Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Е. М. РЕМЕЗОВА

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В СРЕДЕ ANYLOGIC

Лабораторный практикум

Владимир 2017

УДК 004.94
ББК 32.973
Р38

Рецензенты:

Кандидат физико-математических наук
доцент кафедры гуманитарных и естественно-научных дисциплин
Владимирского филиала Российского университета кооперации
В. В. Красильщиков

Доктор экономических наук
профессор кафедры управления и информатики в технических
и экономических системах
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В. Г. Чернов

Печатается по решению редакционно-издательского совета ВлГУ

Ремезова, Е. М.

Р38 Имитационное моделирование в среде AnyLogic : лаб. практикум / Е. М. Ремезова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2017. – 87 с.

ISBN 978-5-9984-0806-9

Содержит теоретический материал по основам имитационного моделирования, а также пять лабораторных работ, выполняемых с помощью программного средства моделирования AnyLogic. Достаточно подробно описано выполнение каждой из них, приведены контрольные задания для закрепления практических навыков работы с программным средством AnyLogic.

Предназначен для студентов 4-го курса очной и заочной форм обучения направления 09.03.03 «Прикладная информатика», а также может быть полезен студентам родственных направлений всех форм обучения.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 77. Табл. 1. Библиогр.: 17 назв.

УДК 004.94
ББК 32.973

ISBN 978-5-9984-0806-9

ВВЕДЕНИЕ

Интенсивное развитие компьютерных технологий позволило решать сложные задачи достаточно простыми способами, снижая при этом временные и ресурсные затраты. Один из методов такого упрощения – использование моделирования.

Наиболее широко известным и достаточно удобным способом моделирования сложных систем представляется имитационное компьютерное моделирование объектов и процессов действительности.

Для освоения имитационного компьютерного моделирования необходимы знания в области способов, приемов и технологий данного вида моделирования.

Профессионал, приступая к решению той или иной задачи, должен располагать сведениями в области динамических процессов, подходов и методов решения сложных процессов и систем, таких как аналитические и имитационные, а также знать конкретные информационные системы моделирования и используемые в них языки программирования. Среди множества сред аналитического моделирования основными являются: Maple, MathCAD, MATLAB+Simulink и др.

Освоение моделирования сложных систем может проходить с применением различных сред и методологий разработки аналитических и имитационных моделей сложных систем: MvStudium, MATLAB, Arena, GPSS, Extend, iThink Analyst, Process Model и др. Особое место среди таких систем занимает многоподходная среда моделирования имитационных моделей – AnyLogic. Разные инструментальные средства, представленные в AnyLogic, дают возможность строить модели (динамические, дискретно-событийные, агентные), имитирующие практически любой процесс реального мира, проводить полный анализ модели на ПК без дополнительных вычислений и экспериментов над реальными объектами. Но для возможности оперировать этой программной средой и получать при моделировании верные результаты пользователь AnyLogic должен овладеть технологией работы в среде, понять ее функциональные особенности – для

этого и служит данный лабораторный практикум, касающийся создания компьютерных моделей сложных систем в среде AnyLogic.

В AnyLogic модели представляются визуально-иерархическими на основе простого графического языка моделирования, работающего с объектами и связями между ними. Данная программная среда дает возможность достаточно легко и быстро создавать агентные модели на высоком уровне благодаря языковым конструкциям для задания поведения агентов, их взаимодействия, моделирования среды, а также богатейшим анимационным возможностям. Создавая модели в среде AnyLogic, студенты получают возможность не только «собирать» модели, но и осваивают базовые функции программирования в виде написания некоторого объема программного кода. Использование программы AnyLogic позволяет студенту за ограниченное количество учебных часов научиться строить достаточно сложные и интересные компьютерные модели и в конечном итоге глубже изучить курс «Имитационное и компьютерное моделирование».

В практикуме изложены теоретические основы моделирования, а также приведены пять лабораторных работ, которые содержат задания разной сложности – от самых простых до моделирования дискретных событий и систем массового обслуживания.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Теория основ математического и компьютерного моделирования предполагает содержательное и формальное определение категорий, определений и понятий с целью построения математических моделей сложных систем.

Основными методологическими категориями теоретических основ моделирования являются понятия «объект», «класс», «отношение (связь)», «система», «элемент», «структура».

Определение понятия «объект» имеет различное толкование в зависимости от области рассмотрения. Если рассматривать область имитационного моделирования, то в стратегии объектно ориентированного подхода объект – первое важное понятие. *Объект* – это сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния [1].

Еще одним важным понятием объектно ориентированного подхода считается *класс*. Родственные по определенным характеристикам, поведению объекты объединяются в классы. В зависимости от характеристик одни и те же объекты могут быть в различных классах.

В одном из разделов современной математики – «теории категорий» – термин «объект» используется для обозначения элементов произвольной категории, играющих роль множеств, групп, топологических пространств и т. п. [2]. Здесь также вводится понятие класса объектов и изучаются свойства отношений между математическими объектами, не зависящие от внутренней структуры объектов.

Понятие *«отношение»* определяет взаимное положение объектов, связи между объектами в виде иерархических, ассоциативных, алгоритмических, табличных и других структур.

Понятие *«система»* – основополагающее в теории математического моделирования. Существует несколько десятков различных определений указанного понятия, используемых в зависимости от контекста, области знаний и целей исследования. Изучением систем занимаются такие научные дисциплины, как системология, кибернетика, системный анализ, теория систем, системная динамика и др.

Система – это: 1) целое, созданное из частей и элементов целенаправленной деятельности и обладающее новыми свойствами, от-

сутствующими у элементов и частей, его образующих; 2) объективная часть мироздания, включающая схожие и совместимые элементы, образующие особое целое, которое взаимодействует с внешней средой; 3) объективное единство закономерно связанных друг с другом предметов, явлений, сведений, а также знаний о природе, обществе и т. п. Допустимы и многие другие определения. Объединяет их то, что система есть некоторое правильное сочетание наиболее важных, существенных свойств изучаемого объекта. Каждый объект, для того чтобы его можно было считать системой, должен обладать четырьмя основными свойствами, или признаками: целостностью и делимостью, наличием устойчивых связей, организацией и эмерджентностью [2].

Элемент – это простейшая неделимая часть системы, а ее свойства определяются конкретной задачей. Элемент всегда связан с самой системой. Элемент сложной системы в свою очередь может быть сложной системой в другой задаче [3].

Подсистема – компонент системы – объединение элементов, но по масштабу меньше, чем система в целом [3].

Система может включать большой перечень элементов, поэтому ее целесообразно разделить на ряд подсистем.

Признаки системы: множество составляющих ее элементов, единство главной цели для всех элементов, наличие связей между ними, целостность и единство элементов, наличие структуры и иерархичности, относительная самостоятельность и наличие управления этими элементами. Термин «организация» в одном из своих лексических значений означает также «систему», но не любую систему, а в определенной мере упорядоченную, организованную.

Понятие «подсистема» выработано для анализа сложноорганизованных, саморазвивающихся систем, когда между элементами и системой имеются «промежуточные» комплексы, более сложные, чем элементы, но менее сложные, чем сама система. Они объединяют в себе разные части (элементы) системы, в своей совокупности способные к выполнению единой (частной) программы системы. Будучи элементом системы, подсистема в свою очередь оказывается системой по отношению к элементам, ее составляющим. Аналогично обстоит дело с отношениями между понятиями «система» и «элемент»: они переходят друг в друга. Иначе говоря, система и элемент относительны. С этой точки зрения вся материя представляется как беско-

нечная система систем. «Системами» могут быть системы отношений, детерминаций и т. п.

Наряду с представлением об элементах в представление о любой системе входит и представление о ее структуре.

Структура – это совокупность устойчивых отношений и связей между элементами. Сюда включается общая организация элементов, их пространственное расположение, связи между этапами развития и т. п. [3].

По своей значимости для системы связи элементов (даже устойчивые) неодинаковы: одни малосущественны, другие существенны, закономерны. Структура прежде всего – это закономерные связи элементов. Среди закономерных наиболее значимы интегрирующие связи (или интегрирующие структуры). Они обусловливают интегрированность сторон объекта. В системе производственных отношений, например, имеются связи трех родов: относящиеся к формам собственности, к обмену деятельностью и к распределению. Все они существенны и закономерны. Но интегрирующую роль в этих отношениях играют отношения собственности, или формы собственности. Интегрирующая структура – это ведущая основа системы.

Существует ряд подходов к выделению систем по сложности и масштабу. Например, для систем управления удобно использовать классификацию по числу (количеству) элементов: малые (10 – 103 элемента); сложные (104 – 107 элементов); ультрасложные (108 – 1030 элементов); суперсистемы (1030 – 10200 элементов) [2].

Большая система – это всегда совокупность материальных и энергетических ресурсов, средств получения, передачи и обработки информации, людей, которые принимают решения на разных уровнях иерархии. В настоящее время для понятий «сложная система» и «большая система» используют такие определения:

- *сложная система* – упорядоченное множество структурно взаимосвязанных и функционально взаимодействующих разнотипных систем, которые объединены структурно в целостный объект функционально разнородными взаимосвязями для достижения заданных целей в определенных условиях [4];
- *большая система* – совокупность разнотипных сложных систем.

Тогда определение системы звучит так: «Система – это упорядоченное множество структурно-взаимосвязанных и функционально-

взаимодействующих однотипных элементов любой природы, объединенных в целостный объект, состав и границы которого определяются целями системного исследования» [5].

Среди характерных особенностей больших систем выделяются: значительное количество элементов, взаимосвязь и взаимодействие между элементами, иерархичность структуры управления и участие человека в управлении, необходимость принятия решений в условиях неопределенности.

Описание динамики системы или ее поведения составляет основу любой имитационной модели. В качестве исходных данных для решения этой задачи используются результаты, полученные на этапе разработки концептуальной модели системы, к которым можно отнести определение принадлежности моделируемой системы к одному из известных классов, описание рабочей нагрузки системы и выбор уровня детализации представления системы в модели и ее декомпозиция [2].

Все последующие действия исследователя по созданию модели могут быть отнесены к этапу ее формализации, который в общем случае предполагает выбор метода отображения динамики системы (на основе событий, процессов, или транзактов), формальное (математическое) описание случайных факторов, подлежащих учету в модели, выбор механизма изменения и масштаба модельного времени.

Рассмотрим устоявшиеся понятия в имитационном моделировании: «процесс», «работа», «событие», «транзакт».

Работа (активность) – это единичное действие системы по обработке (преобразованию) входных данных. В зависимости от природы моделируемой системы под входными данными могут пониматься информационные данные или какие-либо материальные ресурсы [6].

Под *процессом* понимают логически связанный набор работ. Некоторые процессы могут рассматриваться как работы в процессе более высокого уровня. Любой процесс характеризуется совокупностью статических и динамических характеристик [2].

К статическим характеристикам процесса относятся длительность, результат, потребляемые ресурсы, условия запуска (активизации), условия остановки (прерывания). Статические характеристики процесса не изменяются в ходе его реализации, однако при необходимости любая из них может быть представлена в модели как случайная величина, распределенная по заданному закону [7].

Динамической характеристикой процесса выступает его состояние (активен или находится в состоянии ожидания) [7].

Моделирование в терминах процессов проводится в тех случаях, когда система оценивается по каким-либо временным показателям либо с точки зрения потребляемых ресурсов. Например, при оценке производительности вычислительной сети обработка заданий может быть представлена в модели как совокупность соответствующих процессов, использующих ресурсы сети (оперативную память, пространство на жестких дисках, процессорное время, принтеры и т. д.).

Если модель строится с целью изучения причинно-следственных связей, присущих системе, то динамику системы целесообразно описывать в терминах событий.

Событие представляет собой мгновенное изменение некоторого элемента системы или состояния системы в целом. Событие характеризуется условиями (или законом) возникновения, типом, который определяет порядок обработки (дисциплину обслуживания) данного события, нулевой длительностью [2].

События подразделяют на две категории [2]:

- *события-следования*, которые управляют инициализацией процессов (или отдельных работ внутри процесса);
- *события-изменения состояний* (элементов системы или системы в целом).

Механизм событий используется в качестве основы построения моделей, предназначенных для исследования причинно-следственных связей в системах при отсутствии временных ограничений. К таким задачам можно отнести, например, некоторые задачи по оценке надежности.

Еще один способ имитационного моделирования систем основан на использовании понятия транзакта, или сущности.

Транзакт, или *сущность,* – это некоторое сообщение (заявка на обслуживание), которое поступает извне на вход системы и подлежит обработке [7].

В некоторых случаях, например при моделировании автоматизированных систем управления, удобно проследить функционирование системы относительно алгоритма обработки транзакта. В рамках одной имитационной модели могут рассматриваться транзакты нескольких типов. Каждый транзакт характеризуется соответствующим

алгоритмом обработки и необходимыми для его реализации ресурсами системы. Прохождение транзакта по системе в некоторых случаях можно рассматривать как последовательную активизацию процессов, реализующих его обработку («обслуживание заявки»).

Для того чтобы построить качественную компьютерную модель сложной системы, необходимо уметь [2]:

- определенным способом представлять в модели динамику (движение) системы. Это может быть описано посредством событий, работ, процессов, транзактов;
- определять способ изменения модельного времени. Здесь выделяют моделирование с постоянным шагом и моделирование по особым состояниям.

В большинстве случаев конечная цель моделирования – оптимизация каких-либо параметров системы, в связи с чем можно выделить следующие виды имитационного моделирования [2]:

- исследование относительного влияния различных факторов на значения выходных характеристик системы;
- нахождение аналитической зависимости между интересующими исследователя выходными характеристиками и факторами;
- отыскание оптимальных значений параметров системы (так называемый «экстремальный эксперимент»);
- сравнение альтернатив для принятия решений;
- оптимизация системы для оценки и выработки оптимальной стратегии;
- анализ ситуаций и обучение в различных отраслях через виртуальные имитационные модели игр;
- визуализация и анимация работы разрабатываемого объекта.

Вид эксперимента влияет не только на выбор схемы его формализации, но также на построение плана эксперимента и выбор метода обработки его результатов.

С точки зрения организации взаимодействия исследователя с моделью (по способу взаимодействия с пользователем), в ходе эксперимента имитационные модели делятся на автоматические и диалоговые. *Автоматические имитационные модели* – взаимодействие пользователя с ними сводится только к вводу исходной информации и управлению началом и окончанием работы моделей. *Диалоговые имитационные модели* позволяют исследователю активно управлять

ходом моделирования: приостанавливать сеанс моделирования, изменять значения параметров модели, корректировать перечень регистрируемых данных и т. д. [8].

Компьютерная модель, или численная модель, – это: 1) компьютерная программа, работающая на отдельном компьютере, суперкомпьютере или множестве взаимодействующих компьютеров (вычислительных узлов), реализующая абстрактную модель некоторой системы; 2) модель, выполненная с помощью компьютерных информационных, схематичных, электронных устройств и технологий и сетей; 3) созданный за счет ресурсов компьютера виртуальный образ, качественно и количественно отражающий внутренние свойства и связи моделируемого объекта, иногда передающий и его внешние характеристики; 4) модель, воспроизводящая моделируемый объект программными средствами на компьютере [2].

Созданию компьютерной модели предшествует разработка мысленных, вербальных, структурных, математических и алгоритмических моделей.

Компьютерные модели подразделяются на аналитические и имитационные. Компьютерные модели различаются по видам применения: обучающие, научно-исследовательские, научно-технические (для исследования процессов и явлений, реальных объектов) и промышленные, встроенные в производственный процесс или адекватно моделирующие производственные процессы на компьютерах. Имитационные модели не только отражают реальность с той или иной степенью точности, но и имитируют ее. Эксперимент с моделью либо многократно повторяется при разных исходных данных, чтобы изучить и оценить последствия каких-либо действий на реальную обстановку, либо проводится одновременно со многими другими похожими объектами, но поставленными в разные условия [2].

Имитационное моделирование при изучении сложных систем является практически основным доступным методом получения информации о поведении системы в условиях неопределенности.

Компьютерные модели сложных систем условно подразделяются на *структурно-функциональные*, которые представляют собой условный образ объекта (технологические диаграммы, сетевые графики, структурные схемы, ГИС, табличный способ, анимационные и мультипликационные), описанный с помощью программных и ком-

пьютерных технологий; *имитационные* – это программа или комплекс программ, позволяющий воспроизводить процессы функционирования объекта в разных условиях; *комбинированные* – позволяют наблюдать и исследовать объект на динамических условных образах модели и имитационных моделях объекта.

Существует множество программных комплексов, которые позволяют проводить построение и исследование моделей (моделирование). Каждое программное средство имеет свой инструментальный набор и дает возможность оперировать различными видами информационных моделей. Поэтому перед исследователем возникает нелегкий вопрос выбора наиболее удобной и эффективной среды для решения поставленной задачи. Следует отметить, что одну и ту же задачу можно решить, используя различные программные среды, от выбора которой зависит алгоритм построения компьютерной модели и форма его представления.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа № 1. ПЕРВАЯ МОДЕЛЬ НА ANYLOGIC: «СОЗДАНИЕ МОДЕЛИ МЯЧИКА»

ЧАСТЬ 1
Цели работы

1. Знакомство с программой AnyLogic на примере модели Balls.
2. Освоение интерфейса программы.
3. Ознакомление с технологией имитационного моделирования.

Ход работы

Интерфейс программы

При первом запуске AnyLogic (с особенностями работы и структуры можно ознакомиться в приложении) открывается начальная страница (рис. 1), которая содержит краткое описание основных возможностей программы, ссылки на примеры моделей, поставляемые вместе с AnyLogic, а также ссылки на веб-сайт компании XJ technologies (разработчика этого программного продукта) и на форму обратной связи с компанией.

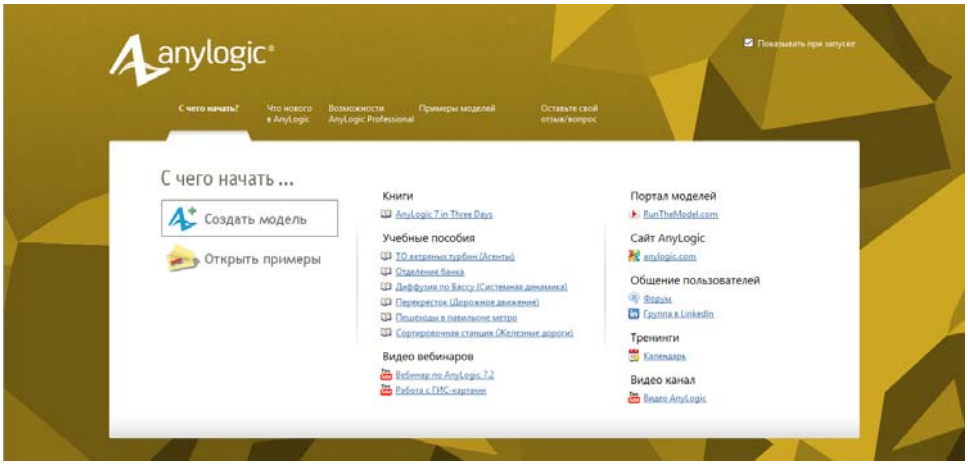


Рис. 1. Стартовое окно AnyLogic

С помощью кнопки **Открыть модель** панели инструментов или команды **Файл/Открыть** в главном меню выберите файл Balls. Это несложная модель прыгающего мяча. На экране возникнет окно, представленное на рис. 2.

