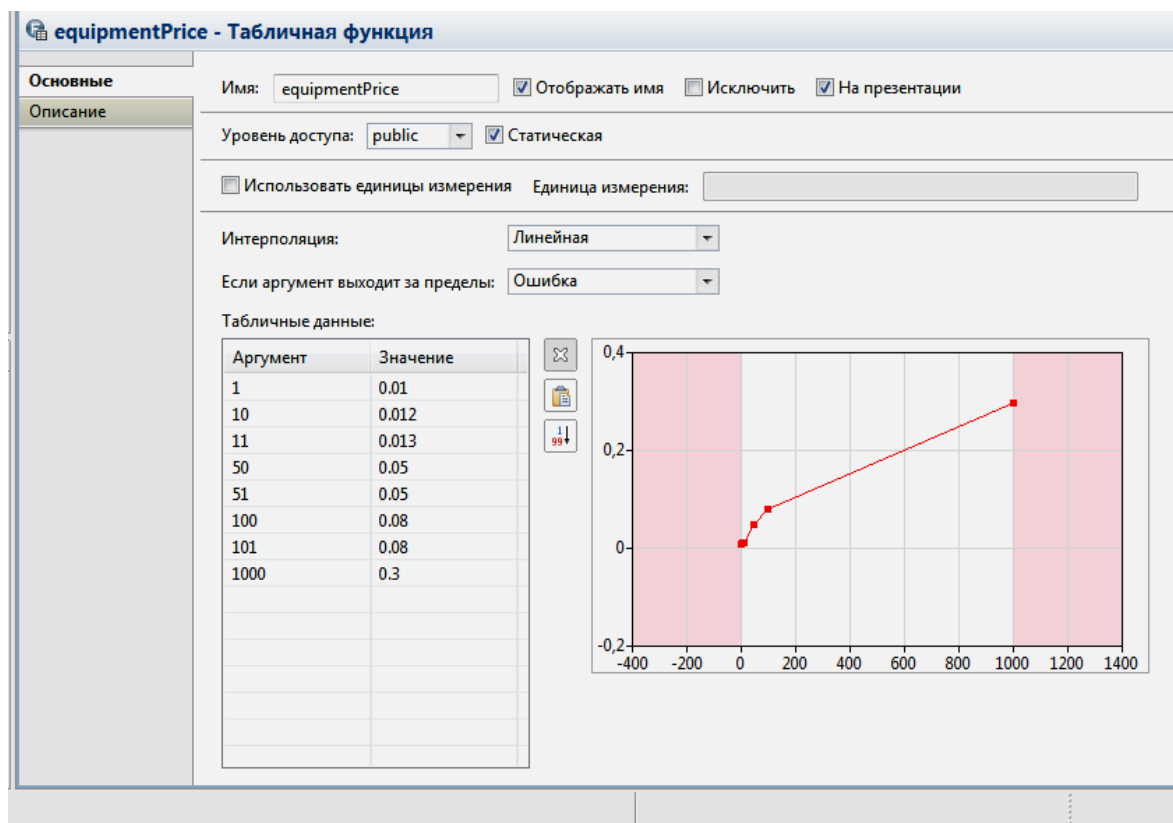


Рис.4.3 Функция оценки стоимости оборудования

Расходы на оборудование при числе каналов N , приведенные к единице времени, теперь можно подсчитать вызовом функции $\text{equipmentPrice}(N)$.

Целевая функция

Средний доход в нашей модели определяется случайной величиной gain , средний штраф определяется случайной величиной penalty , стоимость оборудования, уже приведенная к единице модельного времени, определяется функцией $\text{equipmentPrice}(N)$. Для подсчета прибыли (в единицу времени) введите в модель вспомогательную переменную ***BenefitMean*** (среднюю прибыль) из палитры **Системная динамика**, определяемую формулой так:

$$\text{gain.mean()} - \text{penalty.mean()} - \text{equipmentPrice}(N)$$

Здесь $\text{gain.mean}()$ и $\text{penalty.mean}()$ - математические ожидания величин gain и penalty . Создайте связь функции с параметром N .

Запустите модель. Можно видеть, что прибыль в единицу времени (переменная BenefitMean) при установленных параметрах отрицательна. Измените значение N , установив его 20. Прибыль стала положительной, порядка, 0.375 у.е. в минуту. При числе каналов N , равном 100, прибыль становится меньше, порядка 0,3 у.е. в минуту. Следовательно, существует оптимальное число каналов, обеспечивающее максимальную прибыль.

Для нахождения оптимального числа каналов, дающего максимальную прибыль, можно либо перебрать все значения N от 1 до, например, 100 в эксперименте для

варьирования параметров, но можно и использовать эксперимент с оптимизацией. Оптимизационный эксперимент в AnyLogic позволяет найти такие значения параметров модели, при которых обращается в минимум или максимум некоторая определенная пользователем целевая функция. Значения целевой функции подсчитываются в AnyLogic каждый раз по окончании очередного выполнения модели, и алгоритм оптимизации автоматически выбирает новые значения параметров для очередного запуска модели.

Оптимизация в AnyLogic реализована с использованием встроенного пакета OptQuest. Для оптимизации пользователь должен в соответствующих окнах задать функционал, который следует минимизировать либо максимизировать, задать параметры и ограничения их диапазона, в которых должна выполняться оптимизация, а также указать ограничения, определяющие класс допустимых решений. Затем пользователь должен запустить оптимизацию, и пакет OptQuest будет использовать метаэвристику рассеянного поиска для выбора очередных значений входных параметров на основании значений целевой функции, полученных на предыдущих прогонах модели.

Щелкните по кнопке **Создать** панели инструментов и в появившемся окне выберите **Оптимизационный эксперимент**. В окне свойств эксперимента установите в поле **Целевая функция:** **root.BenefitMean** в качестве целевого функционала и поставьте флажок **Максимизировать**. Изменяемым параметром является **N** (вкладка параметры), из выпадающего меню выберите *Тип: дискретный*, установите нижнюю границу 3, верхнюю 100, число прогонов для нахождения оптимального числа каналов сделайте 100 (параметр **Количество итераций**), сбросьте флажок в поле **Автоматическая остановка**, это позволит программе остановить эксперимент до истечения числа итераций, если изменения целевой станут незначительными (рис. 4.4). Щелкните по кнопке **Создать интерфейс** – программа создаст стандартный интерфейс для отслеживания результатов оптимизационного эксперимента (диаграммы интерфейса должны появиться в окне эксперимента).

Имя: Optimization Корневой класс модели: Main ☐ Исключить

Создать интерфейс

Генератор случайных чисел:

☒ Случайное начальное число (уникальные "прогоны")

☐ Фиксированное начальное число (воспроизводимые "прогоны") Начальное число: 1

☐ Нестандартный генератор (подкласс класса Random): new Random()

Целевая функция: ☐ минимизировать ☒ максимизировать

root.BenefitMean

Условия остановки оптимизации

☒ Количество итераций: 1000

☒ Автоматическая остановка

Параметры:

Параметр	Тип	Значение			
		Мин.	Макс.	Шаг	Начальное
N	дискретный	3	100	1	3
lambda	фиксированный	1.5			
mu	фиксированный	0.5			
minPrice	фиксированный	0.12			
penaltyPercall	фиксированный	1.0			

Рис. 4.4. Параметры оптимизационного эксперимента

Запустите модель. Оптимизационный эксперимент найдет наилучшее число каналов, максимальную прибыль в минуту.