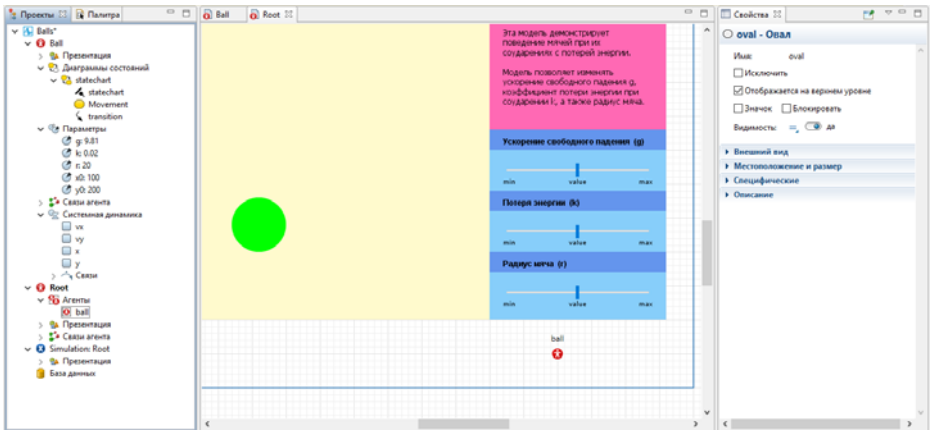


Рис. 2. Окно модели Balls

AnyLogic при открытии проекта открывает несколько панелей: **Проекты, Диаграмма, Палитра, Ошибки** и **Свойства**. Рассмотрим их поочередно.

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

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

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

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

Можно разворачивать и сворачивать ветви дерева элементов модели с помощью кнопок «+» и «-».

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

На диаграммах можно:

- нарисовать презентацию с помощью фигур и элементов управления;
- задать поведение активного объекта с помощью событий и диаграмм действий;
- задать структуру класса, добавив вложенные объекты;
- добавить на презентацию визуализирующие графики, диаграммы.

Панель **Палитра** содержит элементы, которые могут быть добавлены на диаграмму класса активного объекта или эксперимента. По умолчанию она прикреплена к правому краю окна приложения.

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

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

Панель **Свойства** используется для просмотра и изменения свойств выбранного в данный момент элемента модели. Она содержит несколько вкладок. Каждая вкладка содержит элементы управления, такие как поля ввода, флажки, переключатели, кнопки и т. д., с помощью которых можно просматривать и изменять свойства элементов модели. Число вкладок и их внешний вид зависят от типа выбранного элемента.

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

На этапе компиляции модели AnyLogic производит проверку синтаксиса диаграмм, типов и параметров. Все обнаруженные на эта-

пе компиляции и построения модели ошибки отображаются в панели **Ошибки**. Для каждой ошибки показывается ее описание и местоположение – имя элемента модели, при задании которого эта ошибка была допущена.

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

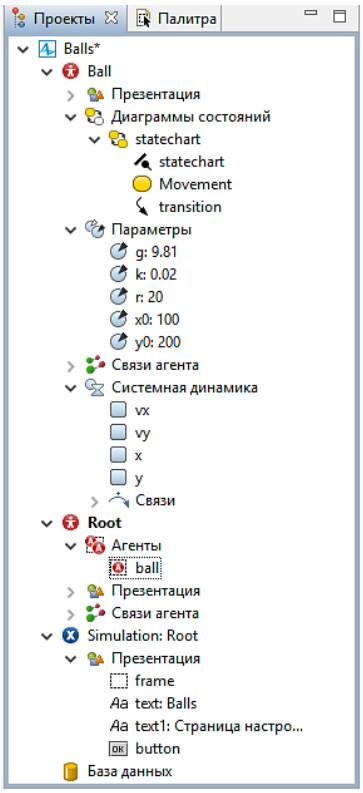


Рис. 3. Дерево проекта

Активный объект строится как класс, который в качестве составных элементов может включать в себя экземпляры других классов активных объектов. Проект Balls включает два класса активных объектов: класс Ball и класс Root. На дереве проекта (рис. 3) как составные элементы класса Ball показаны **Параметры**, **Динамические переменные** и **Диаграммы состояний**, у класса Root составными элементами показаны **Вложенные объекты** и **Презентация**.

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

AnyLogic поддерживает следующие типы экспериментов:

- Простой эксперимент;
- Варьирование параметров;
- Оптимизация;
- Сравнение «прогонов»;
- Монте-Карло;
- Анализ чувствительности;
- Калибровка;
- Нестандартный.

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

Диаграмма класса активного объекта

В рассматриваемой модели диаграмма класса активного объекта – мяча – задается в окне с именем Ball, в котором содержатся его переменные (координаты мяча *x, y* и его скорости *Vx* и *Vy*), параметры (*g, r, k, x*0 и *y*0) и диаграмма состояний с именем statechart (рис. 4).

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

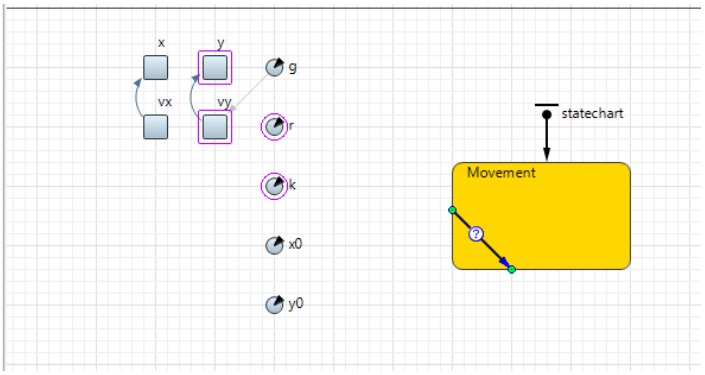


Рис. 4. Диаграмма класса активного объекта

Стрелка, направленная от переменной *Vx* к переменной *x* означает, что переменная *Vx* упоминается в формуле переменной *x*. Стрелки зависимостей рисуются автоматически, всегда синхронизированы с формулами переменных и автоматически появляются или исчезают на диаграмме, как только переменная появится в формуле или будет исключена. Для улучшения наглядности можно редактировать внешний вид стрелок зависимостей – изменять их цвет и радиус закругления.

Структура корневого активного объекта Root задана в окне с именем Root. В модели (см. рис. 2) активный объект Root содержит иконку с именем Ball – один экземпляр активного объекта Ball и анимацию.

Панель свойств объектов модели

Каждый элемент модели обладает теми или иными свойствами или параметрами. При выделении какого-либо элемента в панели **Проект** или **Диаграммы** внизу появляется окно свойств именно этого выделенного элемента. Окно **Свойства** (рис. 5) используется для просмотра и изменения свойств элементов.

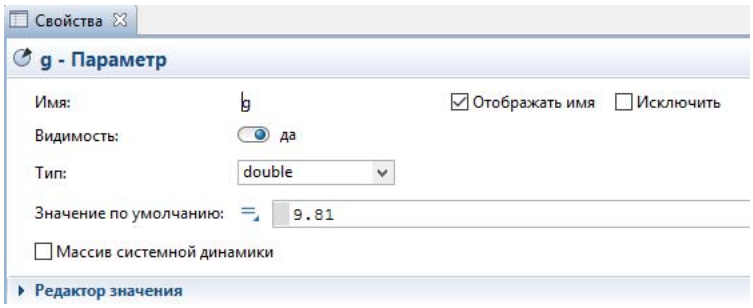


Рис. 5. Окно свойств

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

Поведение активного объекта

Поведение мяча представлено в стейтчарте (см. рис. 3), который состоит из: **Начала диаграммы состояний,** одного состояния с именем Movement и одного перехода. Переход срабатывает при выполнении условия касания поверхности земли при движении мяча вниз, которое можно записать выражением

$$y <= r \;\&\&\; Vy < 0.$$

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

Указанное выражение записано в поле **При выполнении условия** окна свойств перехода (рис. 6). При выполнении данного условия мяч отскакивает, то есть его скорость меняет свой знак на противоположный и уменьшается на долю *k*, моделирующую потерю энергии при отскоке. Это отражено в поле **Действие окна свойств перехода.**

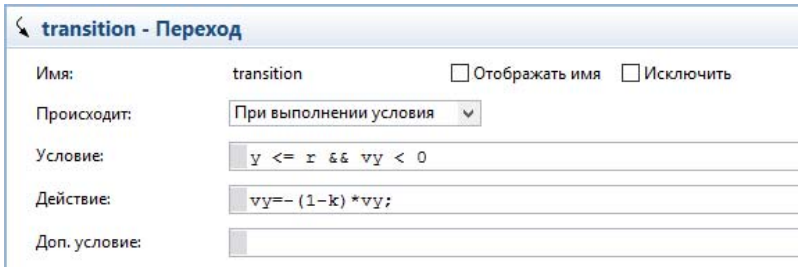


Рис. 6. Окно свойств перехода

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

Презентация

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

Презентация позволяет более наглядно представить динамику моделируемой системы, т. е. координаты, радиус, цвет связываются с переменными и параметрами модели. Изменение переменных модели во времени приводит к изменению во времени внешнего вида геометрических фигур, что позволяет наглядно представить динамику ее поведения. Для презентации используются геометрические фигуры, например окружности, в виде которых представлены параметры моделируемой системы.

На рис. 7 показано, что в окне свойств зеленого круга координаты *X* и *Y* его центра и радиус *r* имеют динамические значения, которые связаны с переменными *x, y* и *r* экземпляра ball класса активного объекта Ball. Таким образом, при работе модели изменение данных переменных будет вызывать перемещение центра и изменение радиуса окружности, моделирующей мяч.

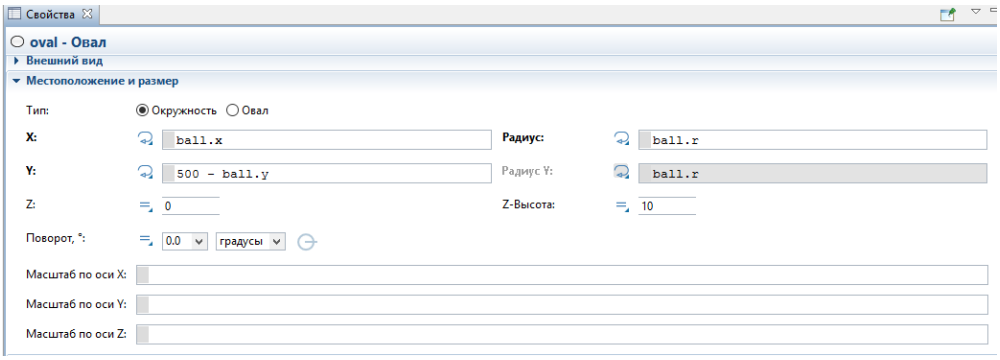


Рис. 7. Свойства объекта ***Мяч***

Щелкнув мышью по различным анимационным объектам в панели диаграммы Root, можно увидеть, что для различных элементов мо-

дели окно свойств содержит различные параметры, характеризующие именно данный элемент. Например, если щелкнуть мышью в презентации на слайдере, то в окне свойств на вкладке **Основные** будет представлена информация о связи слайдера с конкретным параметром модели и геометрическими параметрами самого слайдера.

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

Итак, модель Balls построена и готова к запуску.

Режим выполнения модели

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

При отсутствии ошибок откроется окно презентации эксперимента (рис. 8), которое содержит кнопку **Запустить модель и открыть презентацию класса Main.**

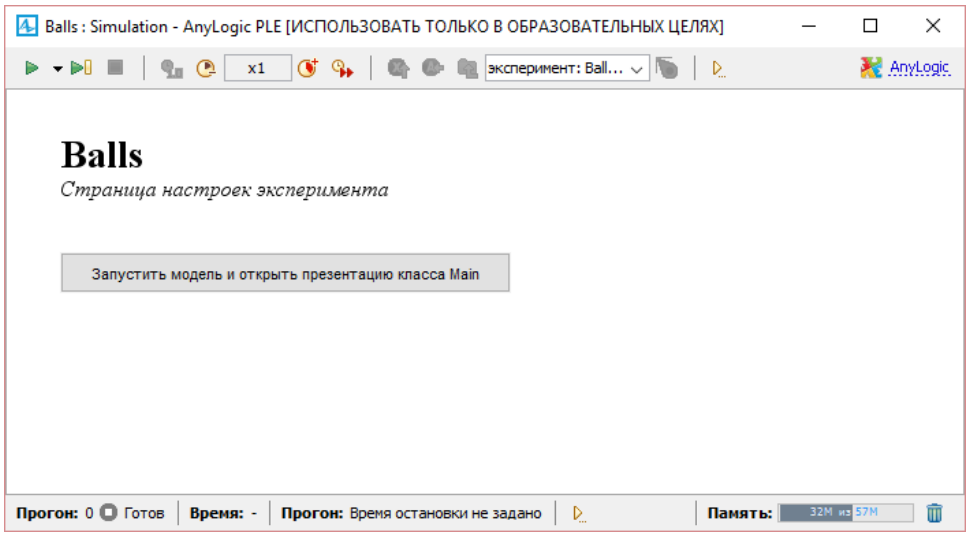


Рис. 8. Окно презентации эксперимента

Когда модель будет запущена, откроется окно презентации либо эксперимента, либо одного из активных объектов запущенной модели

(рис. 9). На презентации будут видны все элементы, в свойствах которых были установлены флажки «На презентации».

При проведении компьютерных экспериментов можно использовать все кнопки, показанные в верхней части окна (рис. 9):

- **Запуск** или **Продолжение моделирования**;
- **Запуск выполнения модели по шагам**;
- **Пауза**;
- **Остановка модели** и **Возврат в окно презентации эксперимента**.

В нижней части окна виден статус модели (**Пауза** или **Выполнение**, **№ прогона** и другая информация).

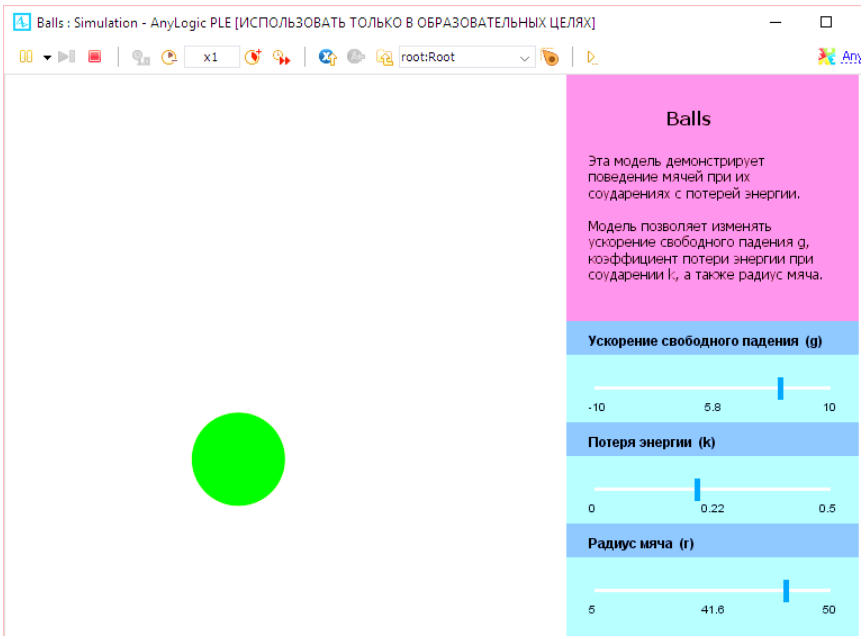


Рис. 9. Запущенная модель

Эксперименты с моделью

На рис. 9 кроме движущегося изображения мяча видны текстовый комментарий и «бегунки», или «слайдеры», – подвижные указатели для изменения параметров модели во время ее выполнения. Перемещая бегунки слайдеров, можно менять три параметра – ускорение свободного падения, долю потери скорости мяча при каждом отскоке и радиус мяча. Изменение параметров позволяет исследовать поведение модели в различных условиях – это и есть компьютерный эксперимент.

Проведите несколько экспериментов с моделью, изменяя ее параметры.

Управление скоростью выполнения модели и изображением

Модель AnyLogic может быть выполнена либо в режиме виртуального, либо в режиме реального времени.

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

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

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

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

Навигация по модели

Расположенный в панели управления окна презентации выпадающий список **Навигация** открывает организованный в виде дерева список объектов модели, обеспечивая простую навигацию по модели и быстрый доступ к любым ее объектам (рис. 10). Корнем дерева объектов является корневой объект запущенного эксперимента. Если структура модели меняется во время ее выполнения, то эти изменения тут же отображаются в дереве объектов модели.

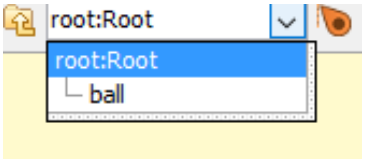


Рис. 10. Выпадающий список

Если выбрать объект Ball, то «всплывает» его структурная диаграмма с динамично изменяющимися значениями переменных *Y* и *Vy*.

AnyLogic поддерживает различные инструменты для сбора, отображения и анализа данных во время выполнения модели. Простейшим способом просмотра текущего значения и истории изменения значений переменной или параметра во время выполнения модели является использование окна **инспекта**. Если щелкнуть мышью по значку переменной в окне презентации, то появится небольшое желтое окно – это и есть окно **инспекта** (рис. 11).

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

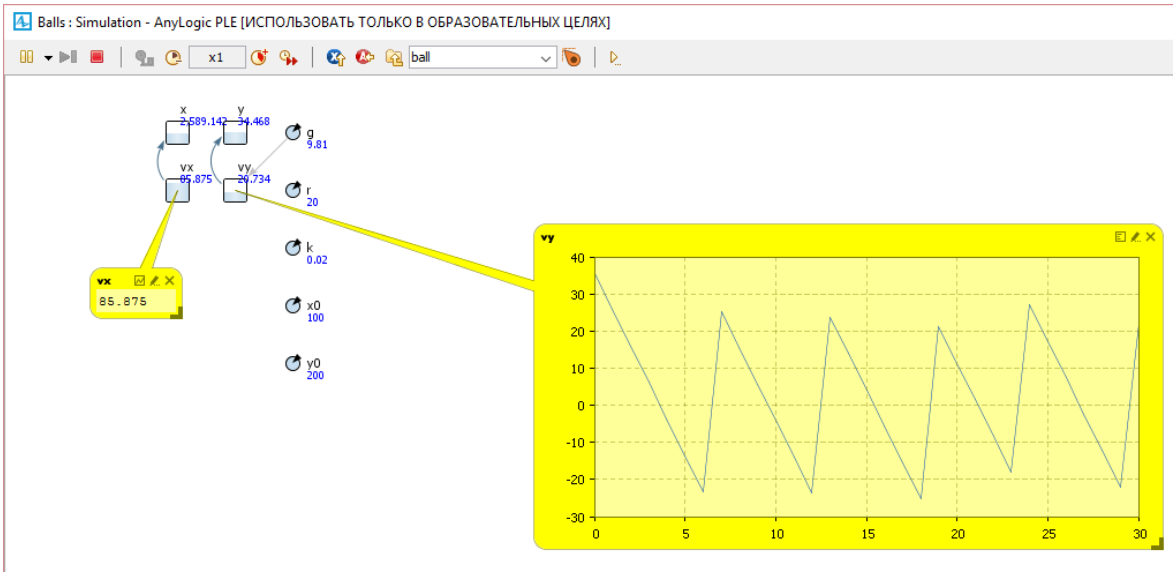


Рис. 11. Окно инспекта

ЧАСТЬ 2

Цели работы

1. Продолжение знакомства с программой AnyLogic на примере модели Balls.
2. Освоение методов редактирования модели.
3. Ознакомление с технологией обработки событий (стейтчартами).

Ход работы

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

Изменение цвета мяча при отскоке

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

Создайте переменную **t0**, которая будет фиксировать момент отскока. Для этого перейдите на диаграмму класса активного объекта Ball, затем в панели **Палитра** откройте вкладку **Системная динамика** и перенесите иконку (**Параметр**) на диаграмму. В поле **Имя** открывшегося окна свойств этого параметра введите **t0**, а в поле **Значение** по умолчанию введите -1 (рис. 12).

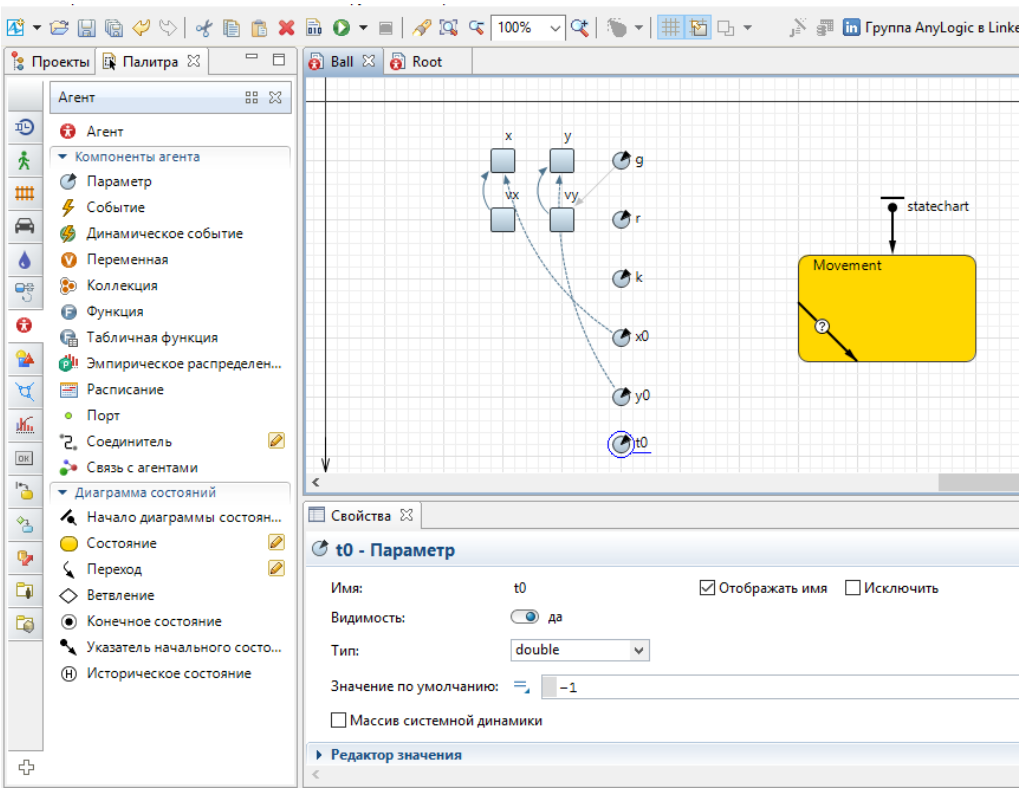


*Рис. 12. Добавление переменной **t0***

Для того чтобы параметр **t0** фиксировал момент отскока, необходимо запомнить значение текущего времени в модели при выполнении условия «отскок» в этом параметре. За наступлением данного условия следит стейтчарт, поэтому выделите мышью переход стейтчарта (рис. 13) и в поле **Действие** добавьте выражение **t0 = time()**.

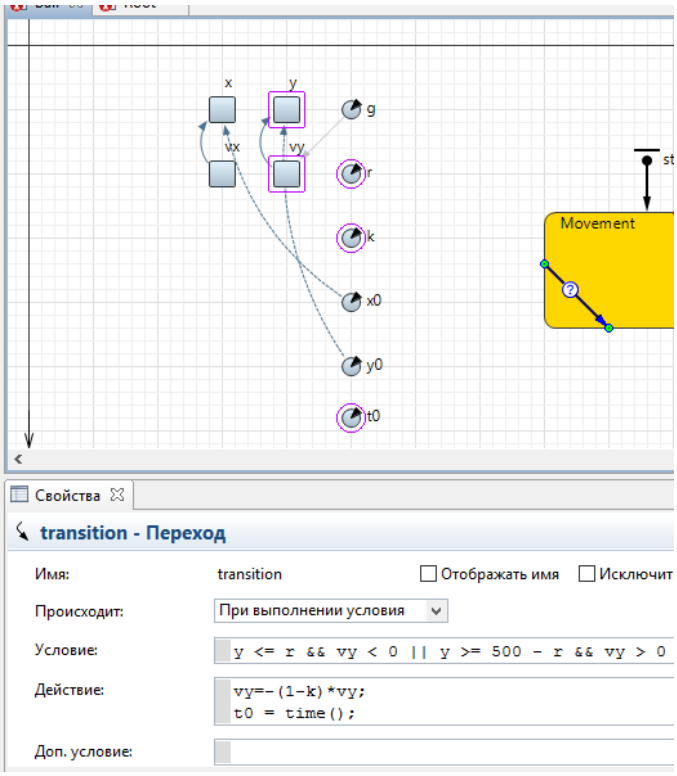


Рис. 13. Свойство стейтчарта

При каждом вызове функция **time()** возвращает текущее значение модельного времени. Параметр **t0** имеет начальное значение -1, а при работе модели хранит значение момента времени последнего отскока. Для того чтобы каждый раз при отскоке мяч изменял цвет на красный (в течение 0,3 с), нужно перейти на диаграмму класса Root, выделить зелёный овал (мяч), в панели свойств этого овала открыть вкладку **Внешний вид** и в поле **Цвет заливки** ввести time()<ball.t0+0.3? red: lime. Это условное выражение устанавливает красный цвет заливки изображения мяча ball на 0,3 с после каждого отскока (рис. 14).

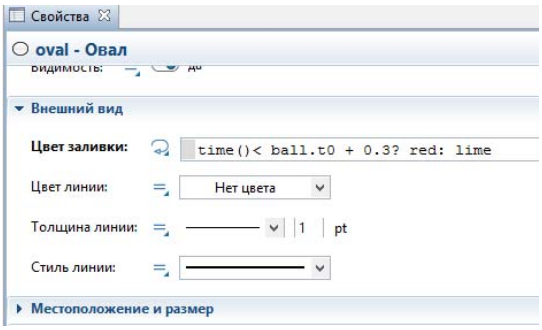


Рис. 14. Свойство мяча

Модель с двумя мячами

Добавим в модель второй мяч. Перейдите на диаграмму класса активного объекта Root и перенесите мышью на него еще один экземпляр мяча. Появившийся объект автоматически получит имя ball1

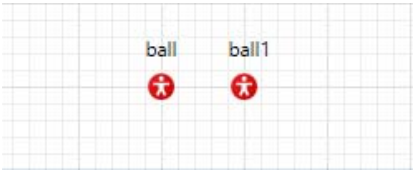


Рис. 15. Добавление второго мяча

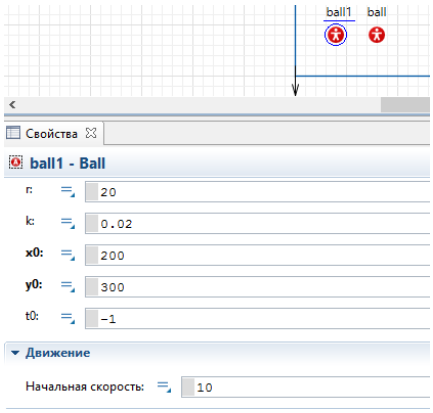


Рис. 16. Установка свойств для второго мяча

(рис. 15). При этом в окне свойств нового экземпляра мяча появятся те же значения параметров мяча, которые были определены для активного объекта Ball.

Установите начальные значения **x0** и **y0** нового мяча равными 200 и 300 соответственно, как представлено на рис. 16.

Для того чтобы на презентации показать движение второго мяча, продублируйте изображение первого мяча с помощью клавиш <Ctrl+C> и <Ctrl+V>. Параметры нового изображения овала (координаты и цвета) связаны с характеристиками объекта ball. Их нужно связать с новым объектом – шаром с именем Ball1, т. е. вместо Ball.x, Ball.y и Ball.t0 в соответствующих полях нужно записать Ball1.x, Ball1.y и Ball1.t0 (рис. 17). А для значения радиусов Радиус X и Радиус Y нужно установить Ball1.r вместо Ball.r.

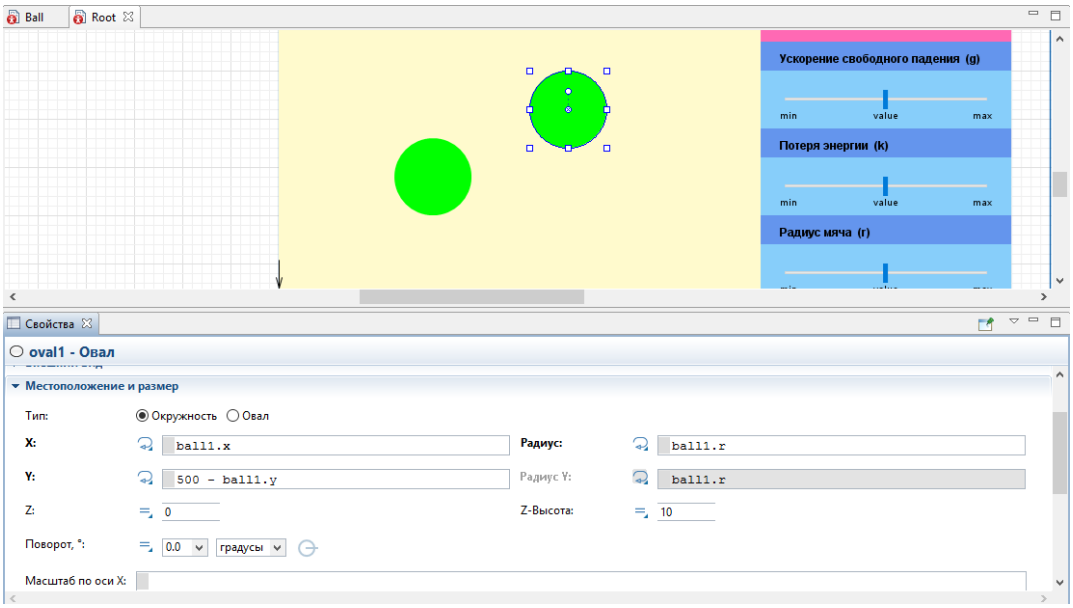


Рис. 17. Настройка презентации второго мяча

Теперь при запуске модели будут имитироваться независимые движения двух шаров.

Произвольные перемещения мяча

В рассматриваемой модели мячи движутся строго вертикально, отскакивая от горизонтальной поверхности. Это происходит потому, что начальная скорость мячей по координате *x* равна 0. Если изменить начальные скорости мячей по *x*, то возникнет необходимость описать поведение мячей при столкновении с вертикальными стенками и потолком.

Зададим случайные начальные значения скоростей *Vx* и *Vy*. Для этого необходимо перейти на диаграмму класса активного объекта Ball, выделить переменную *Vx* и в поле **Начальное значение** этой переменной заменить значение 0 на значение uniform(-100, 100). При этом у различных экземпляров активного объекта Ball начальная скорость по координате *x* будет задана случайно из диапазона (-100, +100) метров в секунду. То же самое сделайте для переменной *Vy*.

Для моделирования отскока мяча от потолка нужно на переходе стейтчарта изменить условие столкновения мяча с поверхностью. Мячи двигаются в пространстве размером 500×500 м. В поле **При выполнении условия** панели свойств перехода стейтчарта активного объекта Ball выражение y <= r && vy < 0 необходимо заменить на выражение y <= r && vy < 0 || y >= 500 - r && vy> 0.

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

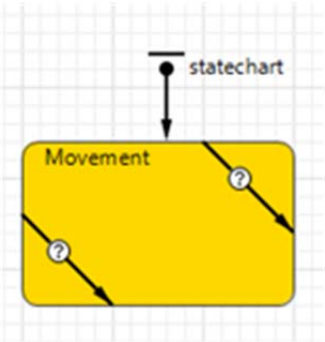


Рис. 18. Добавление нового перехода

Для того чтобы мяч отскакивал от вертикальных стен, нужно записать это условие в стейтчарте, добавив дополнительный переход. Откройте палитру **Диаграмма состояний** и сделайте двойной щелчок мышью по иконке объекта **Переход**, включив тем самым режим рисования. Нарисуйте переход внутри состояния Movement, как показано на рис. 18.

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

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

В поле **Действие** запишите изменение направления составляющей *Vx* скорости мяча и запомните момент времени, когда произошло касание стенки, для последующего изменения цвета мяча (рис. 19):

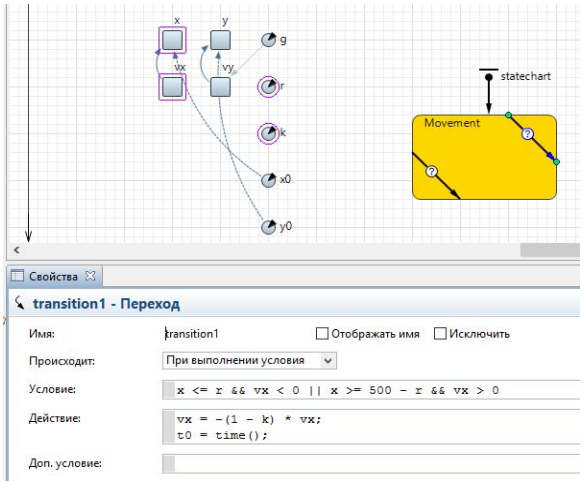


Рис. 19. Задание свойств второго перехода

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

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

Контрольные задания

1. Измените направление движения мяча на горизонтальное.
2. Измените модель таким образом, чтобы при отскоке цвет одного из мячей становился оранжевым, а другого – черным.
3. Измените модель таким образом, чтобы мяч вначале выполнения модели был в левом верхнем углу.
4. Измените модель таким образом, чтобы при отскоке мячи меняли цвет на 2 с.
5. Измените направление движения мяча на вертикальное.
6. Измените модель таким образом, чтобы при каждом отскоке диаметр мяча изменялся на 15 %.
7. Измените модель таким образом, чтобы мяч № 1 менял свой цвет тогда, когда мяч № 2 соударяется со стенками.
8. Измените модель таким образом, чтобы в верхней половине пространства мячи были красного цвета, а в нижней – черного, при отскоке цвет не меняется.
9. Измените модель таким образом, чтобы при движении вправо мяч был голубого цвета, а влево – фиолетового, при отскоке цвет не меняется.
10. Измените модель таким образом, чтобы подсчитывалось количество отскоков мяча.
11. Измените модель таким образом, чтобы при движении вверх мяч был желтого цвета, а вниз – зеленого, при отскоке цвет не меняется.
12. Измените модель таким образом, чтобы моделировалось движение мяча в правой половине пространства.
13. Измените модель таким образом, чтобы моделировалось движение мяча в верхней половине пространства.

Лабораторная работа № 2. СТЕЙТЧАРТЫ: МОДЕЛЬ ПЕШЕХОДНОГО ПЕРЕХОДА

Цели работы

1. Построение стейтчартов.
2. Действия при входе и выходе из состояния, иерархические состояния.
3. Переход по исчерпании таймаута.
4. Переход при получении сообщения.

Ход работы

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

Описание проблемы

Создайте новый проект под названием Svetofor и назовите класс корневого активного объекта Model. Модель будет иметь только один активный объект, представляющий светофор, поэтому корневой объект Model будет единственным активным объектом. На диаграмму класса активного объекта Model поместите **Начало диаграммы состояний** из панели **Диаграмма состояний** и назовите ее «Для_автомобилей», заметьте, что AnyLogic может работать с элементами, набранными кириллицей. Перетащите мышью элемент **Состояние** под стрелочку опции **Начало диаграммы**, как показано на рис. 20.

Рис. 20. Построение диаграммы состояний

Для того чтобы построить стейтчарт, следует использовать элементы из палитры **Диаграмма состояний** (рис. 21).

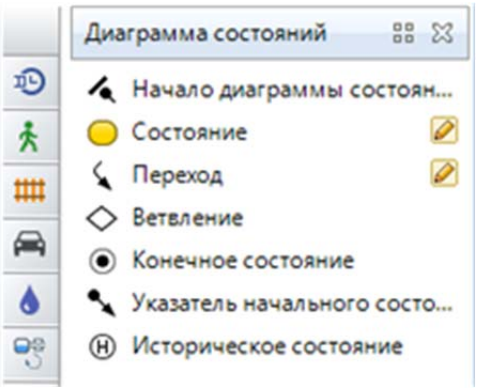


Рис. 21. Элементы диаграммы состояний

Заметьте, что для любого выделенного объекта внизу появляется панель его свойств, в которой можно изменить параметры, в частности имя объекта, если это необходимо. Структурные ошибки при рисовании стейтчарта – повисшие переходы, дублированные указатели начального состояния и т. п. – выделяются в панели **Про-**

екты значком X красного цвета и записью в панели **Ошибки**. Выделенные переходы должны иметь на концах зеленые точки; если точки белые, это значит, что переход не соединен с состоянием – висит.

В соответствии с алгоритмом работы светофора помимо начального состояния в модель нужно ввести дополнительные состояния (рис. 22). Начальное состояние назовите **Движение** – движение автомобилям разрешено (зеленый свет), затем светофор переходит в состояние **Внимание** – внимание (мигающий зеленый), **Медленно** – приготовиться к остановке (желтый свет), остановка транспорта **Stop** – запрет движения (красный свет) и **Приготовиться** – приготовиться к движению (красный и желтый свет горят одновременно).

Состояние **Внимание** представим гиперсостоянием с парой переключающихся элементарных состояний: в одном из них зеленый горит (состояние A), в другом – нет (состояние B). Постройте эти состояния и соедините их переходами, как показано на рис. 22.

Рис. 22. Диаграмма стейтчарта

Зададим условия срабатывания переходов. Переходы в моделируемом автоматическом светофоре выполняются по таймауту, т. е. по истечении времени, которое прошло с момента прихода системы в данное состояние:

- в состоянии **Движение** светофор находится 25 с (t1);
- затем 7 с зеленый сигнал мигает (t4);
- в состоянии **Медленно** 4 с горит желтый (t5);
- в течение 10 с движение запрещено (t6);
- 4 с светофор находится в состоянии **Приготовиться** (t7).

В указанной модели единица модельного времени соответствует 1 с реального времени.

Для задания условий срабатывания переходов выделите переход **t1**, а в поле **Происходит** оставьте без изменения вариант **По таймауту**, в поле **Таймаут** введите 25, единица измерения «Сек» (рис. 23).