Самостоятельная работа по вариантам

1. Измените программу так, чтобы оптимальное число каналов рассчитывалось при использовании тарифного плана с посекундной оплатой.
2. Рассчитайте прибыль поставщика сервиса при оптимальном количестве каналов связи и оптимальной стоимости 1 минуты разговора при условии, что средняя длительность разговора μ обратно пропорциональна стоимости разговора minPrice согласно формуле: $\mu = 0,06 / \text{minPrice}$.
3. Рассчитайте оптимальное число каналов, если в системе обслуживания вызовов задействовать очередь в объеме 2 абонентов (нужно изменить условие отклонения заявок).
4. Рассчитайте оптимальное число каналов связи при условии, что средняя длительность разговора μ обратно пропорциональна стоимости разговора minPrice согласно таблице:

μ	minPrice
0,5	$\geq 0,12$
1,5	0,06
5,0	$\leq 0,03$

Для решения этой задачи следует помимо оптимального числа каналов найти оптимальную стоимость минуты разговора.

5. Рассчитайте оптимальное число каналов связи при условии, что средняя

интенсивность входящих вызовов λ обратно пропорциональна стоимости разговора \minPrice согласно таблице:


λ	\minPrice
0	$\geq 0,2$
1,5	0,12
20	$\leq 0,03$

Для решения этой задачи следует помимо оптимального числа каналов найти оптимальную стоимость минуты разговора.

- Найдите оптимальное число каналов для случая, когда стоимость АТС в пересчете на 1 минуту обслуживания в течение 5 лет составляет 0,1 у.е., и не зависит от емкости самой АТС.
- Рассчитайте оптимальное число каналов при использовании тарифного плана «каждая вторая минута – бесплатно».
- Переделайте модель таким образом, чтобы исключить из нее набор данных `gain` (для этого нужно предусмотреть вычисление величин `gain.mean`, `gain.add`).
- Переделайте модель таким образом, чтобы исключить из нее набор данных `penalty` (для этого нужно предусмотреть вычисление величин `penalty.mean`, `penalty.add`).
- Найдите оптимальное число каналов при условии, что ни одна из заявок не отклоняется (они все ставятся в очередь), но стоимость оплаты разговора (\minPrice) меняется обратно пропорционально времени ожидания в очереди по закону: $\minPrice = 0,12/(1+0,5T_q)$, где T_q – время ожидания в очереди (в минутах).


Работа 5. Агентное моделирование. Диффузия Басса

1. Создание модели

Давайте создадим простейшую агентную модель, которая дает возможность изучить влияние рекламы на продажи товаров. Модель предложена Бассом в 60-х годах прошлого века и получила название «Диффузия Басса». Для создания модели мы воспользуемся **Мастером**. Щелкните мышью по кнопке панели инструментов **Создать** . Появится диалоговое окно **Новая модель**.

- Задайте имя новой модели. В поле **Имя модели** введите Bass Diffusion Agent Based.
- Выберите каталог, в котором будут сохранены файлы модели. Если Вы хотите сменить предложенный по умолчанию каталог на какой-то другой, Вы можете ввести путь к нему в поле **Местоположение** или выбрать этот каталог с помощью диалога навигации по файловой системе, открывающегося по нажатию на кнопку **Выбрать**.
- Щелкните мышью по кнопке **Далее**. Откроется вторая страница **Мастера создания модели**. Здесь Вам будет предложено выбрать шаблон модели, на базе которого Вы будете разрабатывать Вашу модель. Поскольку мы хотим создать новую агентную модель, установите флажок **Использовать шаблон модели** и выберите **Агентная модель** в расположенном ниже списке **Выберите метод моделирования**.

4. Щелкните мышью по кнопке **Далее**. Откроется следующая страница **Мастера создания модели**. Поскольку первым шагом при создании агентной модели всегда является создание агентов, то здесь Вам как раз предлагается задать имя класса агента и количество агентов, которое будет изначально создано в нашей модели. Задайте в качестве имени класса *Person*. и введите в поле **Начальное количество агентов** 500. Автоматически в нашей модели будет создано 500 агентов (то есть, экземпляров класса *Person*, каждый из которых будет представлять отдельного агента).
5. Щелкните мышью по кнопке **Далее**. Откроется следующая страница **Мастера создания модели**. Здесь Вам будет предложено задать свойства пространства, в котором будут обитать агенты, а также выбрать фигуру анимации агента.
6. Установите флажок **Добавить пространство** и выберите ниже тип этого пространства: **Непрерывное 2D**. Здесь же Вы можете задать размерности этого пространства: давайте введем в поле **Ширина** 600, а в поле **Высота** 350. Тем самым, в результате наши агенты будут располагаться каким-то образом в пределах непрерывного пространства, отображаемого на презентации моделей областью размером 600*350 пикселей.
7. Не меняйте значения, выбранные в выпадающих списках **Начальное расположение** и **Анимация**: пусть агенты изначально расставляются по пространству случайным образом, а анимируются с помощью фигурки человечка.
8. Щелкните мышью по кнопке **Далее**. Откроется следующая страница **Мастера создания модели**. Здесь Вам будет выбрать, хотите ли Вы, чтобы была задана сеть взаимосвязей агентов, и если да, то каким образом должны устанавливаться связи между агентами.
9. Установите флажок **Использовать сеть** и оставьте выбранной опцию **Случайное**.
10. Щелкните мышью по кнопке **Далее**. Откроется последняя страница **Мастера создания модели**. Установите на ней флажок **Добавить простое поведение**. Тем самым, у нашего агента будет создана диаграмма. Мы закончили конфигурирование шаблона создаваемой модели. Щелкните мышью по кнопке **Готово**, чтобы закончить процесс создания модели.

Наша модель будет содержать созданные **Мастером создания модели** классы активных объектов *Main* и *Person*. Активные объекты являются основными строительными блоками модели AnyLogic. Активные объекты могут моделировать любые объекты реального мира: машины, людей, станки, здания, аппаратное обеспечение и т.д. В нашем случае активный объект *Person* будет моделировать агентов (людей). В панели **Проекты** такой класс отображается значком .

2. Моделирование продаж под влиянием рекламы

Изменим модель следующим образом: пусть наша модель моделирует процесс приобретения нового продукта, но пока только под влиянием рекламной кампании, проводимой с целью вывода нового продукта на сложившийся рынок.

В этой модели интенсивность рекламы и вероятность того, что продукт будет приобретен под ее влиянием, полагаются постоянными. Поэтому зададим

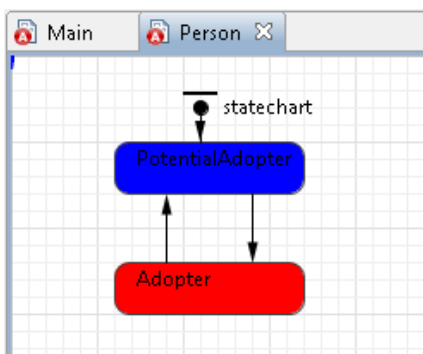
эффективность рекламы константой. Эффективность рекламы определяет, какая доля людей купит продукт вследствие ее влияния.