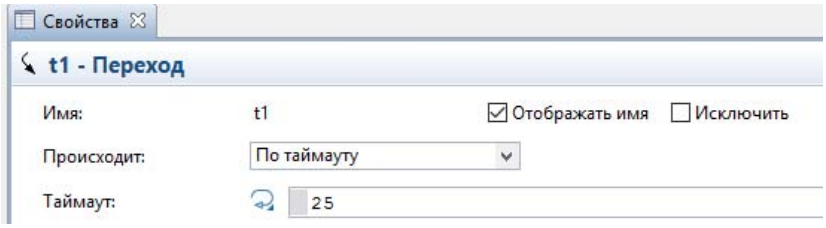


*Рис. 23. Свойства перехода **t1***

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

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

В каждом состоянии светофора должен гореть определенный сигнал: в состоянии **Движение** должен гореть зеленый, в состоянии **Приготовиться** – красный и желтый одновременно и т. п.

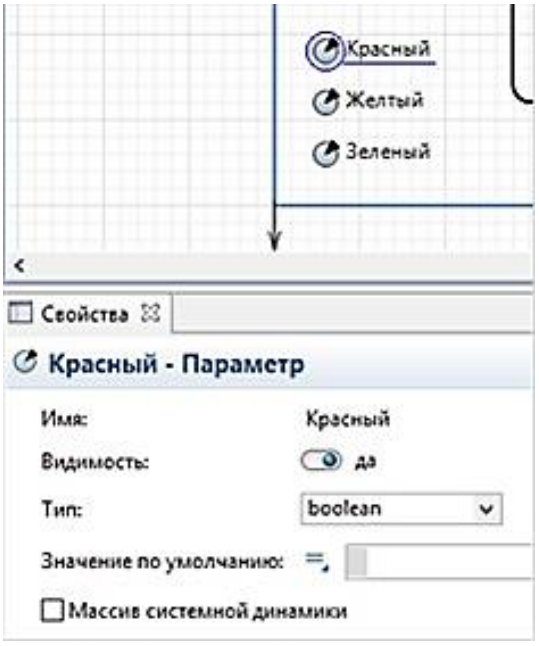


Рис. 24. Добавление трех переменных

Перейдите на диаграмму класса активного объекта Model. Создайте три параметра логического типа: **Красный**, **Желтый** и **Зеленый**, которые будут принимать истинное значение тогда, когда у светофора горит соответствующий сигнал: красный, желтый или зеленый (рис. 24). Начальные значения этих булевых параметров можно не задавать: по умолчанию они будут равны **false**.

В свойствах состояния **Движение** в поле **Действие при входе** запишите **зеленый=true**, а в поле **Действие при выходе** запишите **зеленый = false** (рис. 25).

То же самое запишите для состояния B гиперсостояния **Внимание**, а у состояния A эти поля нужно оставить пустыми – когда светофор находится в этом состоянии, он не горит.

Аналогично в состоянии **Медленно** нужно включить желтый сигнал, т. е. при входе в это состояние установить параметр **желтый** в **true**, а при выходе из этого состояния установить его в **false**.

Для состояния **stop** аналогично опишите состояния параметра **Красный**.

Для состояния **Приготовиться** оба параметра (**Красный** и **Желтый**) нужно установить в **true** при входе и в **false** при выходе.

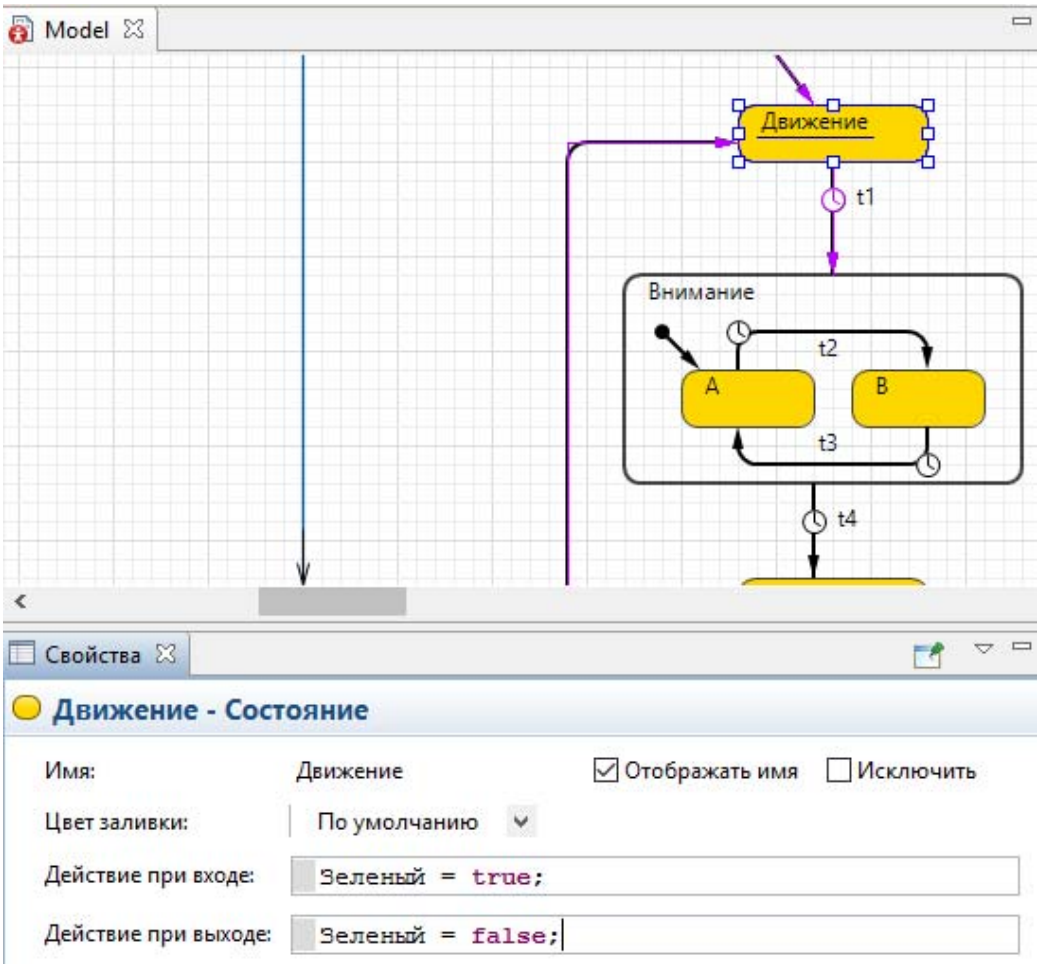


*Рис. 25. Свойство состояния **Движение***

Презентация модели

Презентация модели рисуется в той же диаграмме (в графическом редакторе), в которой задается и диаграмма моделируемого процесса (рис. 26). Графические объекты цвета сигналов светофора в

презентации имеют динамические параметры, все остальные – статические. Светофор строится из трех овалов, повернутых на 45º (поле **Поворот** вкладки **Дополнительные окна свойств овала**).

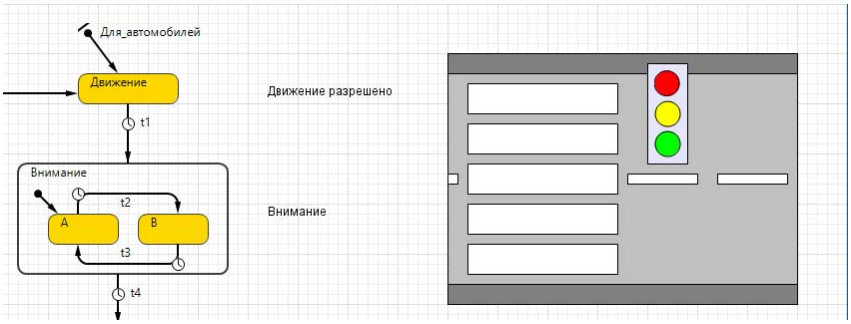


Рис. 26. Модель пешеходного перехода

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

Красный? red: gray

Данное выражение необходимо записать в поле **Цвет заливки**, предварительно сделав его динамическим, нажав на значок =, который преобразуется в «завернутую» стрелочку (рис. 27).

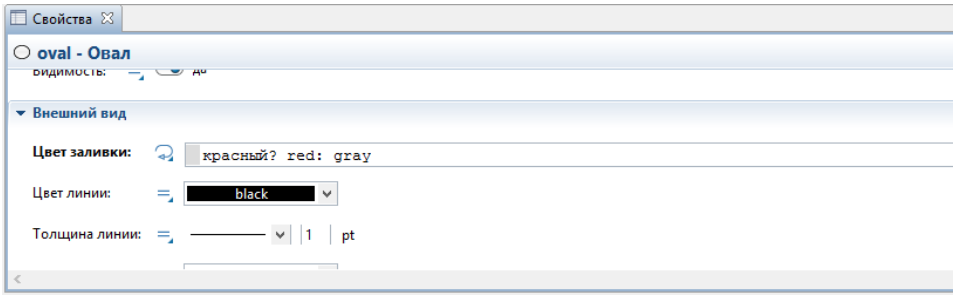


Рис. 27. Свойства переключения светофора

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

желтый? yellow: gray

зеленый? green: gray

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

Запустите модель и проверьте ее работу.

Срабатывание перехода по сигналу

Добавим к модели второй светофор, для пешеходов. Он будет иметь два сигнала, зеленый и красный, и три состояния: **Идите** (зеленый), **Внимание** (мигающий зеленый) и **Стойте** (красный). Добавим в модель два булевых параметра **Стойте** и **Идите**, их значениями будет управлять второй стейтчарт – для пешеходов. Создадим этот стейтчарт на той же диаграмме класса Model, назвав его **Для_пешеходов** (рис. 28).

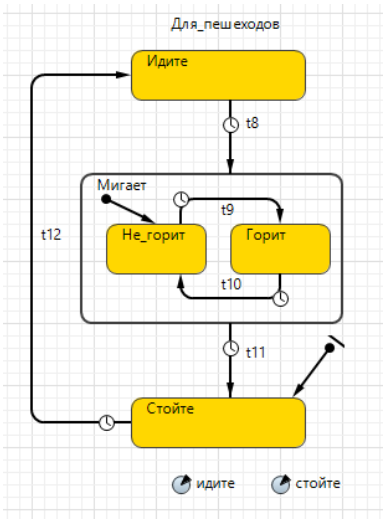


Рис. 28. Стейтчарт светофора для пешеходов

Поскольку управление светофором для пешеходов похоже на управление светофором для автомобилей, новый стейтчарт можно построить, скопировав и изменив уже построенный стейтчарт **Для_автомобилей**. Выделите три состояния стейтчарта **Для_автомобилей** и вставьте их в другое место диаграммы. Эти элементы скопируются, им автоматически будут даны уникальные имена (для того чтобы не было конфликта имен). Переименуйте состояния стейтчарта **Для_пешеходов**, дорисуйте недостающий переход **t12** и перенесите начало диаграммы на состояние **Стойте**, как показано на рис. 28.

Измените параметры в полях **Действие при входе** и **Действие при выходе** в свойствах состояний стейтчарта **Для_пешеходов**. Теперь стейтчарт должен управлять параметрами **Идите** и **Стойте**, которые будут управлять зажиганием света именно пешеходного светофора.

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

В рассматриваемой модели стейтчарты будут обмениваться следующими сигналами: **АВТОМОБИЛИ** и **ПЕШЕХОДЫ**. В стейтчарте **Для_пешеходов** переход **t12** будет срабатывать, когда получен сигнал **ПЕШЕХОДЫ**, который будет генерироваться в стейтчарте **Для_автомобилей** при переходе **t5** в состояние **Stop**. В свою очередь в стейтчарте **Для_автомобилей** переход **t6** будет срабатывать, когда получен сигнал **АВТОМОБИЛИ**, который генерируется в стейтчарте **Для_пешеходов** при переходе **t11** в состояние **Стойте** (рис. 29).

В AnyLogic есть несколько способов передачи сообщения в диаграмму состояний. В указанной модели лучше использовать метод fireEvent( ), который должен вызываться в том стейтчарте, которому предназначено сообщение, т. е. если из некоего объекта необходимо послать сообщение стейтчарту, то в этом объекте нужно написать команду стейтчарт.fireEvent(сообщение). Поэтому в поле **Действие** перехода **t5** стейтчарта **Для_автомобилей** нужно вставить команду

Для_пешеходов.fireEvent("ПЕШЕХОДЫ")

В такое же поле перехода **t11** стейтчарта **Для_пешеходов** вставьте команду

Для_автомобилей.fireEvent("АВТОМОБИЛИ")

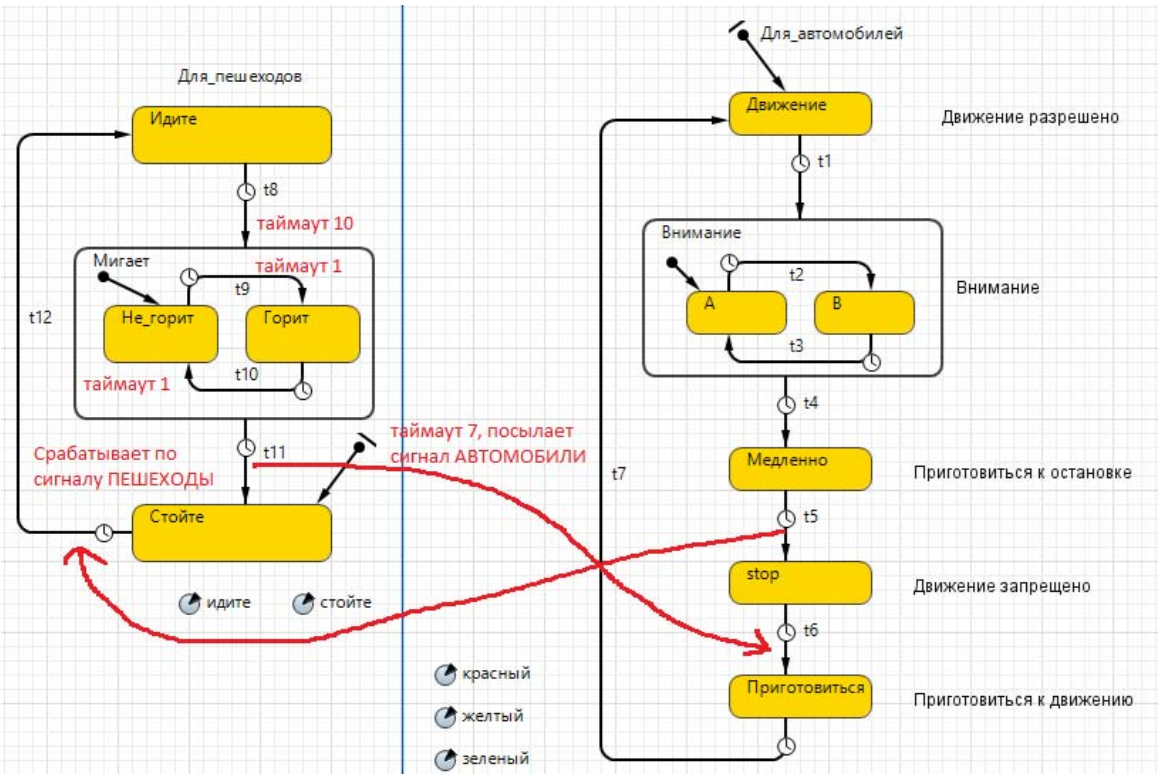


Рис. 29. Связь двух диаграмм

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

Для срабатывания перехода стейтчарта при получении нужного сообщения, в стейтчарте **Для_пешеходов** в поле **Происходит** окна свойств перехода **t12** выберите вариант **При получении сообщения**, укажите тип сообщения **String**, а в поле **Осуществлять переход** выберите **При получении заданного сообщения** и введите **"ПЕШЕХОДЫ"** (рис. 30).

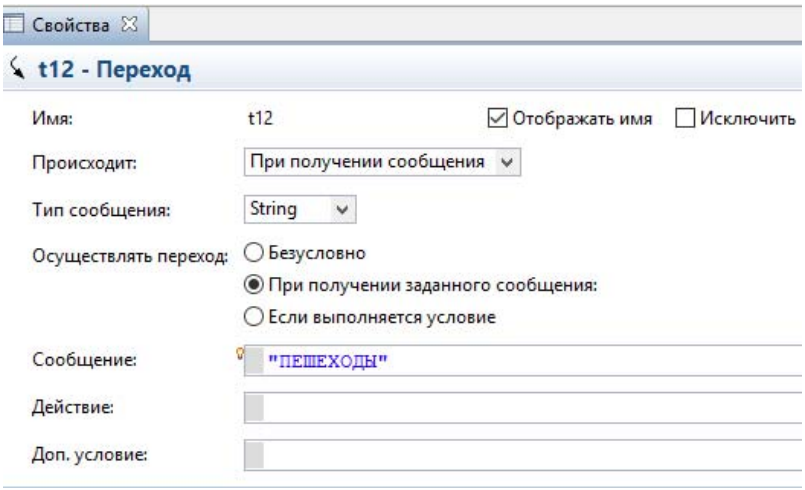


*Рис. 30. Окно свойств перехода **t12***

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

Остальные переходы этих стейтчартов будут срабатывать по таймаутам, как и прежде. Проверьте по рис. 29 установленные параметры переходов стейтчартов. Запустите модель на выполнение.

На презентации модели, в дополнение к светофору для автомобилей, следует нарисовать светофор для пешеходов с двумя сигналами – красной надписью СТОЙТЕ и зеленой ИДИТЕ (как показано на рис. 31). Динамикой цвета этих надписей будут управлять логические параметры **Стойте** и **Идите**, которые нужно создать на диаграмме по аналогии с параметрами **Красный, Желтый** и **Зеленый**.

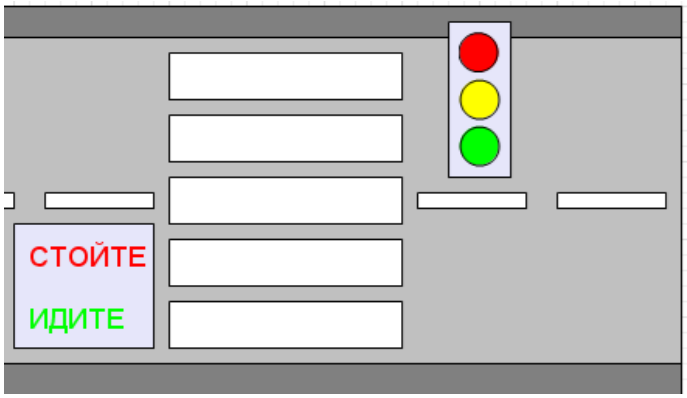


Рис. 31. Презентация с пешеходным светофором

Контрольные задания

1. Измените модель таким образом, чтобы в автоматическом режиме (без кнопки) регулировалось движение двух пересекающихся дорожных потоков, т. е. создайте модель двух трехцветных светофоров со следующей последовательностью сигналов: зеленый – зеленый мигающий – желтый – красный – (красный + желтый) – зеленый.
2. Измените модель светофора с кнопкой таким образом, чтобы пешеход видел, нажата уже кнопка или нет, например, с помощью надписи.
3. Измените модель таким образом, чтобы трехцветный светофор переключался в полуавтоматическом режиме по нажатию кнопок регулировщиком: по кнопке **КРАСНЫЙ** начинает мигать зеленый свет, затем зажигается желтый и через несколько секунд – красный; по кнопке **ЗЕЛЕНЫЙ** зажигаются одновременно красный и желтый, затем горит зеленый.
4. Измените модель таким образом, чтобы частота мигания зеленого света регулировалась с помощью слайдера.
5. Измените модель автоматических светофоров для автомобилей и пешеходов (без кнопки), удалив переходы по сигналу, и подберите подходящие таймауты для согласованной работы светофоров.
6. Измените модель светофора с кнопкой таким образом, чтобы с помощью слайдера можно было регулировать время, через которое повторное нажатие кнопки **ПЕРЕХОД** приведет к переключению светофора для автомобилей в режим ожидания.
7. Как передать сигнал от стейтчарта CT_1 стейтчарту CT_2 ?