1. Откройте диаграмму класса *Person*, сделав двойной щелчок мышью по элементу *Person* в панели **Проекты**.
2. Перетащите элемент **Параметр** из палитры **Основная** на диаграмму класса
3. Измените имя параметра. Введите AdEffectiveness в поле **Имя**.
4. В поле **Значение по умолчанию** введите 0.011.

Поведение агента обычно описывается визуально в классе этого агента (в нашей модели это класс *Person*) с помощью *диаграммы состояний*.

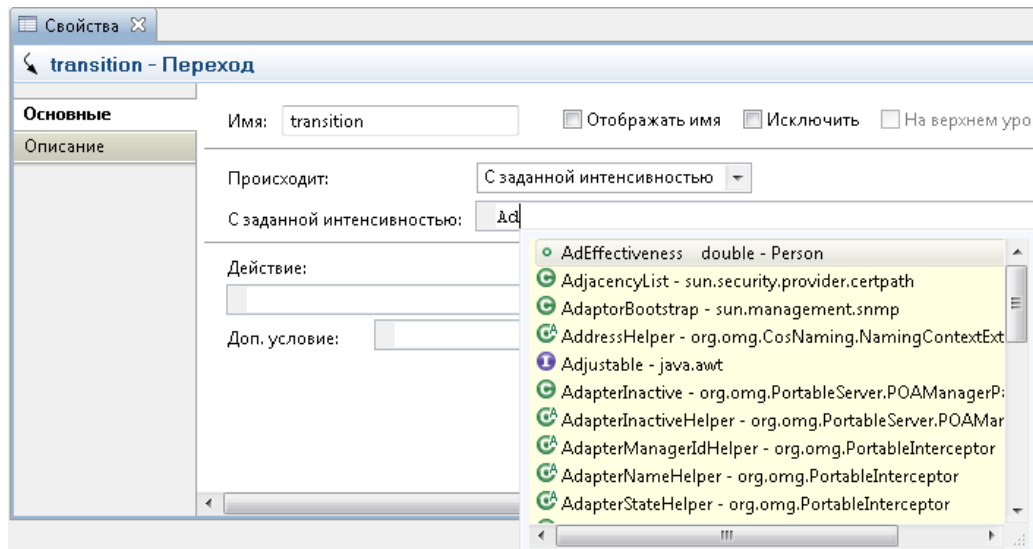
Измените диаграмму состояния

1. Откройте диаграмму класса *Person*, сделав двойной щелчок мышью по элементу *Person* в панели **Проекты**.
2. Измените имя верхнего состояния на *PotentialAdopter* (поле **Имя** на странице свойств перехода) Это начальное состояние, о чем свидетельствует элемент *Начало диаграммы состояний*, направленный в это состояние. Если диаграмма состояний будет находиться в этом состоянии, то это будет означать, что этот человек еще не купил продукт.
3. Назовите его *Adopter*. Если это состояние диаграммы будет активным, это будет означать, что этот человек уже купил продукт.
4. В нашей модели состояние *Adopter* должно становиться активным в момент приобретения агентом продукта. Процесс приобретения продукта этим человеком моделирует *переход*, ведущий из верхнего состояния в нижнее. Нам нужно изменить его свойства, чтобы он срабатывал в нужный нам момент времени.
5. Время, через которое человек купит продукт, экспоненциально зависит от эффективности рекламы продукта. Поскольку время, необходимое человеку, чтобы принять решение о покупке продукта экспоненциально зависит от подверженности этого человека влиянию рекламы, то выберите из выпадающего списка **Происходит С заданной интенсивностью** и введите в поле свойства **Интенсивность** этого перехода имя созданного нами только что параметра AdEffectiveness.



6. Введите AdEffectiveness в расположенном ниже поле **Интенсивность**. Чтобы не печатать полностью имена функций и переменных в формулах, можете

пользоваться **Мастером подстановки кода**. Чтобы открыть **Мастер**, щелкните мышью в том месте поля (в нашем случае - поля **Интенсивность**, куда Вы хотите поместить имя, а затем нажмите Ctrl+пробел. Появится окно **Мастера подстановки кода**, перечисляющего переменные модели и функции, доступные в текущем контексте. Прокрутите список к имени, которое Вы хотите вставить, или введите первые буквы имени, пока оно не будет выделено в списке. Двойным щелчком мыши по имени добавьте его в поле формулы.

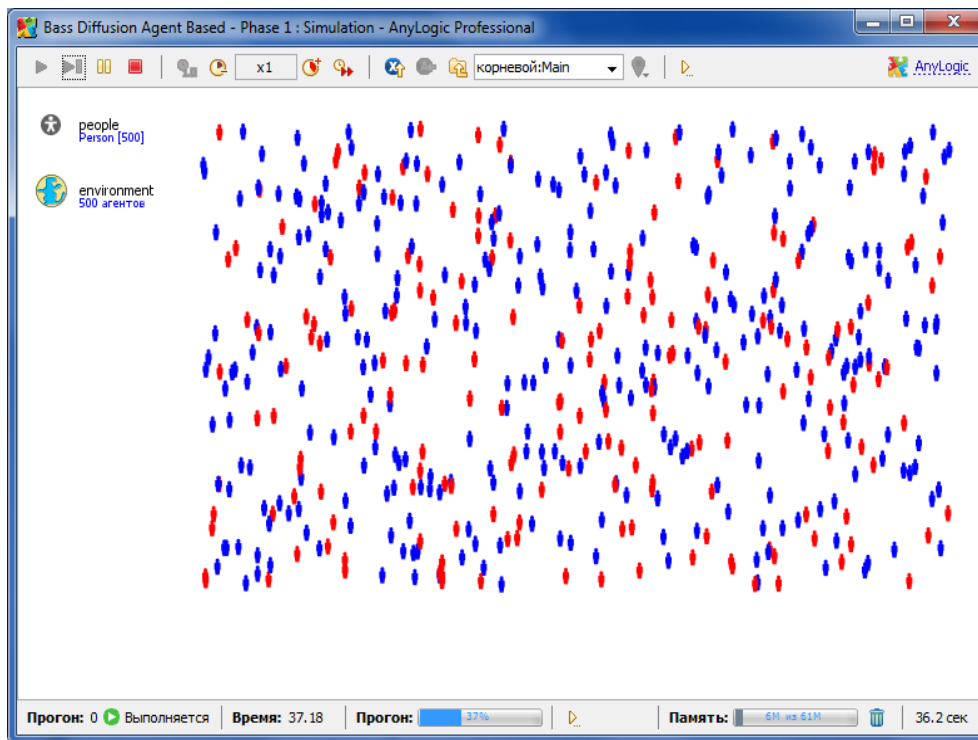


7. Удалите переход, ведущий из нижнего состояния в верхнее, поскольку мы пока создаем простейшую модель, в которой человек, однажды приобретший продукт, навсегда остается его потребителем, и соответственно перехода из состояния *Adopter* в состояние *PotentialAdopter* пока что быть не должно. Чтобы удалить переход, выделите его на диаграмме и нажмите Del.

Если мы сейчас запустим модель, то она будет работать бесконечно. Поскольку мы хотим наблюдать поведение модели только тогда, когда происходит процесс распространения продукта, нам нужно остановить модель, когда система придет в точку равновесия. Поскольку под единицей модельного времени мы будем понимать один год, а процесс распространения продукта в этой модели длится примерно 8 лет, то нам нужно будет остановить модель после 8 единиц модельного времени. Для этого:

1. В панели **Проекты**, выделите эксперимент Simulation:Main щелчком мыши.
2. На странице **Модельное время** панели **Свойства**, выберите **В заданное время** из выпадающего списка **Остановить**. В расположенном ниже поле введите 8. Модель остановится после того, как истекут 8 единиц модельного времени.

Запустите модель. Примерная картина того, что Вы увидите, приведена на рисунке ниже.



Главная задача модели распространения продукта – изучение того, как быстро люди покупают новый продукт. Поэтому сейчас мы добавим возможность отслеживания того, сколько людей уже купило продукт, а сколько – еще нет. Мы будем подсчитывать число потребителей и потенциальных потребителей продукта с помощью специальных функций сбора статистики по агентам. Создадим функции сбора статистики.

1. Откройте диаграмму класса *Main* и выделите на диаграмме вложенный объект *people*.
2. Перейдите на страницу **Статистика** панели **Свойства** и щелкните мышью по кнопке **Добавить ф-ю сбора статистики**. Откроется секция свойств для задания свойств новой функции сбора статистики по элементам этого реплицированного объекта (*people*).
3. Введите *potentialAadopters* в поле **Имя**. Это будет именем нашей функции, оставьте выбранный по умолчанию **Тип** функции: **Кол-во**.
4. Задайте **Условие**:
`item.statechart.isActive(item.PotentialAdopter)`
 Эта функция будет вести подсчет количества агентов, для которых выполняется заданное условие, т.е. тех агентов, которые находятся в текущий момент времени в состоянии *PotentialAdopter* (являются потенциальными потребителями продукта).
 Здесь *item* - это агент (элемент реплицированного объекта *people*).

Свойства

people - Person

Основные
Параметры
Статистика
Описание

Имя: potentialAdopters

Тип: ☒ Кол-во ☐ Сумма ☐ Среднее ☐ Мин. ☐ Макс.

Выражение:

Условие: item.statechart.isStateActive(item.PotentialAdopter)

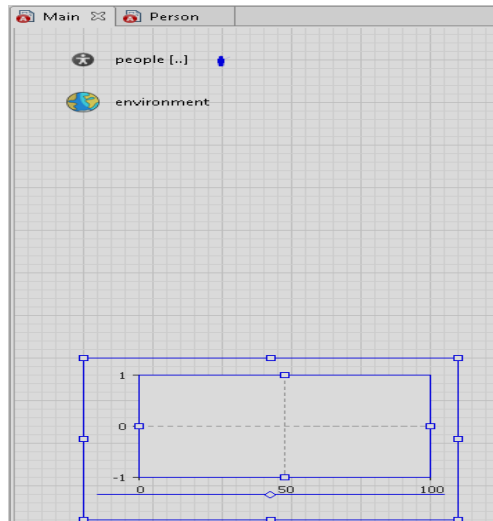
Добавить ф-ю сбора статистики

Аналогично создайте еще одну функцию сбора статистики.

1. Назовите ее *adopters*. Оставьте выбранный по умолчанию **Тип** функции: **Кол-во**.
2. Задайте **Условие**: `item.statechart.isStateActive(item.Adopter)`
Эта функция будет вести подсчет количества агентов, которые находятся в состоянии *Adopter* (то есть, уже приобрели продукт).

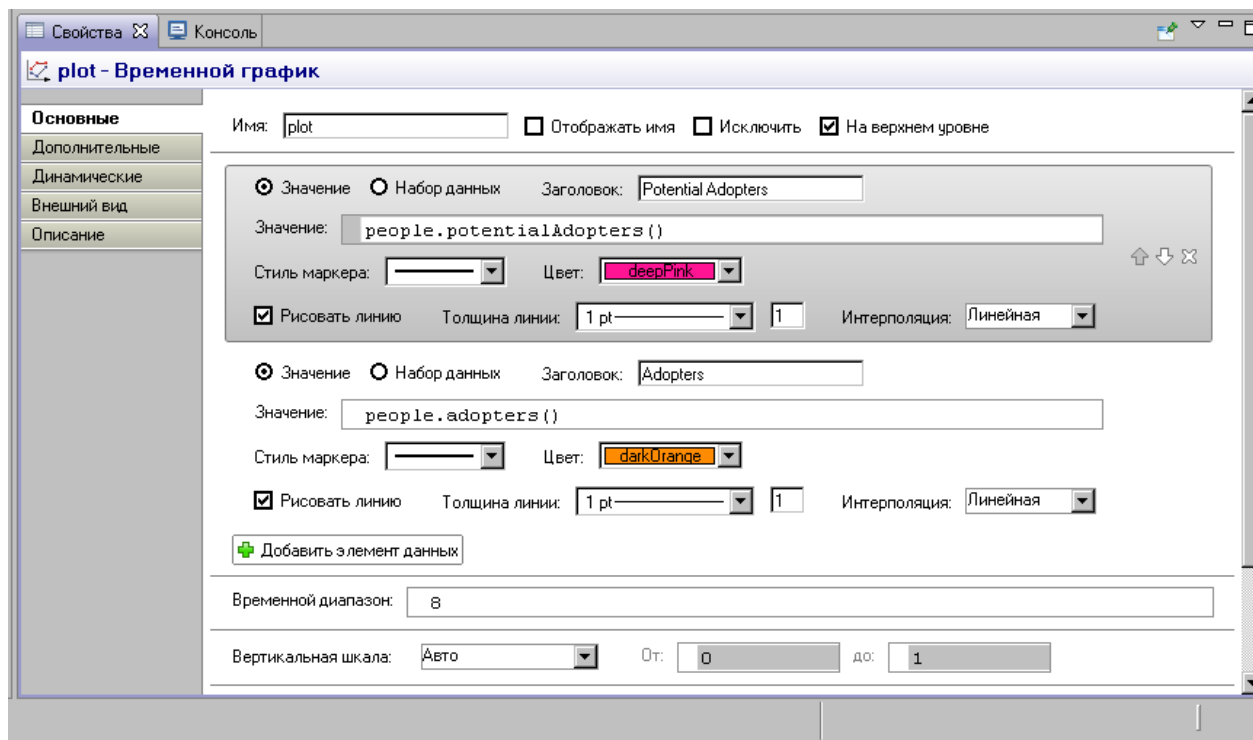
Создадим диаграмму, отображающую динамику изменения числа потребителей и потенциальных потребителей продукта.

1. Откройте диаграмму класса *Main* и перетащите элемент **Временной график** из палитры **Статистика** на диаграмму класса. Измените размер графика и разметьте его так, как показано на приведенном ниже рисунке:



2. Перейдите на страницу **Основные** панели **Свойства** и укажите, что именно Вы хотите отображать на графике - то есть, задайте *элементы данных* этого графика. Для этого щелкните мышью по кнопке **Добавить элемент данных** и в открывшейся секции свойств задайте свойства этого элемента.
3. Введите `people.potentialAdopters()` в поле **Значение**. Здесь мы задаем выражение, результат вычисления которого будет отображаться на нашем графике - в нашем случае мы помещаем здесь вызов ранее созданной нами функции сбора статистики по агентам, возвращающей текущее количество потенциальных потребителей продукта.