Лабораторная работа № 3. ПОСТРОЕНИЕ МОДЕЛИ МАЯТНИКА

Цели работы

1. Освоение методов самостоятельного построения имитационных моделей физических систем, исходя из содержательной, концептуальной и математических постановок задачи.
2. Построение имитационной модели маятника.

Часть 1. ФИЗИЧЕСКИЙ МАЯТНИК

Ход работы

Необходимо построить модель маятника. Колебательные движения механических систем широко распространены в технике: качания маятников, движения поршней двигателей внутреннего сгорания, колебания струн, стержней и пластин, вибрации двигателей, фундаментов и др.

Содержательная постановка задачи

Тело единичной массы прикреплено к неподвижному кронштейну с помощью нерастяжимой и несжимаемой нити длиной l. Необходимо исследовать колебательные движения тела (рис. 32).

Концептуальная постановка задачи

Примем следующие предположения:

1. Объектом исследования является тело массой 1, принимаемое за материальную точку.
2. Движение тела подчиняется второму закону Ньютона.
3. Тело находится под действием трех сил: силы тяжести mg, реакции натяжения нити и силы сопротивления воздуха, пропорциональной квадрату скорости движения.
4. Тело совершает колебательные движения, так как сила тяжести mg уравновешивается вертикальной составляющей реакции натяжения нити.

Математическая постановка задачи

С математической точки зрения имеем задачу Коши:

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

$x = l\sin(\alpha)\,; y = l\cos(\alpha),$

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

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

Описание модели

Необходимо построить модель по вышеприведенным задачам. Модель должна содержать две переменных состояния *alpha* и *omega* и три параметра *l, mu* и *g*, а также начальное значение переменной *alpha*, которая задается параметром с именем *alpha*0. Переменные *x* и *y* задают координаты центра масс маятника, значения которых можно вычислить по формулам, в соответствии со схемой рис. 32.

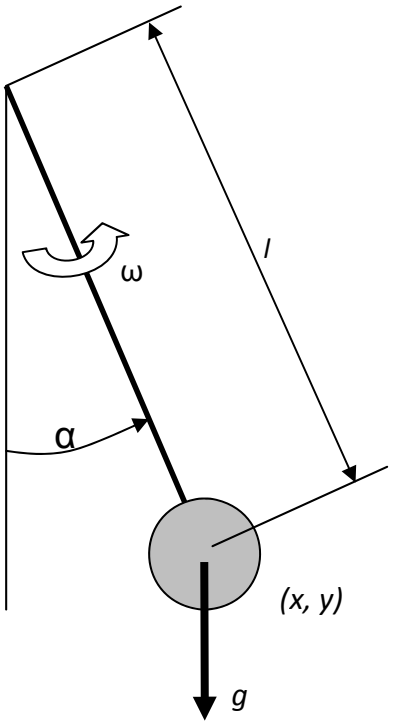


Рис. 32. Схема маятника

В корневом объекте модели создайте четыре переменных: *x*, *y*, *alpha* и *omega*. Переменные *x* и *y* следует задать формулами, а *alpha* и *omega* – интегралами по задаче Коши. Начальное значение угловой скорости *omega* равно 0. Четыре параметра *l*, *mu*, *g* и *alpha0* также необходимо создать на диаграмме класса активного объекта.

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

Презентация

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

Соответственно, на закладке **Динамические панели** свойств линии, отображающей нить, в поле значений координат начала нити *X* и *Y* введите 0, в поле значений координат конца нити *dX* и *dY* введите величины *x* и *y* (рис. 33). Это приведет к тому, что при работе модели концы отрезка всегда будут находиться в точках с координатами (0, 0) и (*x, y*). У **овала** во вкладке с динамическими значениями координат установите *x* и *y*.

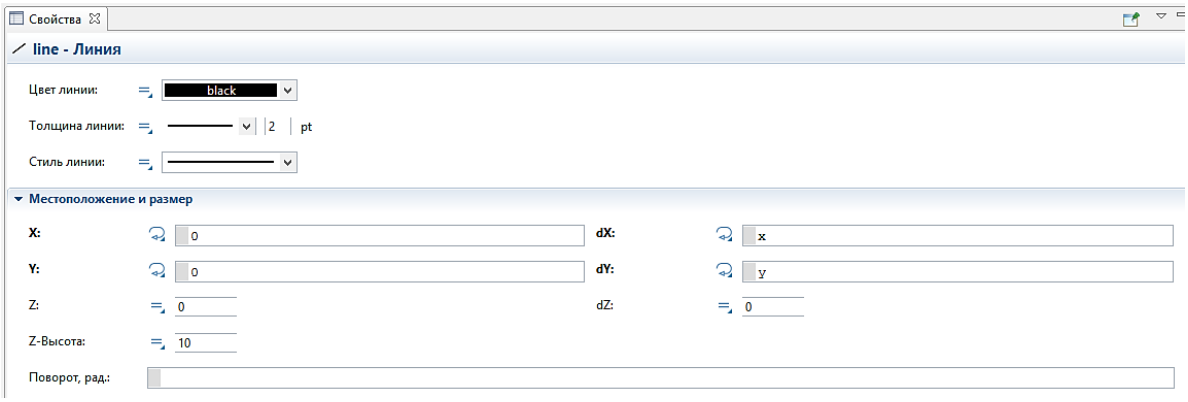


Рис. 33. Свойство линии

Третья область должна содержать название модели и небольшой поясняющий текст.

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

В нижней части диаграммы класса активного объекта разместите графики зависимости *alpha* (ось *Y*) и *omega* (ось *Y*) от времени (ось *X*, прописывается как time()) и фазовую диаграмму зависимости *alpha* (ось *Y*) от omega (ось *X*).

Постройте модель, как показано на рис. 34, и продемонстрируйте ее преподавателю.

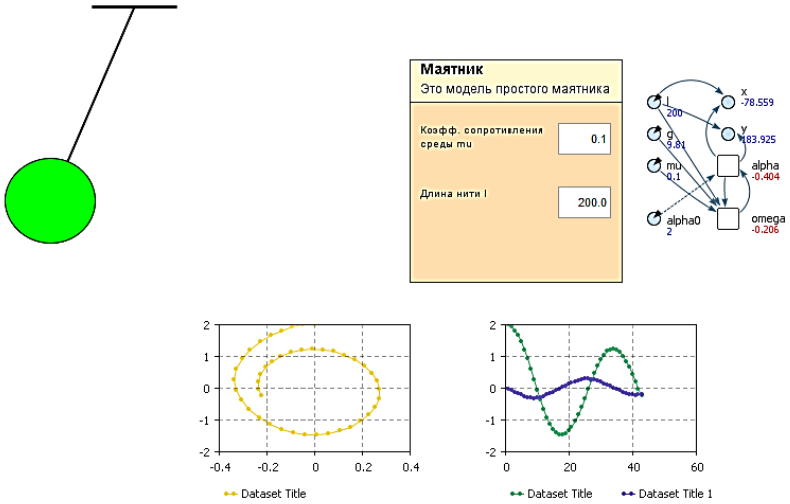


Рис. 34. Модель маятника

Часть 2. МАЯТНИК С ОГРАНИЧИТЕЛЕМ

Ход работы

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

Описание проблемы

Модель представляет собой детализированный вариант предыдущей модели маятника. Маятник при соприкосновении нити с ограничителем начинает закручиваться вокруг ограничителя, т. е. в модели появляется второй центр вращения. Уравнения, описывающие движение маятника без ограничителя (на нити длиной $L + l$) и с ограничителем (на нити длиной l), представлены ниже:

1) движение без ограничения:

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

$$x = (L+l)\sin(\alpha),$$

$$y = (L+l)\cos(\alpha);$$

2) движение при ограничении:

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

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

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

где α – текущий угол отклонения маятника от вертикали; ω – его угловая скорость; μ – коэффициент сопротивления среды, пропорциональный квадрату угловой скорости.

Начальные условия: $\alpha(0) = \alpha 0, \omega(0) = 0$.

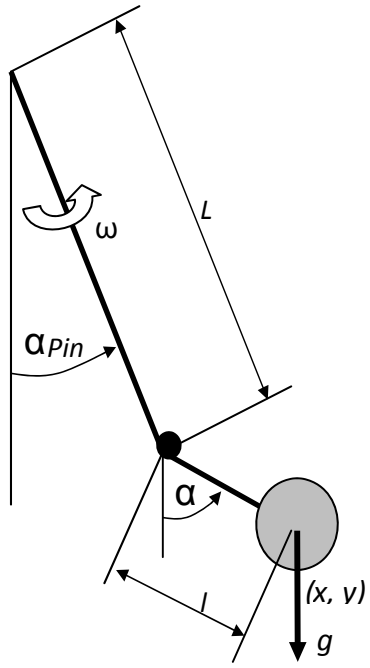


Рис. 35. Модель маятника

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

Модель должна содержать четыре переменные: *x*, *y*, *alpha* и *omega*, и шесть параметров: *L*, *l*, *mu*, *alphaPin*, *g* и *alpha0*. Их физический смысл ясен из рис. 35. Как и в предыдущей модели, *alpha0* – начальное значение угла отклонения маятника.

Поведение маятника описывается стейтчартом с именем **Колебания** (рис. 36), с двумя состояниями. В одном состоянии описывается непрерывное движение маятника без ограничения (состояние **Без_ограничения**), в другом (состояние **С_ограничением**) – описывается движение вокруг ограничителя. В зависимости от состояния, в котором находится система, переменные *x*, *y* и *omega* будут описываться различными уравнениями.

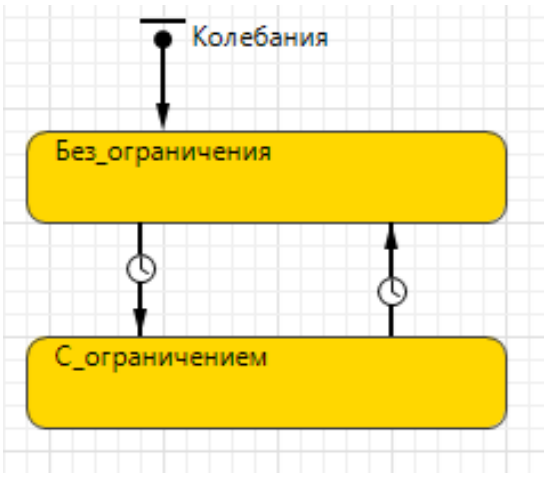


Рис. 36. Стейтчарт перехода маятника

Условия перехода из одного состояния в другое, а следовательно, изменение поведения системы определяются следующим образом. Начальное положение маятника задается его углом α0. Маятник соприкасается с ограничителем, если он движется против часовой стрелки (ω > 0) и его угол

равен α*Pin* или если он движется по часовой стрелке и его угол меньше, чем (2π – α*Pin*). Запишем это на языке Ява в условии перехода из состояния **Без_ограничения** в состояние **С_ограничением**:

alpha ≥ alphaPin && $omega > 0$ || $alpha$

$\leq -(2 * \mathrm{PI} - \mathrm{alphaPin})$ && $omega < 0$

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

$alpha > 0$ && $alpha \leq alphaPin$ && $omega < 0$ ||$alpha$

< 0 && $alpha \geq -(2 * PI - alphaPin)$ && $omega > 0$

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

$$omega_{\text{огр}} = omega_{\text{не\_огр}}(L + 1)/1,$$

$$omega_{\text{не\_огр}} = omega_{\text{огр}}1/(L + 1).$$

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

Эти соотношения следует записать в соответствующие поля **Действие** панели свойств переходов стейтчарта (рис. 37).

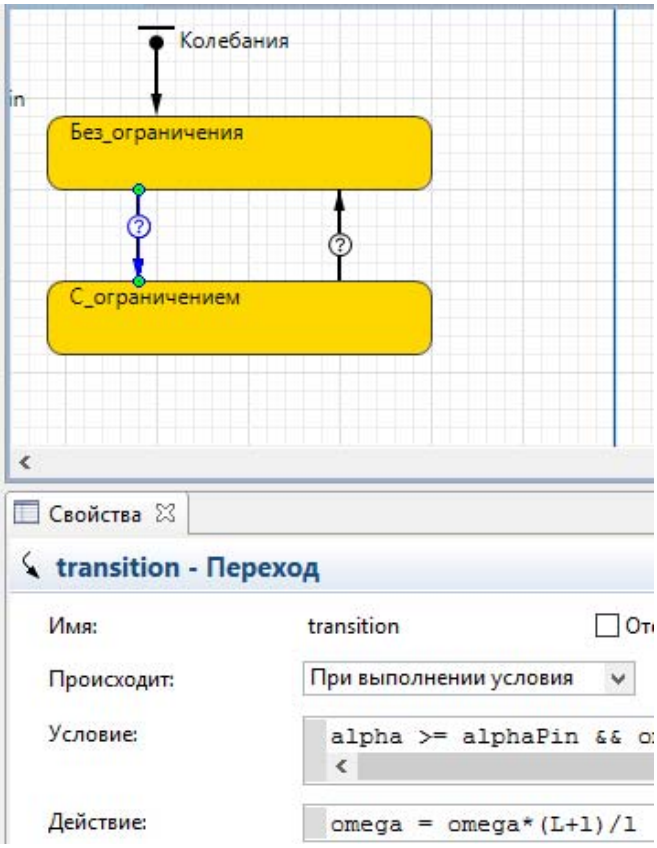


Рис. 37. Действие при переходе

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

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

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

Презентация

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

X: L*sin(alphaPin)

Y: L*cos(alphaPin)

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

Координаты начала нити *L* строятся точно так же, как и в ч.1 лабораторной работы. А вот координаты конца нити *L* (или начала *l*) строятся иначе; заметьте, что в этих точках нити соединяются и их координаты будут одинаковы – это точка излома.

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

X: L * sin(alpha), Y: L*cos(alpha)

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

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

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

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

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

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

Аналогично заполняется поле **dy**.

Достройте презентацию модели, как показано на рис. 38, и продемонстрируйте построенную модель преподавателю.

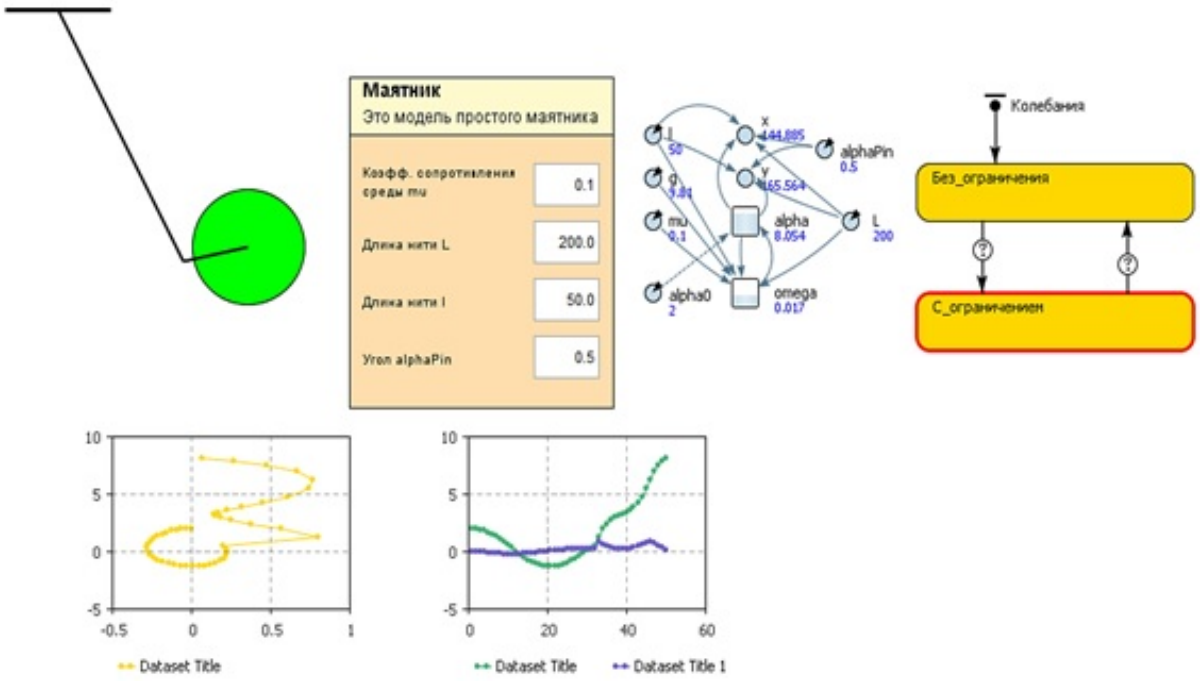


Рис. 38. Модель маятника с ограничением

Контрольные задания

1. Измените модель таким образом, чтобы сила тяжести была направлена по горизонтальной оси.
2. Доработайте презентацию модели таким образом, чтобы при смене направлений движения маятник менял цвет (вверх-вниз).
3. Доработайте презентацию модели таким образом, чтобы при зависании в верхней точке траектории маятник менял цвет.
4. Измените модель таким образом, чтобы сила тяжести была направлена вверх.
5. Доработайте презентацию модели таким образом, чтобы при смене направлений движения маятник менял цвет (вправо-влево).
6. Доработайте модель таким образом, чтобы маятник сопровождала движущаяся рядом надпись, показывающая мгновенное значение линейной скорости.

Лабораторная работа № 4. МОДЕЛЬ ОБРАБОТКИ ДОКУМЕНТОВ В ОРГАНИЗАЦИИ

Цели работы

1. Получить представление о системах массового обслуживания (СМО).

2. Провести имитационное моделирование работы СМО и сравнить полученные результаты с аналитическим решением.

Постановка задачи

Для приёма и обработки документов в организации назначена группа из трёх сотрудников. Ожидаемая интенсивность потока документов – 15 документов в час. Среднее время обработки одного документа одним сотрудником $t_{\text{об. с}} = 12$ мин. Каждый сотрудник может принимать документы из любой организации. Освободившийся сотрудник обрабатывает последний из поступивших документов. Поступающие документы должны обрабатываться с вероятностью не менее 0,95. Определите, достаточно ли назначенной группы из трёх сотрудников для выполнения поставленной задачи [9].

Ход работы

Аналитическое решение

Группа сотрудников работает как СМО, состоящая из трёх каналов, с отказами, без очереди. Поток документов с интенсивностью $\lambda = 15$ (1/ч) можно считать простейшим, так как он суммарный от нескольких организаций. Интенсивность обслуживания $\mu = \frac{1}{t_{\text{об. с}}} = \frac{60}{12} = 5$ (1/ч). Закон распределения неизвестен, но это несущественно, так как показано, что для систем с отказами он может быть произвольным. Граф состояний СМО – это схема «гибели и размножения». Для неё имеются готовые выражения для предельных вероятностей состояний системы:

$$P_1 = \frac{p}{1!}P_0, P_2 = \frac{p^2}{2!}P_0, \dots, P_n = \frac{p^n}{n!}P_0, P_{n+1} = \frac{p^{n+1}}{nn!}P_0, \dots, P_{n+m} =$$

$$= \frac{p^{n+m}}{n^m n!}P_0, P_0 = (1 + \frac{p}{1!} + \cdots + \frac{p^n}{n!} + \frac{p^{n+1}}{nn!} + \frac{p^{n+2}}{n^{2n}} + \cdots + \frac{p^{n+m}}{n^{mn!}}.$$

Отношение $p = \lambda/\mu$ называют приведенной интенсивностью потока документов (заявок). Физический смысл её следующий: величина p представляет собой среднее число заявок, приходящих в СМО за среднее время обслуживания одной заявки. В задаче $p = \frac{\lambda}{\mu} = \frac{15}{5} = 3.$

В рассматриваемой СМО отказ наступает при занятости всех трёх каналов, т. е. при $P_{отк} = P_3$. Тогда

$$P_0 = \left(1 + \frac{3}{1} + \frac{3^2}{2!} + \frac{3^3}{3!}\right)^{-1} = 0{,}77, P_3 = \frac{3^3}{3!\,P_0} = 4{,}5 \cdot 0{,}077 = 0{,}346.$$

Так как вероятность отказа в обработке документов составляет более 0,34 (0,346), то необходимо увеличить количество сотрудников группы. Увеличим состав группы в два раза, т. е. СМО будет иметь теперь шесть каналов, и рассчитаем $P_{отк}$:

$$P_0 = \left(1 + \frac{3}{1} + \frac{3^2}{2!} + \frac{3^3}{3!} + \frac{3^4}{3!} + \frac{3^5}{3!} + \frac{3^6}{6!}\right)^{-1} = 0{,}051,$$

$$P_6 = \frac{3^6}{6!} P_0 = \frac{729}{720} = 4{,}012 \cdot 0{,}051 = 0{,}052.$$

Теперь $P_{об.\ с} = 1 - P_{отк} \approx 0{,}95.$

Таким образом, только группа из шести сотрудников сможет обрабатывать поступающие документы с вероятностью 0,95.

Решение задачи в AnyLogic

Создайте модель **Обработка_документов**. Объекты и элементы указанной модели показаны на рис. 39. Перетащите их на агента Main, разместите, соедините и установите значения свойств согласно таблице.

Для ввода исходных данных используйте элемент **Параметр**, тип первых двух – **double**, третьего – **int**:

1) **срИнтПост** – средний интервал поступления документов, по умолчанию – 4;

2) **срВрОбр** – среднее время обработки документа, по умолчанию – 12;

3) **колСотруд** – количество сотрудников, по умолчанию – 3.

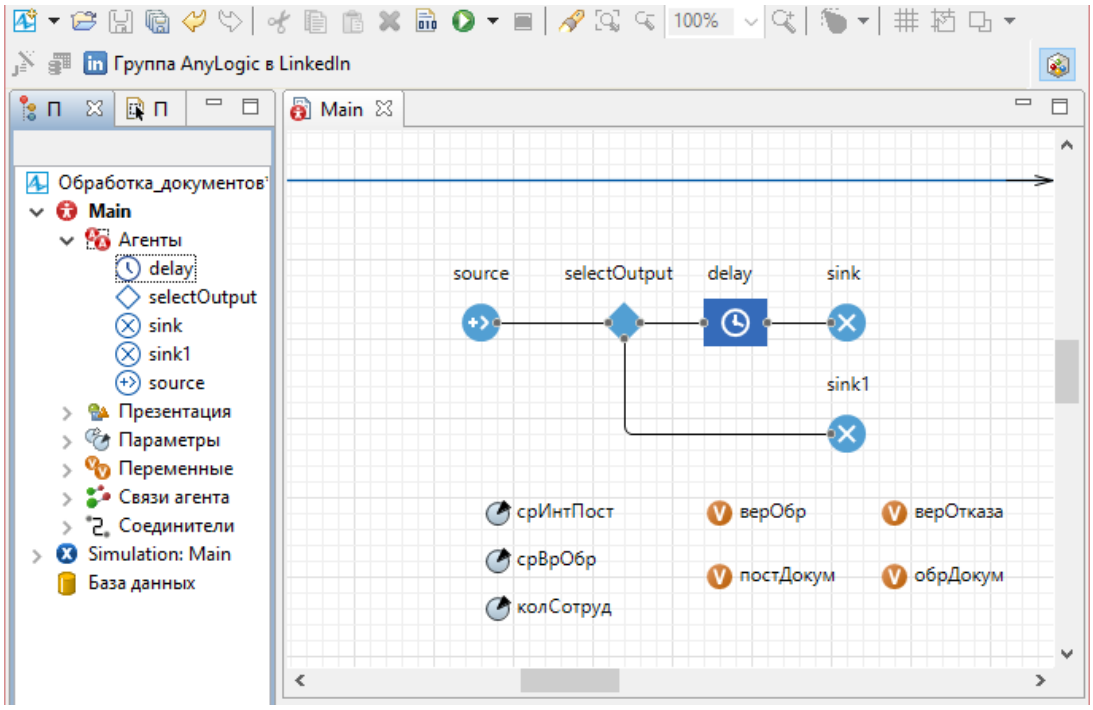


Рис. 39. Модель обработки документов

Объекты и элементы модели ОбрДокументов

Свойство	Значения
Имя	source
Тип заявки	Entity
Прибывают согласно	Времени между прибытиями
Время между прибытиями	exponential(1/срИнтПост)
Действия При выходе:	постДокум++;
Имя	selectOutput
Выход true выбирается	При выполнении условия
Условие	delay.size()<колСотруд
Имя	delay
Тип	Определённое время
Время задержки	exponential(1/срВрОбр)
Вместимость	колСотруд
Включить сбор статистики	Установить флажок
Имя	sink
Действие При входе:	обрДокум++;
	верОбр=обрДокум/постДокум;
	верОтказа=1-верОбр;

Для вывода результатов моделирования используются элементы **Переменная**, тип которых – double:

1) постДокум – количество поступивших документов;
2) обрДокум – количество обработанных документов;
3) верОбр – вероятность обработки документов;
4) верОтказа – вероятность необработки документов.

AnyLogic-модель построена.

Выделите в окне **Проекты** Simulation:Main.

На странице **Модельное время** выберите из списка **Остановить: В заданное время**. Введите **Конечное время**: 600000 (модельное время увеличено в 10000). Режим выполнения: **Виртуальное время** (максимальная скорость).

На странице **Случайность** установите **Фиксированное начальное число** (воспроизводимые прогоны) и **Начальное число**: 1055.

Запустите модель. Вы должны получить результаты, приведенные на рис. 40.

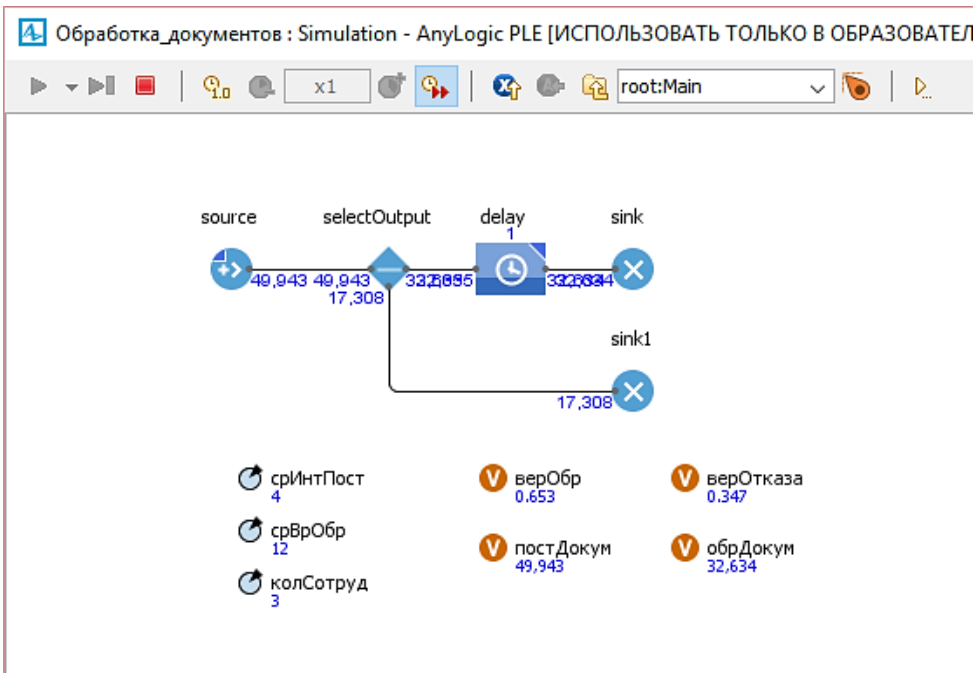


Рис. 40. Результат решения задачи в AnyLogic7

Вероятность необработки всех документов **верОтказа** = 0,347, т. е. отличается от полученного аналитическим путём решения на 0,001. Хотя это отличие можно отнести на счёт округления до трёх знаков после запятой.

Теперь измените количество сотрудников с трёх на шесть. Для этого выделите элемент **Параметр** с именем **колСотруд** и установите по умолчанию 6. Все остальные данные оставьте без изменения. Запустите модель. Вероятность необработки документов **верОтказа** = = 0,053, т. е. отличается от полученного аналитическим путём решения на 0,001.

Сравнительную оценку можно было бы провести и при проведении расчётов с большим числом знаков после запятой, т. е. с большей точностью.

Контрольные задания

1. Какие из полученных результатов можно использовать для предварительной оценки адекватности модели?
2. Каким образом можно задать значение приоритета транзакта?
3. Для приёма и обработки документов в организации назначена группа из пяти сотрудников. Ожидаемая интенсивность потока документов – 30 документов в час. Среднее время обработки одного документа одним сотрудником $t_{\text{об. с}}$ = 10 мин. Каждый сотрудник может принимать документы из любой организации. Освободившийся сотрудник обрабатывает последний из поступивших документов. Поступающие документы должны обрабатываться с вероятностью не менее 0,9. Определите, достаточно ли назначенной группы из пяти сотрудников для выполнения поставленной задачи.

Лабораторная работа № 5. МОДЕЛЬ ОБРАБОТКИ ЗАПРОСОВ СЕРВЕРОМ

Цель работы

Изучить принципы работы СМО, основанных на автоматизированных системах, и провести имитационное моделирование такой системы для оптимизации ее параметров.

Постановка задачи

Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест, с интервалами, распределенными по показательному закону со средним значением 2 мин. Время обработки сервером одного запроса распределено по экспоненциальному закону со средним значением 3 мин. Сервер имеет входной буфер ёмкостью 5 запросов.

Постройте имитационную модель для определения математического ожидания времени и вероятности обработки запросов [9].

Сервер представляет собой однофазную систему массового обслуживания разомкнутого типа с ограниченной входной емкостью, т. е. с отказами, и абсолютной надёжностью (рис. 41).

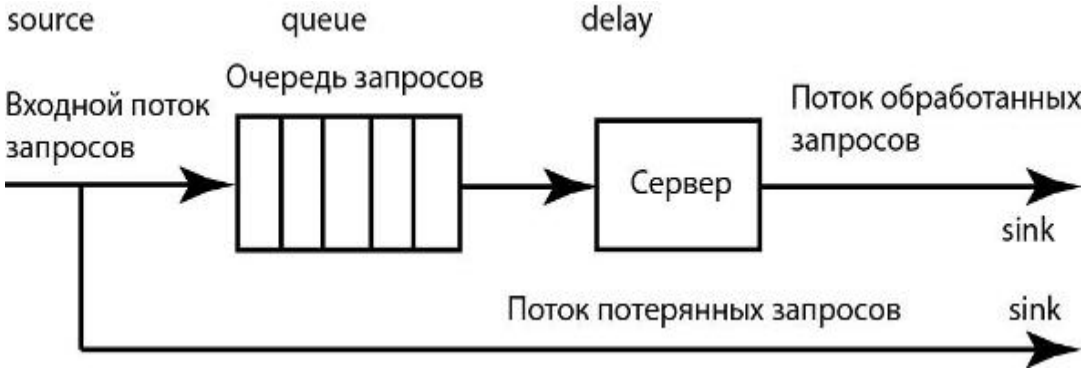


Рис. 41. Сервер как система массового обслуживания

Ход работы

Создание диаграммы процесса

Создадим новую модель, назовём её Server. Создадим диаграмму процесса. Для этого понадобятся четыре элемента, которые необходимо соединить так, как показано на рис. 42.

Объект source генерирует заявки определенного типа. Обычно он используется в качестве начальной точки диаграммы процесса,

формализующей поток заявок. В нашем примере заявками будут запросы на обработку сервером, а объект source будет моделировать их поступление.

Объект queue моделирует очередь заявок, ожидающих приема объектами, следующими за данным в диаграмме процесса. В нашем случае он будет моделировать очередь запросов, ожидающих освобождения сервера.

Объект delay задерживает заявки на заданный период времени. В нашей модели он представляет сервер, обрабатывающий запросы.

Объект sink уничтожает поступившие заявки. Обычно он используется в качестве конечной точки потока заявок (и диаграммы процесса соответственно).

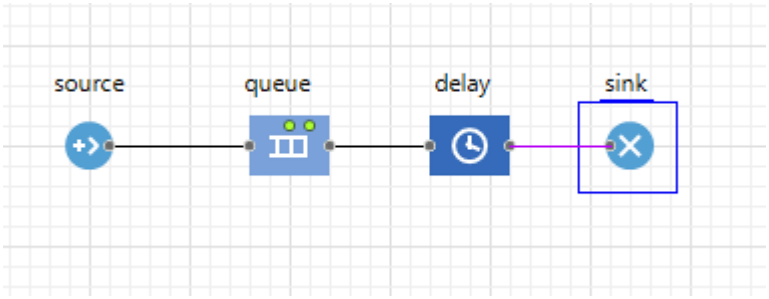


Рис. 42. Диаграмма системы массового обслуживания

Изменение свойств блоков модели, её настройка и запуск

Необходимо помнить, что сначала требуется создать простейшую модель, в которой будет рассматриваться только обработка запросов сервером.

В нашем случае объект source создает заявки через временной интервал, распределенный по показательному (экспоненциальному) закону со средним значением 2 мин. Установим среднее время поступления запросов и среднее время их обработки в секундах.

В выпадающем списке **Прибывают согласно** укажите, что запросы поступают согласно **Времени между прибытиями** (рис. 43).

В поле **Время между прибытиями** появится запись exponential(1). Установите согласно поставленной задаче среднее значение интервалов времени поступления запросов на сервер, изменив свойства объекта source. Для этого вместо характеристики распределения 1 введите 1/120.0.

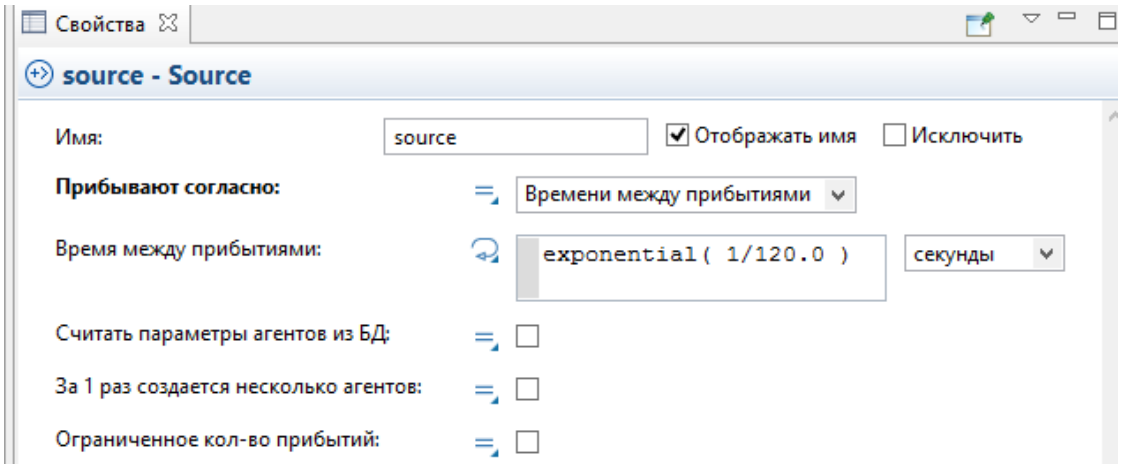


Рис. 43. Свойство source

Следующий объект – queue. Измените свойства объекта queue (рис. 44). Задайте длину очереди. Введите в поле **Вместимость** значение 5. В очереди будут находиться не более 5 запросов.

Установите флажок **Включить сбор статистики**, чтобы включить сбор статистики для этого объекта. В этом случае по ходу моделирования будет собираться статистика по количеству запросов в очереди.

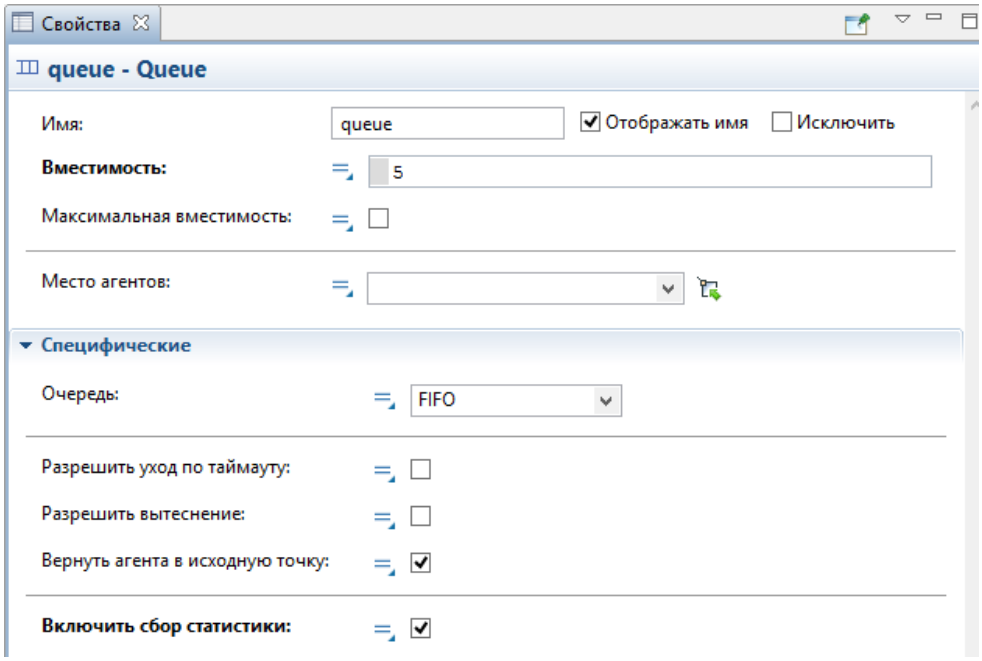


Рис. 44. Свойство queue

Измените свойства объекта delay (рис. 45).