4. Введите *Potential adopters* в поле **Заголовок**. Эта строка будет отображаться в легенде диаграммы для данного элемента данных.
5. Аналогично добавьте еще один элемент данных. Пусть он отображает количество потребителей продукта, возвращаемое другой нашей статистической функцией: `people.adopters()`. Задайте *Adopters* в качестве заголовка этого элемента данных и измените свойства внешнего вида, как и в предыдущем случае.
6. Задайте **Временной диапазон**: 8. Тем самым мы задаем диапазон временной оси графика. Поскольку в текущей модели все время моделирования равно восьми единицам модельного времени, то мы можем ограничить и временную ось аналогичным значением.



Теперь наша диаграмма успешно добавлена и сконфигурирована на отображение численностей интересующих нас групп людей.

Запустите модель. С помощью диаграммы Вы можете понаблюдать за динамикой моделируемого процесса. Вы увидите, что под влиянием рекламы каждую единицу времени постоянная доля от общей численности потенциальных потребителей продукта приобретает изучаемый нами продукт.

3. Усложнение модели: учет влияния общения людей и повторных покупок

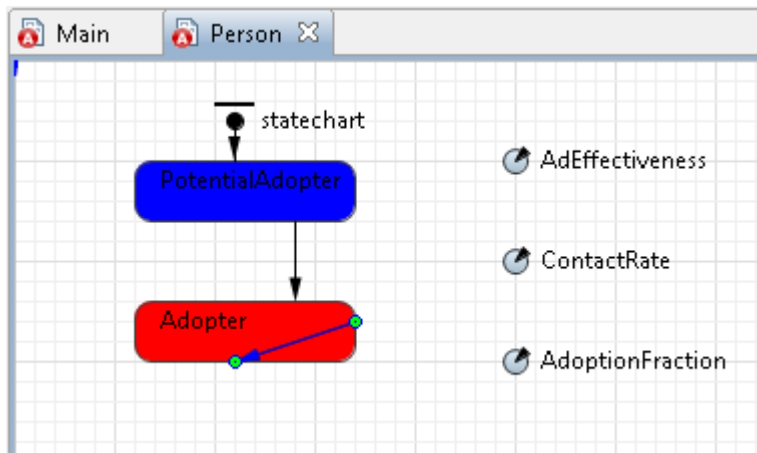
В текущей модели люди приобретают продукт только под влиянием рекламы. На самом деле, рекламный эффект играет значительную роль только в момент выпуска продукта на рынок. В дальнейшем все большую роль будет играть общение людей с теми своими знакомыми, которые этот продукт уже приобрели. В основном люди приобретают новые продукты именно под влиянием убеждения своих знакомых; этот процесс чем-то похож на распространение эпидемии.

Чтобы учесть влияние общения людей, мы должны внести в нашу модель некоторые изменения: задать среднегодовое число встреч и параметр убедительности человека. .

Откройте диаграмму класса *Person*. и создайте новый параметр *ContactRate*. Предположим, что человек в среднем встречается со 100 людьми в год. Выберите **Тип** *int* и введите в поле **Значение по умолчанию** 100.

Добавьте еще один параметр, задающий силу убеждения человека - долю общавшихся с владельцем продукта людей, которая приобретет этот продукт под влиянием общения. Назовите этот параметр *AdoptionFraction*. Задайте **Значение по умолчанию** 0.015.

Измените диаграмму состояний агента: откройте диаграмму класса *Person* и добавьте внутренний переход в состояние *Adopter*.



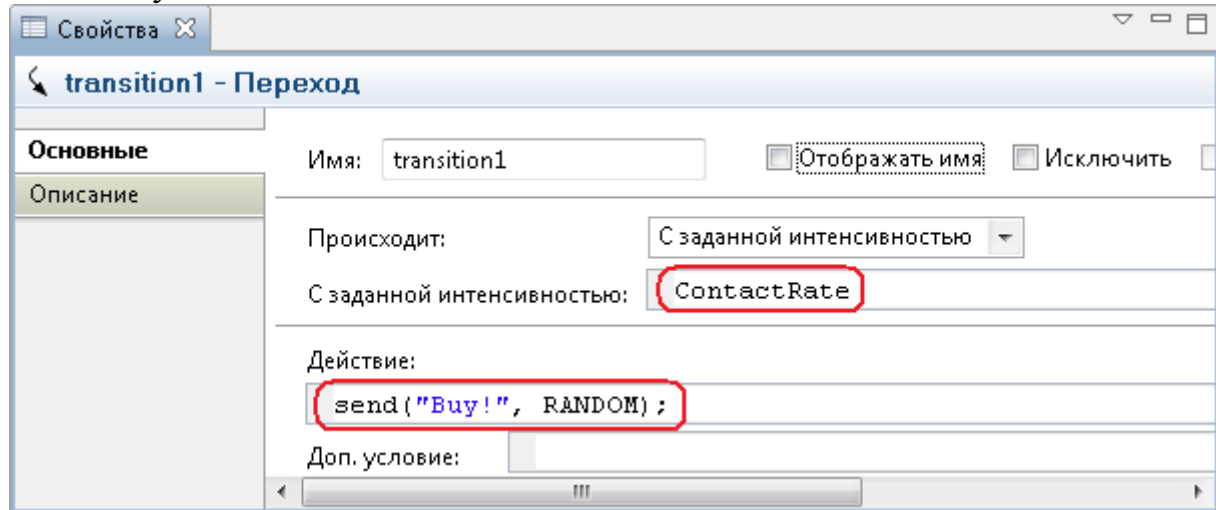
Этот переход будет моделировать общение человека со своим знакомым, в результате тот может быть убежден в покупке нового продукта. Интенсивность срабатывания этого перехода будет зависеть от интенсивности общения этого человека.

Выберите на странице свойств этого перехода из выпадающего списка **Происходит с заданной интенсивностью** и задайте новое значение **Интенсивности** срабатывания этого перехода: *ContactRate*

Задайте **Действие** этого перехода: `send("Buy!", RANDOM);`

Этот переход посылает сообщение случайно выбранному человеку. Позднее мы сделаем так, что вследствие этого будет срабатывать переход диаграммы состояния этого человека, моделирующий покупку им продукта. Метод `send()` отправляет сообщение другому агенту. Первый аргумент задает сообщение, которое будет послано, а второй задает агента, которому это сообщение будет адресовано. В нашем случае мы посылаем сообщение какому-то случайно выбранному агенту, поэтому в качестве значения этого аргумента мы используем специальную

константу RANDOM.