Обработка одного запроса занимает примерно 3 мин. Задайте время обслуживания, распределенное по экспоненциальному закону

со средним значением 3 мин. Для этого введите в поле **Время задержки** exponential(1/180.0). Функция exponential() является стандартной функцией генератора случайных чисел AnyLogic. Установите флажок **Включить сбор статистики**.

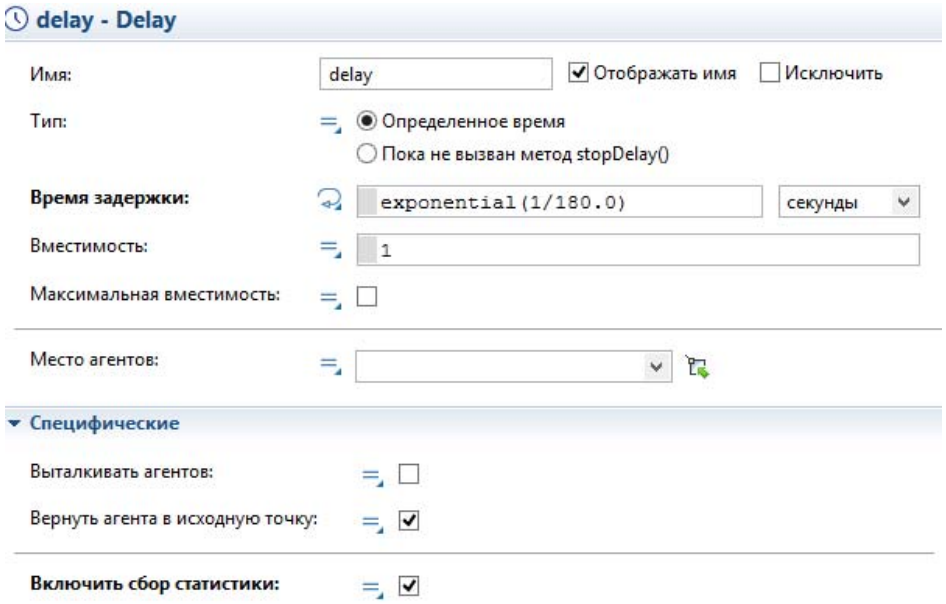


Рис. 45. Свойства delay

Настройка запуска модели

Обработку запросов сервером планируется исследовать в течение одного часа, т. е. 3600 с.

В панели **Проекты** выделите эксперимент **Simulation:Main**. Раскройте вкладку **Модельное время**. Установите **Виртуальное время** (максимальная скорость). В поле **Остановить** выберите из списка **В заданное время**. В поле **Конечное время** установите 3600. Раскройте вкладку **Случайность**. Выберите опцию **Фиксированное начальное число** (воспроизводимые прогоны). В поле **Начальное число** установите 9.

После исправления ошибок и построения модели запустите её (рис. 46).

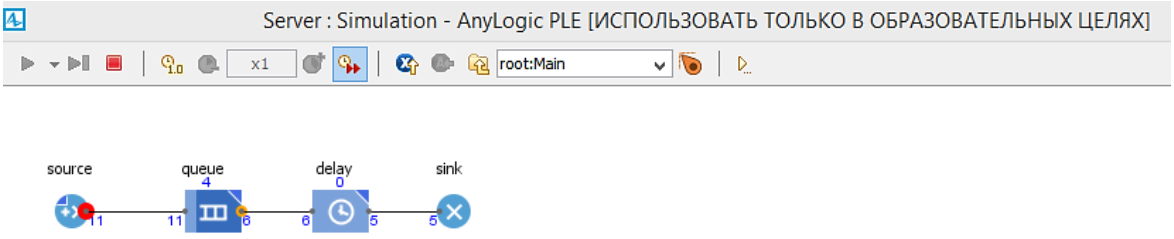


Рис. 46. Запуск модели

Для каждого объекта определены правила, при каких условиях принимать заявки. Некоторые объекты задерживают заявки внутри себя, некоторые – нет. Для объектов также определены правила: может ли заявка, которая должна покинуть объект, ожидать на выходе, если следующий объект не готов её принять. Если заявка должна покинуть объект, а следующий объект не готов её принять, и заявка не может ждать, то модель останавливается с ошибкой (рис. 47). Ошибка означает, что запрос не может покинуть объект source и войти в блок queue, так как его ёмкость, равная 5, заполнена. Также выдаётся сообщение о логической ошибке в модели.

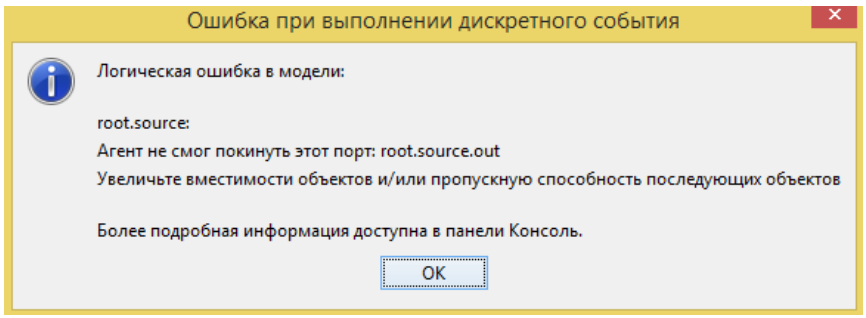


Рис. 47. Ошибка

Нажмите ОК. Далее измените свойства объекта queue, т. е. увеличьте длину очереди. Для этого введите в поле **Вместимость** 15. Можете убедиться, что при увеличении ёмкости в пределах 6 … 14 модель по-прежнему останавливается с этой же ошибкой. Момент появления ошибки зависит от длительности времени моделирования.

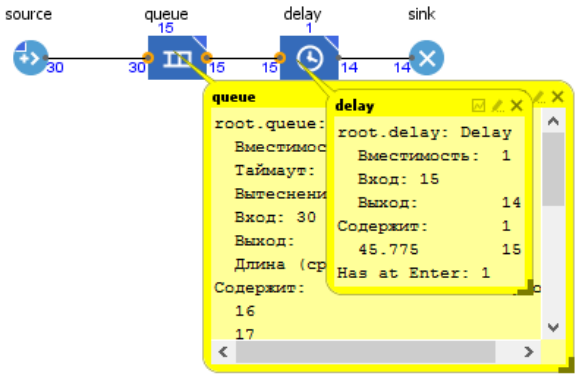


Рис. 48. Второй запуск модели

Снова запустите модель рис. 48). Можно следить за состоянием любого объекта диаграммы процесса во время выполнения модели с помощью окна инспекта этого объекта. В окне инспекта будет отображена базовая информация по выделенному объекту.

Для предотвращения остановок модели по ранее указанной ошибке – недостаточной ёмкости объекта queue – увеличьте ёмкость

объекта queue. Однако можно изменить среднее время имитации поступления запросов объектом source и среднее время обработки запросов сервером, т. е. среднее время задержки объекта delay, оставляя неизменной длину очереди и добиваясь безошибочной работы модели. Конечно, при изменении свойств объектов модели нужно обязательно исходить из целей её построения. Мы же не выполнили условия, указанные в поставленной задаче, поэтому к их выполнению вернемся позже.

Создание анимации модели

Можно наблюдать, анализировать и интерпретировать работу запущенной модели с помощью визуализированной диаграммы процесса. Однако в ряде случаев удобнее иметь более наглядную визуализацию с помощью анимации.

Анимация модели рисуется в той же диаграмме (в графическом редакторе), в которой задается и диаграмма моделируемого процесса.

Так как в данном случае нас не интересует конкретное расположение объектов в пространстве, то можно просто добавить схематическую анимацию интересующих нас объектов – сервер и очередь запросов к нему. Анимация модели рисуется в той же диаграмме (в графическом редакторе), в которой задается и диаграмма моделируемого процесса.

Нарисуйте прямоугольный узел, который будет обозначать на анимации сервер. Поместите элемент **Прямоугольный узел** так, как показано на рис. 49.

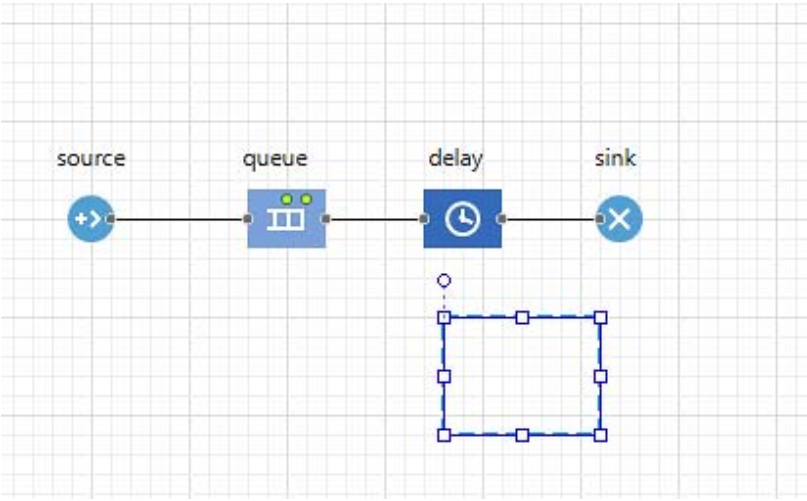


Рис. 49. Прямоугольный узел

Необходимо сделать так, чтобы цвет этого прямоугольного узла менялся в зависимости от того, обрабатывет ли сервер в данный момент времени запрос или нет.

Для этого выделите нарисованную фигуру на диаграмме. Перейдите на страницу **Внешний вид** панели свойств. Необходимо, чтобы во время моделирования менялся цвет фигуры, поэтому щёлкните в поле **Цвет заливки** по стрелке, выберите **Динамическое значение** и введите там следующую строку: delay.size()>0?red:green (рис. 50).

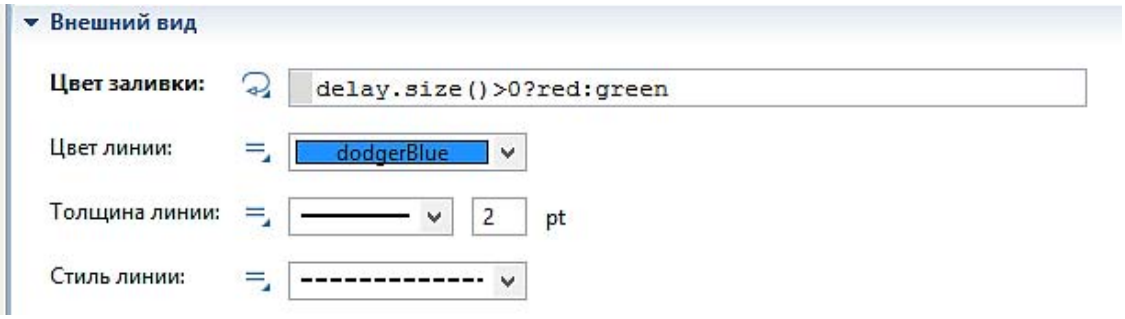


Рис. 50. Цвет заливки

Нарисуйте путь, который будет обозначать на анимации очередь к серверу (рис. 51). Очень важно, какую точку пути вы создаете первой. Заявки будут располагаться вдоль нарисованного вами пути в направлении от конечной точки к начальной.

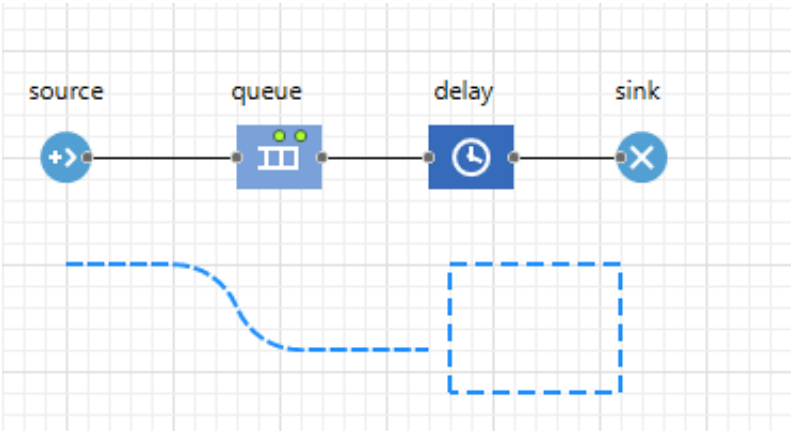


Рис. 51. Путь

Теперь нужно задать созданные анимационные объекты в качестве анимационных фигур объектов диаграммы процесса. Задайте путь в качестве фигуры анимации очереди. Выделите объект queue. На странице свойств объекта queue в поле **Место агентов** выберите из выпадающего списка path.

Задайте прямоугольный узел в качестве фигуры анимации сервера. Выделите объект delay. Введите в поле **Место агентов** из выпадающего списка имя прямоугольного узла: node.

Запустите модель. Теперь у нее есть простейшая анимация – сервер и очередь запросов к нему (рис. 52). Цвет фигуры сервера будет меняться в зависимости от того, обрабатывается ли запрос в данный момент времени или нет.

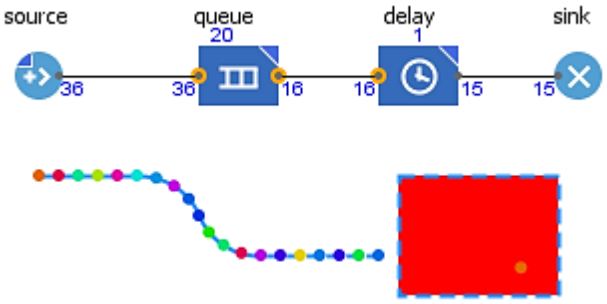


Рис. 52. Фигура сервера

Сбор статистики использования ресурсов

AnyLogic предоставляет пользователю удобные средства для сбора статистики по работе блоков диаграммы процесса. Объекты Enterprise Library самостоятельно производят сбор основной статистики. Все, что нужно сделать, – это включить сбор статистики для объекта.

Добавьте диаграмму для отображения среднего коэффициента использования сервера: перетащите элемент **Столбиковая диаграмма** из палитры **Статистика** на диаграмму класса и измените ее размер, как показано на рис. 53.

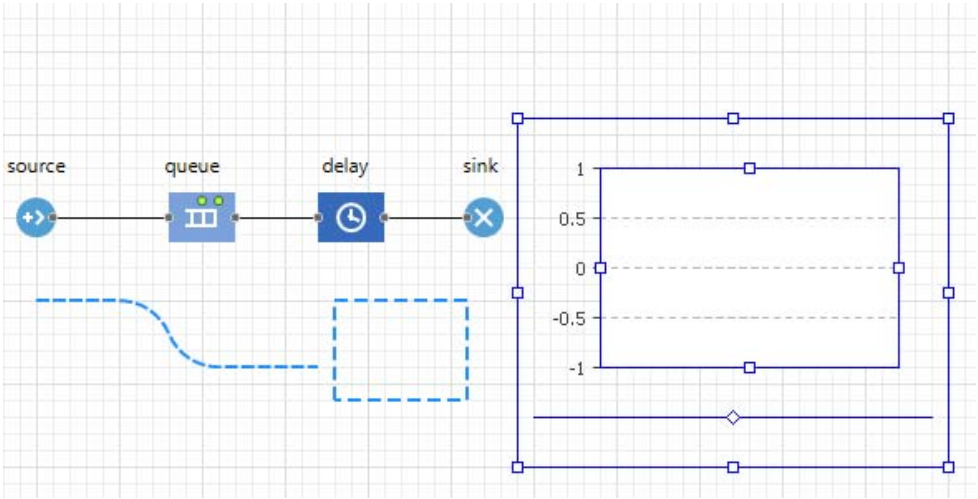


Рис. 53. Столбиковая диаграмма

Перейдите на панель **Свойства**. Щёлкните кнопку **Добавить элемент данных** – появится секция свойств того элемента данных (chart – столбиковая диаграмма), который будет отображаться на этой диаграмме (рис. 54). Измените **Заголовок** на SERVER utilization.

Введите delay.statsUtilization.mean() в поле **Значение**. Здесь delay – это имя объекта delay. У каждого объекта delay есть встроенный набор данных statsUtilization, занимающийся сбором статистики использования этого объекта. Функция mean() возвращает среднее из всех измеренных этим набором данных значений.

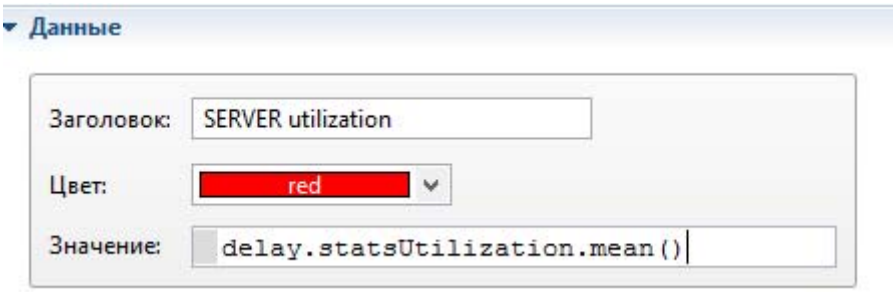


Рис. 54. Цвет

Щёлкните **Внешний вид**. Установите свойства: направление столбцов, цвета фона, границ, меток, сетки, положение подписей у столбцов (рис. 55).

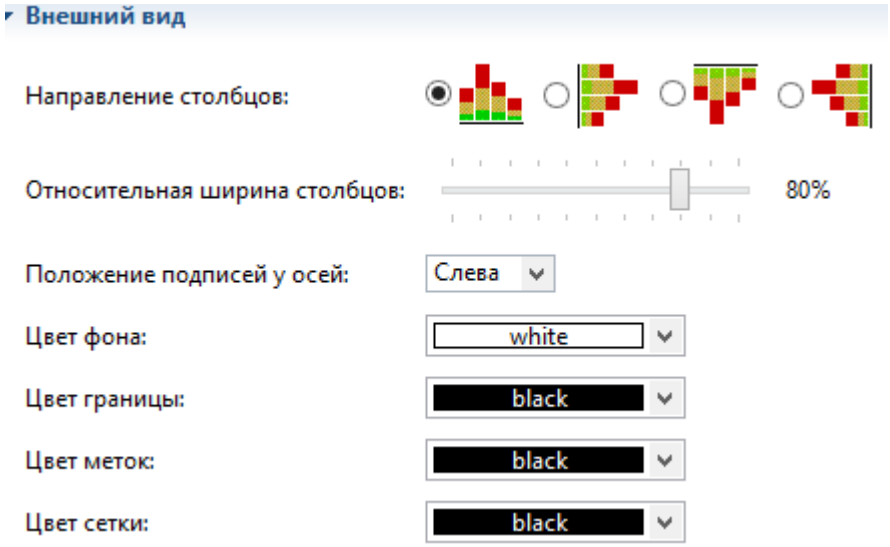


Рис. 55. Внешний вид диаграммы

Откройте страницы (вкладки) **Местоположение и размер**, **Легенда**, **Область диаграммы** (рис. 56). Установите свойства, чтобы изме-

нить расположение легенды относительно диаграммы (необходимо, чтобы она отображалась внизу), размер диаграммы, высоту, ширину, координаты размещения на диаграмме, цвета текста, границы.