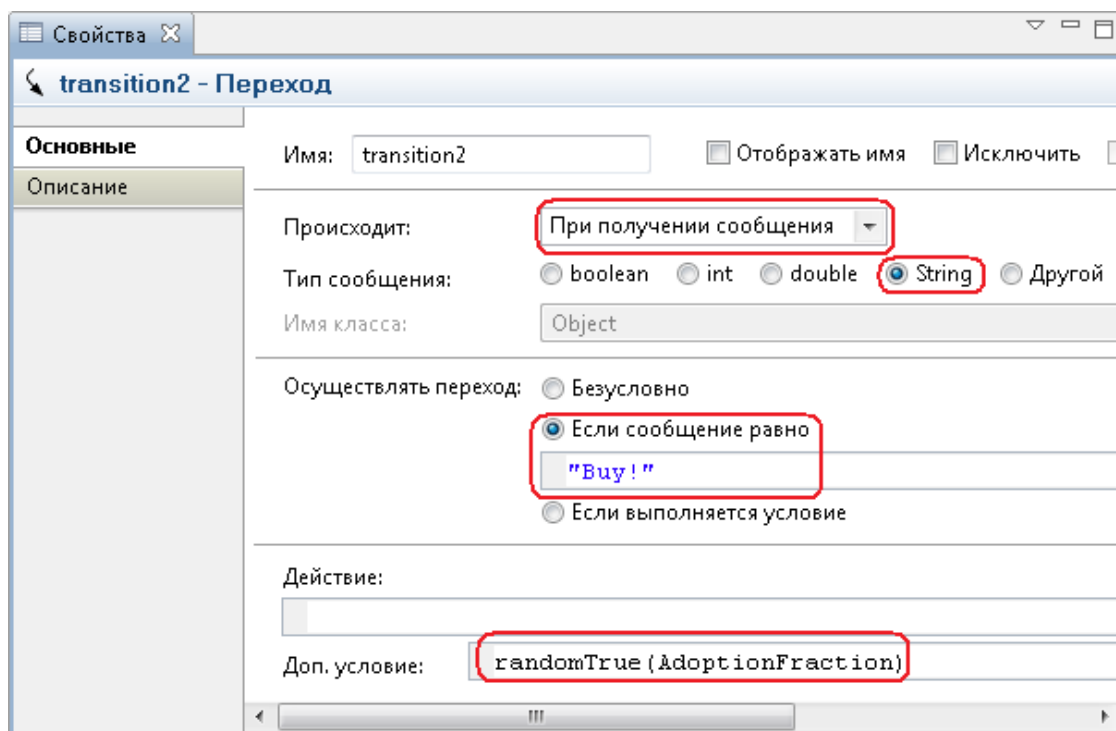


Добавьте еще один переход из состояния *PotentialAdopter* в состояние *Adopter*. Этот переход моделирует процесс приобретения продукта под воздействием общения со знакомым.

Измените свойства этого перехода. Не каждое обсуждение достоинств продукта со своим знакомым приведет к немедленному решению о приобретении этого продукта. Вероятность такого развития событий будет зависеть от того, насколько данный потенциальный потребитель подвержен внушению. В нашей модели данная характеристика задается параметром *AdoptionFraction*. Перейдите на страницу свойств этого перехода и введите `randomTrue(AdoptionFraction)` в поле **Доп. условие**. Это дополнительное условие приведет к тому, что продукт будет приобретаться с вероятностью, задаваемой параметром *AdoptionFraction*.

Этот переход будет срабатывать, когда диаграмма состояний этого агента получит сообщение *"Buy!"* (то есть, "Купи") от другого агента - своего знакомого. Чтобы этот переход срабатывал при получении сообщения, на странице свойств этого перехода выберите из выпадающего списка **Происходит** *При получении сообщения*.

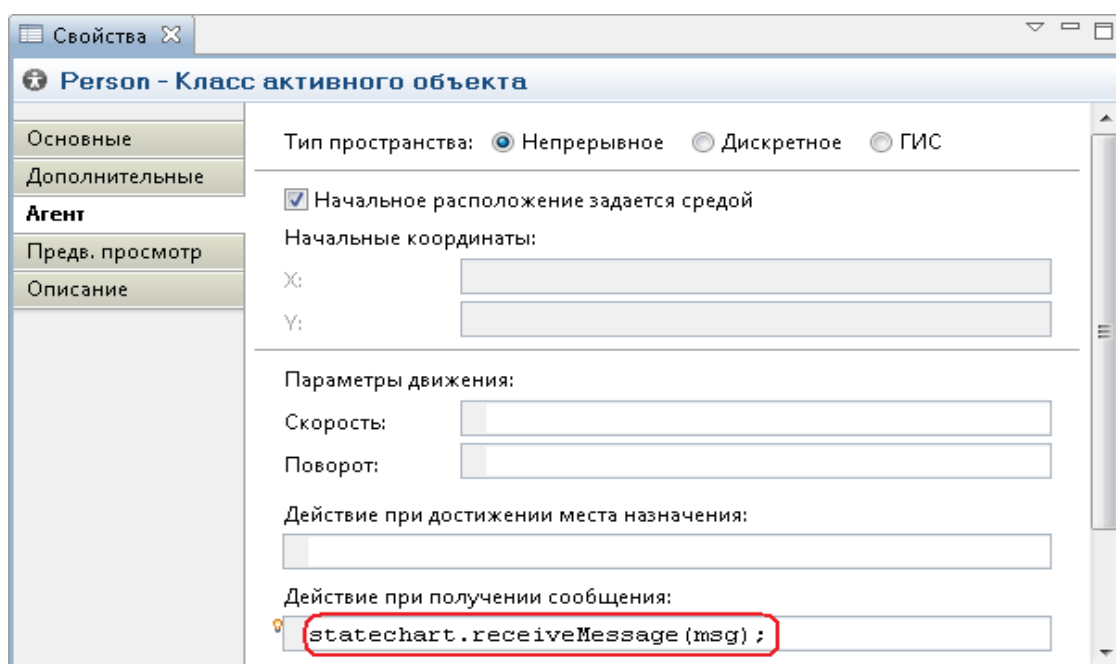
Теперь нам нужно указать, что переход будет срабатывать только при получении сообщения соответствующего содержания. Для этого выберите из группы кнопок **Тип сообщения** опцию **String**, выберите ниже опцию **Если сообщение равно** и введите *"Buy!"* в расположенном ниже поле.



Теперь нам нужно изменить некоторые свойства агента, для того, чтобы получаемые им сообщения от других агентов перенаправлялись в его диаграмму состояний и обрабатывались ею в соответствии с заданной логикой.

Щелкните мышью по классу *Person* в панели **Проекты**, чтобы открыть его свойства в панели **Свойства** и перейдите на страницу свойств **Агент**.

В поле **Действие при получении сообщения** введите `statechart.receiveMessage(msg);`; Теперь когда агент получит сообщение от какого-то другого агента, он будет перенаправлять его в свою диаграмму состояний, где оно будет обрабатываться так, как мы с вами это задали (а именно, вызывать срабатывание перехода, моделирующего приобретение продукта под влиянием личного общения).




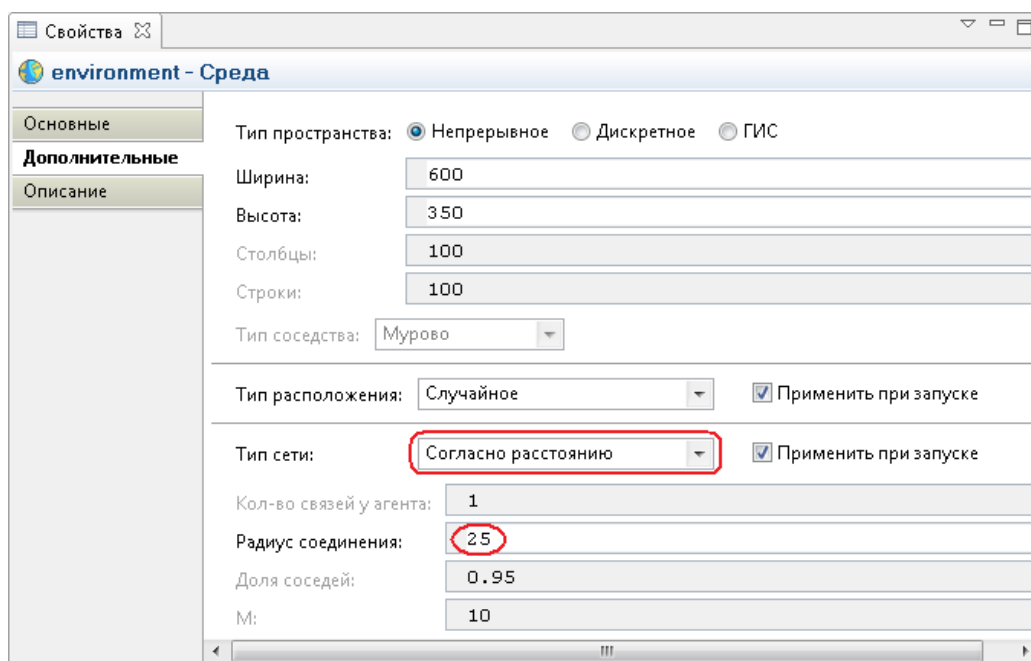
Запустите модель и изучите динамику процесса приобретения продукта. Вы можете увидеть, что из-за учета влияния устного общения этот процесс стал происходить значительно быстрее.

Сейчас люди в нашей модели случайно располагаются в прямоугольном пространстве 650x300 километров (или других условных единиц расстояния). И наша модель допускает общение любого человека с каждым, вне зависимости от того, на каком расстоянии друг от друга они находятся. Обычно же у человека есть определенный круг знакомых, которые живут в непосредственной близости к нему, и именно с ними он и общается. Поэтому мы хотим, чтобы в нашей модели общались только те люди, которые находятся не далее определенного расстояния друг от друга.

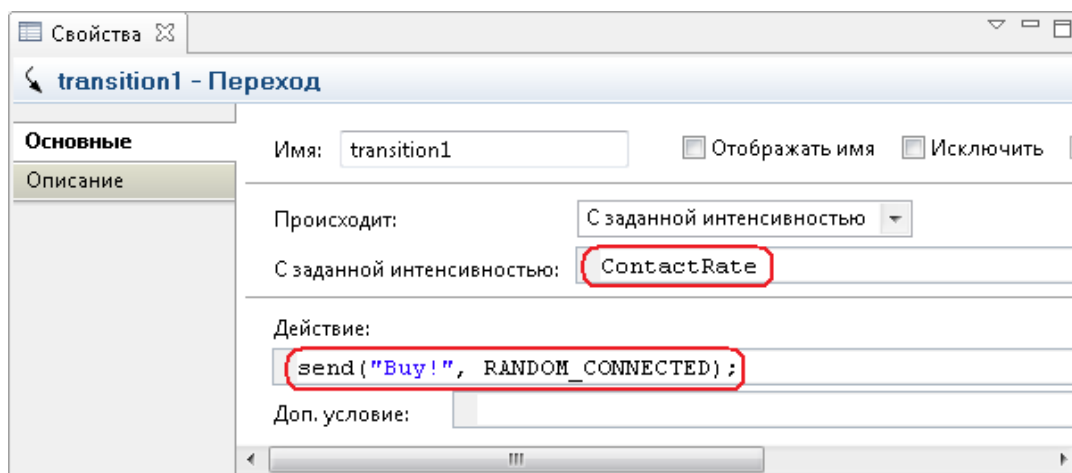
Давайте сделаем нашу модель более реалистичной, допустив возможность общения только тех людей, которые находятся друг от друга на расстоянии, не превышающем 25 километров.

Свойства формирования сетей контактов агентов, как и многие другие свойства агентной модели, задаются в объекте *среда*.

1. Откройте диаграмму класса *Main* и выделите на диаграмме объект *environment* , задающий настройки среды, в которой обитают агенты.
2. Перейдите на страницу свойств **Дополнительные** и измените тип сети контактов. Выберите **Согласно расстоянию** из выпадающего списка **Тип сети** и введите 25 в расположенном ниже поле **Радиус соединения**.



Теперь нам нужно изменить диаграмму состояний агента, чтобы сообщение "Купи продукт!" отсылалось не случайно выбранному агенту, а только тому агенту, который является знакомым данного агента. Для этого откройте диаграмму класса *Person* и измените **Действие** этого перехода на: `send("Buy!", RANDOM_CONNECTED);`

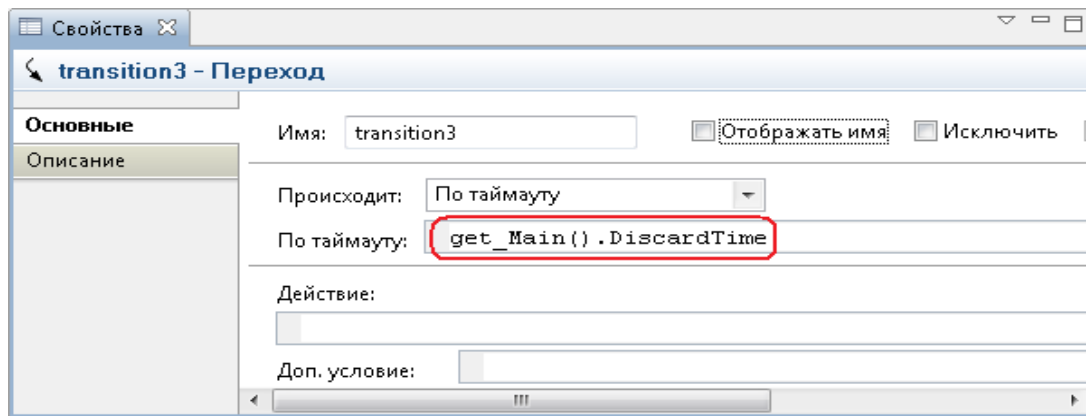


В нашем случае мы посылаем сообщение какому-то случайно выбранному агенту из числа тех, с которым данный агент знаком, поэтому в качестве значения последнего аргумента метода `send` мы теперь используем специальную константу `RANDOM_CONNECTED`. Теперь этот переход посылает сообщение случайно выбранному знакомому этого человека.

Запустите модель и посмотрите, как изменилась динамика приобретения продукта. Можно увидеть, что теперь агенты соединены только с теми, которые находятся от них на расстоянии, не превышающем 25 единиц, а сам процесс распространения продукта происходит медленнее.

Созданная модель не учитывает того, что со временем продукт может быть израсходован или прийти в негодность, что вызовет необходимость его повторного приобретения. Мы промоделируем повторные покупки, полагая, что потребители продукта снова становятся потенциальными потребителями, когда продукт, который они приобрели, становится непригоден. Вначале мы зададим срок службы продукта. Предположим, что средний срок службы нашего продукта - 1 год.

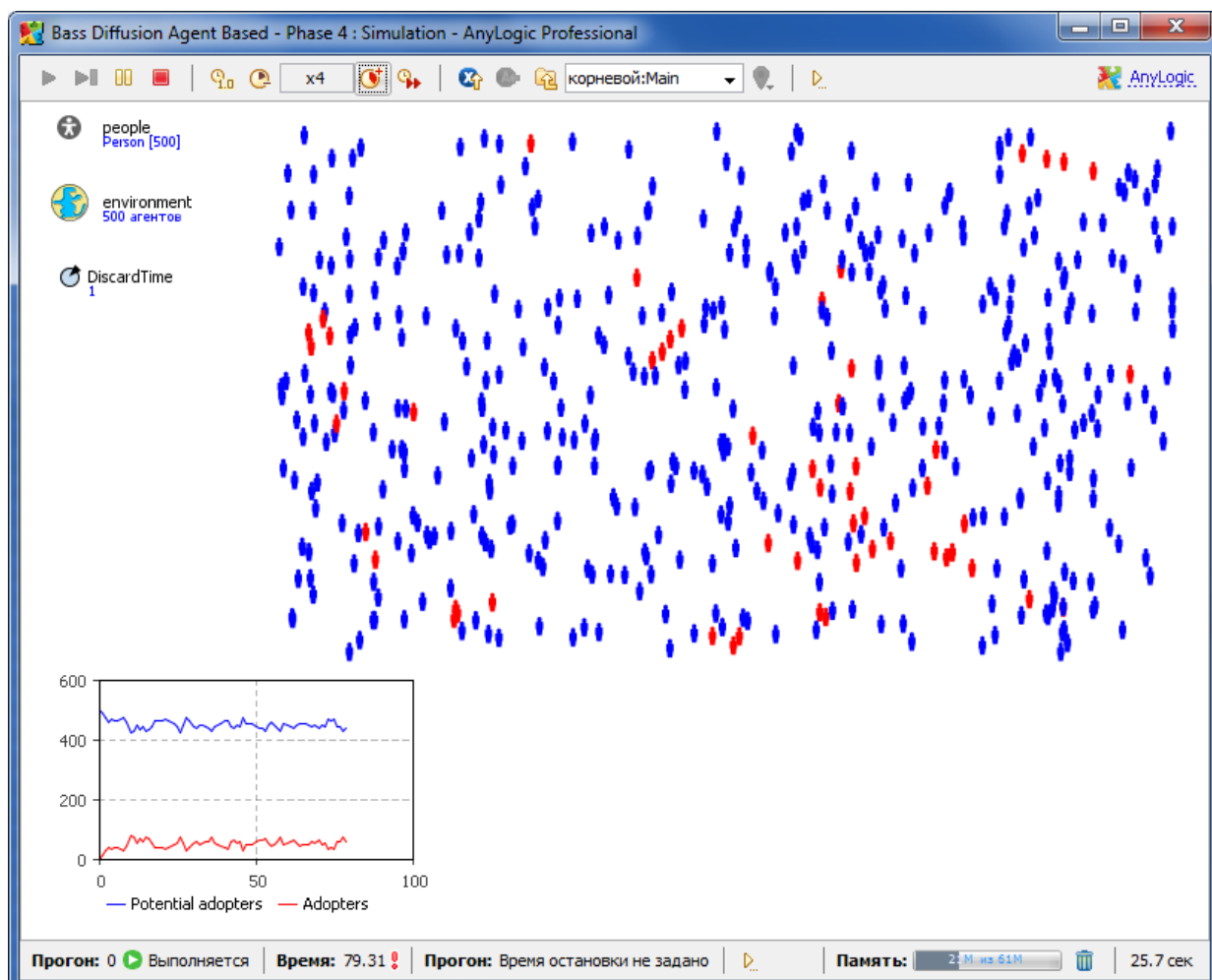
Откройте диаграмму класса *Main*. Создайте параметр *DiscardTime*, со значением по умолчанию 1. Теперь нужно изменить диаграмму состояний таким образом: добавьте переход из состояния *Adopter* в состояние *PotentialAdopter* (*претий по счету*). Свойства перехода: переход будет срабатывать по прошествии срока службы нашего продукта, заданного параметром *DiscardTime*, после того, как управление диаграммы состояний перейдет в состояние *Adopter*. Поэтому оставьте в свойстве **Происходит по** принятое по умолчанию значение *Таймауту* и введите в поле **Таймаут** `get_Main().DiscardTime`. Метод `get_Main()` здесь возвращает экземпляр класса *Main*, в котором мы задали параметр *DiscardTime*.



Теперь мы хотим исследовать процесс приобретения продукта в течение более длительного периода времени. Уберите заданное ранее условие остановки модели по прошествии определенного числа единиц модельного времени, чтобы модель выполнялась бесконечно, пока ее не остановит пользователь.

В панели **Проекты**, выделите эксперимент *Simulation:Main* щелчком мыши. На странице **Модельное время** панели **Свойства**, выберите **Нет** из выпадающего списка **Остановить**.

Запустите модель и с помощью диаграммы проследите динамику изменения числа потребителей продукта. Мы видим, что насыщение рынка в модели с повторными покупками не достигается.



Список литературы

1. Власов, М.П. Моделирование экономических систем и процессов: Учебное пособие / М.П. Власов, П.Д. Шимко. - М.: НИЦ ИНФРА-М, 2013. - 336 с
2. Волгина, О.А. Математическое моделирование экономических процессов и систем: Учебное пособие / О.А. Волгина, Н.Ю. Голодная, Н.Н. Одияко. - М.: КноРус, 2012. - 200 с.
3. Зарубин В.С. Математическое моделирование в технике: Учебник для вузов. – М.: МГТУ им. Баумана, 2010. – 496 с.
4. Емельянов А.А. Имитационное моделирование систем. –М.: из-во МГТУ, 2009
5. Карпов Ю.Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. – СПб: БХВ-Петербург, 2006.
6. Лоу А.М., Кельтон В.Д. Имитационное моделирование. 3-е изд.- СПб: Питер, 2004.
7. Строгалева В.П. Имитационное моделирование. – М.: из-во МГТУ, 2008.
8. Рыжков Ю.И. Имитационное моделирование. Теория и технология. – СПб: КОРОНА принт, 2004
9. Шеннон Р. Имитационное моделирование систем – искусство и наука. –М.: Мир, 1971.
10. Емельянов А.А., Власова Е.А. Дума Р.В. Имитационное моделирование экономических процессов: Учебное пособие. – М.: Финансы и статистика, 2002.
11. Колокольцов, В.Н. Математическое моделирование многоагентных систем конкуренции и кооперации (Теория игр для всех): Учебное пособие / В.Н. Колокольцов, О.А. Малафеев. - СПб.: Лань, 2012. - 624 с.
12. Коросов А.В. Имитационное моделирование в среде MS Excel. Монография: ПитерГУ. Петрозаводск, 2002. -212 с.
13. Томашевский В.Н., Жданова Е.Г. Имитационное моделирование в среде GPSS. – М.: Бестселлер, 2003.

Учебно-методическое издание

**СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В
ЭКОНОМИКЕ**

Методические рекомендации по практикуму

Подписано к печати 01.09.2016.
Формат 60х84 1/16 Бумага офсетная.
Печать ризограф.
Тираж 100 экз.

Международный институт рынка
443030, Самара, ул. Г.С.Аксакова, 21