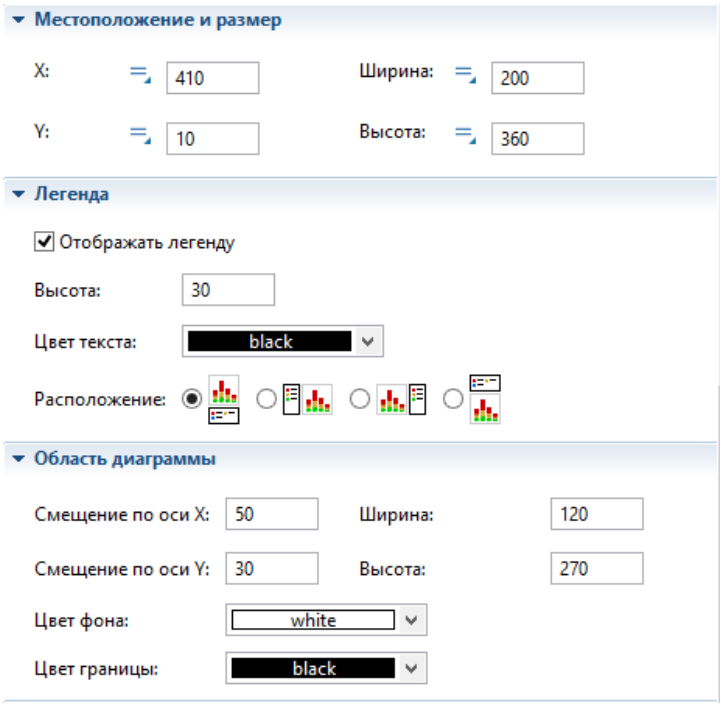


Рис. 56. Параметры диаграммы

Аналогичным образом добавьте еще одну столбиковую диаграмму для отображения средней длины очереди.

На панели **Свойства** щёлкните **Добавить элемент данных**. После щелчка появится страница **Данные свойств элемента данных** (chart1 – столбиковая диаграмма), который также будет отображаться на этой диаграмме.

Заголовок и **Значение** измените так, как показано на рис. 57. В поле **Заголовок** введите Queue lenqth, а в поле **Значение** введите queue.statsSize.mean().

На страницах **Внешний вид**, **Местоположение и размер**, **Легенда**, **Область диаграммы** установите свойства самостоятельно. Столбцы диаграммы должны размещаться горизонтально.

В поле **Значение** выберите queue – это имя объекта queue. У каждого объекта queue, как и объекта delay, также есть встроенный набор данных statsSize, занимающийся сбором статистики использования этого объекта. Функция mean() также возвращает среднее из всех измеренных этим набором данных значений.

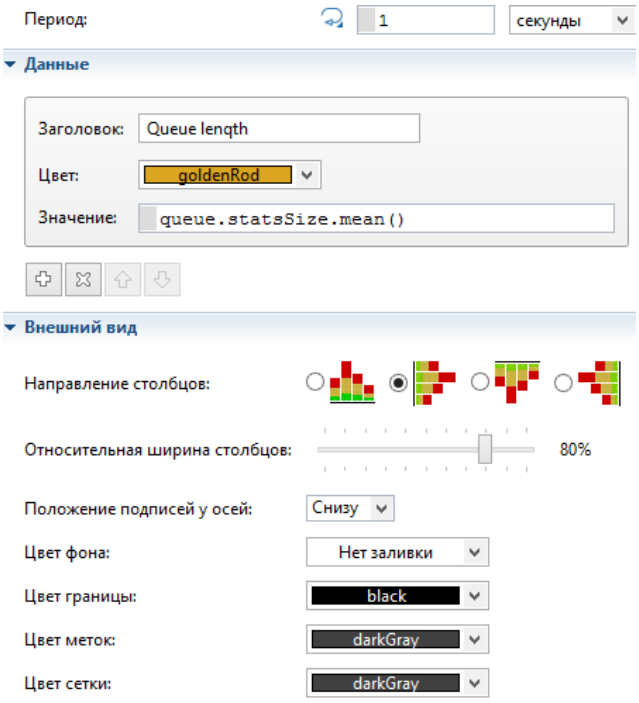


Рис. 57. Параметры диаграммы ***Средняя длина очереди***

На странице **Внешний вид** панели **Свойства** выберите в секции свойств **Направление столбцов** вторую опцию, чтобы столбцы во второй столбиковой диаграмме, расположенной горизонтально, росли вправо.

Запустите модель с двумя столбиковыми диаграммами, установив модельное время 3600 единиц, и понаблюдайте за её работой (рис. 58).

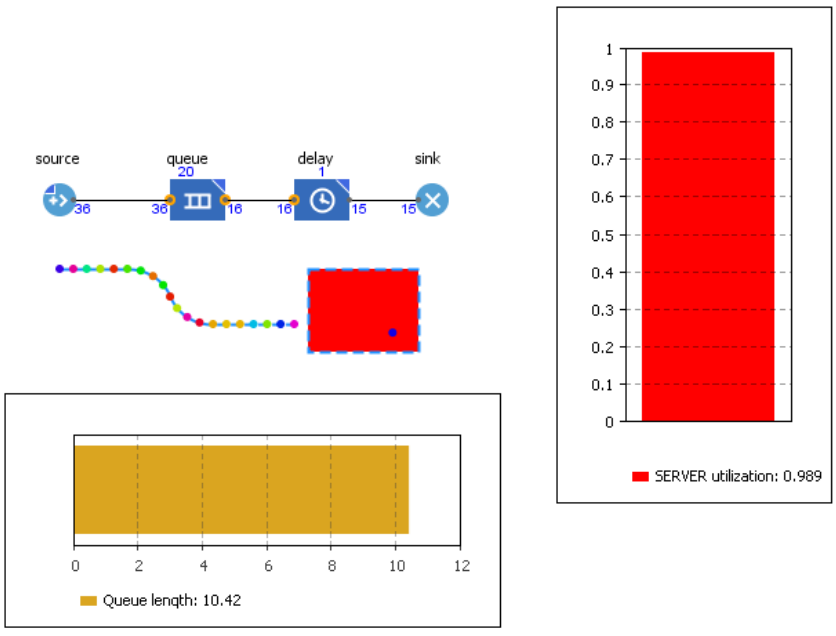


Рис. 58. Работа модели

Уточнение модели согласно ёмкости входного буфера

На рис. 58 (сделан по окончании времени моделирования) видно, что длина очереди равна 16 запросам при установленной максимальной длине 20. Но ведь в постановке задачи ёмкость буфера была определена в 5 запросов. Нам не удалось до этого построить модель с такой ёмкостью из-за ошибки – невозможности очередного запроса покинуть блок source, так как длина очереди уже была равна 5 запросам. Во избежание этой ошибки сначала пришлось увеличить ёмкость буфера до 15 запросов, а затем – до 20.

Возможно ли выполнить данное условие с помощью средств AnyLogic? Оказывается, можно, причем различными способами.

Объект queue моделирует очередь заявок, ожидающих приёма объектами, следующими за ним в потоковой диаграмме, или же моделирует хранилище заявок общего назначения. При необходимости можно задать максимальное время ожидания заявки в очереди, а также с помощью написанной пользователем программы можно извлекать заявки из любых позиций в очереди.

Заявка может покинуть объект queue различными способами:

1) через порт out, когда объект, следующий в блок-схеме за этим объектом, готов принять заявку;

2) через порт outTimeout, если заявка проведет в очереди заданное количество времени (если включен режим таймаута);

3) через порт outPreempted, будучи вытесненной другой поступившей заявкой при заполненной очереди (если включен режим вытеснения);

4) «вручную», путем вызова функции remove() или removeFirst().

В первом случае объект queue покидает заявка, находящаяся в самом начале очереди (в нулевой позиции). Если заявка направлена в порт outTimeout или outPreempted, то она должна покинуть объект мгновенно. Если включена опция вытеснения, то объект queue всегда готов принять новую заявку, в противном случае при заполненной очереди заявка принята не будет.

Поступающие заявки помещаются в очередь в определенном порядке: либо согласно правилу FIFO (в порядке поступления в очередь), либо согласно приоритетам заявок. Приоритет может либо явно храниться в заявке, либо вычисляться согласно свойствам заявки и каким-то внешним условиям. Очередь с приоритетами всегда примет

новую входящую заявку, вычислит её приоритет и поместит в очередь в позицию, соответствующую её приоритету. Если очередь будет заполнена, то приход новой заявки вынудит последнюю хранящуюся в очереди заявку покинуть объект через порт outPreempted. Но если приоритет новой заявки не будет превышать приоритет последней заявки, то тогда вместо неё будет вытеснена именно эта новая заявка.

Для выполнения условия постановки задачи воспользуемся последним способом вытеснения. Все запросы, вырабатываемые объектом source, имеют один и тот же приоритет. Поэтому при полном заполнении накопителя (5 запросов) теряться будет последний запрос. Следует уточнить модель:

1. Выделите объект queue. На панели **Свойства** измените **Вместимость** на 5 запросов.

2. Здесь же установите **Разрешить вытеснение**.

3. Для уничтожения потерянных запросов вследствие полного заполнения накопителя нужно добавить второй объект sink. Добавьте блок sink на диаграмму (рис. 59).

4. Соедините порт outPreempted объекта queue с входным портом InPort блока sink1.

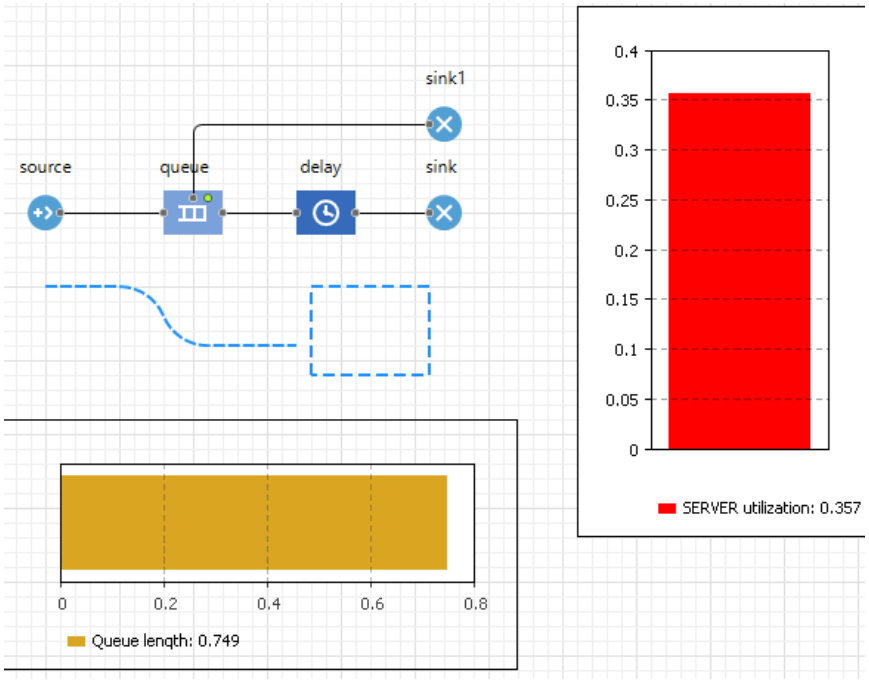


Рис. 59. Sink1

Запустите уточненную модель и понаблюдайте за ее работой. На рис. 60 видно, что запросы при длине очереди в 5 запросов теря-

ются и ошибки при этом не возникает. Модель по ограничению ёмкости входного буфера и значениям других параметров соответствует постановке задачи.

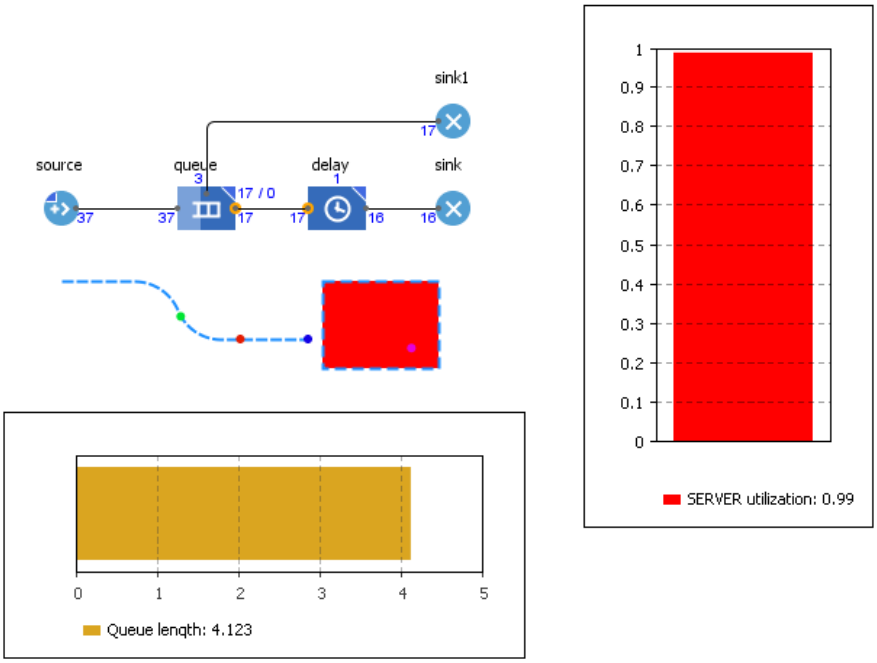


Рис. 60. Уточнённая модель сервера

Однако согласно постановке задачи требуется определить математическое ожидание времени обработки одного запроса и математическое ожидание вероятности обработки запросов.

Сбор статистики по показателям обработки запросов

Согласно постановке задачи нужно определить математическое ожидание времени и вероятности обработки запросов сервером.

Математическое ожидание, или среднее время обработки одного запроса, определяется как отношение суммарного времени обработки n запросов к их количеству, т. е. к n. Для определения суммарного времени нужно знать время обработки i-го запроса. Для этого введем дополнительные поля:

time_vxod – время входа запроса в буфер сервера,

time_vixod – время выхода запроса с сервера (входа в блок sink).

Тогда time_obrabotki=time_vixod-time_vxod

Вероятность обработки запросов сервером определяется как отношение количества обработанных запросов к количеству всех поступивших запросов. Значит, нужно вести счет запросов на выходе

источника запросов и на выходе с сервера (входе в блок sink). Для этого также введем дополнительные поля:

col_vxod – количество поступивших запросов, всего,

col_vixod – количество обработанных сервером запросов.

Тогда ver_obrabotki=col_vixod/col_vxod

Создание нестандартного Java класса

Для включения в запросы дополнительных полей необходимо создать нестандартный тип заявки. Это возможно двумя способами. Создадим первым способом тип заявок Inquiry.

1. В панели **Проект** щёлкните правой кнопкой мыши элемент модели верхнего уровня дерева и выберите из контекстного меню **Создать/Java класс**.

2. Появится диалоговое окно **Новый Java класс**. В поле **Имя** введите имя нового класса Inquiry.

3. В поле **Базовый класс** введите Entity в качестве базового класса. Щёлкните кнопку **Далее**.

4. Появится вторая страница **Мастер создания Java класса**. (рис. 61).

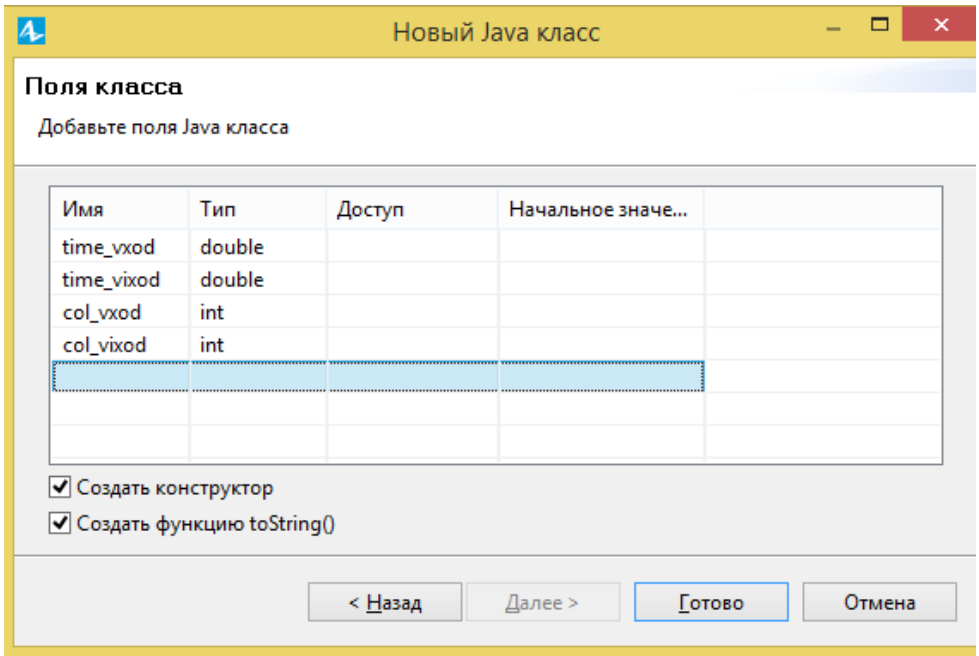


Рис. 61. Создание класса

5. Добавьте поля Java класса: time_vxod типа double, time_vixod типа double, col_vxod типа int, col_vixod типа int.

6. Оставьте выбранными флажки **Создать конструктор** и **Создать метод toString ()**. Тогда у класса будут созданы сразу два конструктора: один, по умолчанию, без параметров, а второй с параметрами, инициализирующими поля класса. Эти конструкторы используются объектами, создающими новые заявки, такие, как source.

7. Щёлкните кнопку **Готово**. Вы увидите редактор кода, в котором будет показан автоматически созданный код вашего Java класса.

8. Теперь нужно преобразовать Java класс в тип агента. Для этого щёлкните правой кнопкой мыши в панели **Проект** по только что созданному Java классу и в контекстном меню выберите **Преобразовать** Java класс в тип агента.

9. Появится окно с автоматически созданными параметрами нестандартного типа заявок Inquiry.

Закройте окно. Удалите всё, что касается только что созданного Java класса и заявки. Перейдём к созданию непосредственно нестандартного типа заявки вторым способом.

1. Создайте тип заявок Inquiry.

2. Откройте палитру **Библиотека моделирования процесов**.

3. Перетащите элемент **Agent** в графический редактор.

4. В поле **Имя нового агента** введите Inquiry.

5. Выберите анимацию агента: установите 2D и выберите из выпадающего списка, например, **Сообщение**.

6. Щёлкните **Далее**.

7. Появится диалоговое окно **Создание агента**. **Шаг 2. Параметры агента**.

8. Щёлкните **Добавить**. В поле **Параметр** введите time_vxod (рис. 62).

9. Из выпадающего списка **Тип** выберите double.

10. Щёлкните второй раз **Добавить**. В поле **Параметр** введите time_vixod.

11. Из выпадающего списка **Тип** выберите double.

12. Щёлкните третий раз **Добавить**. В поле **Параметр** введите col_vxod.

13. Из выпадающего списка **Тип** оставьте int.

14. Щёлкните четвертый раз **Добавить**. В поле **Параметр** введите col_vixod.

15. Из выпадающего списка **Тип** оставьте int.

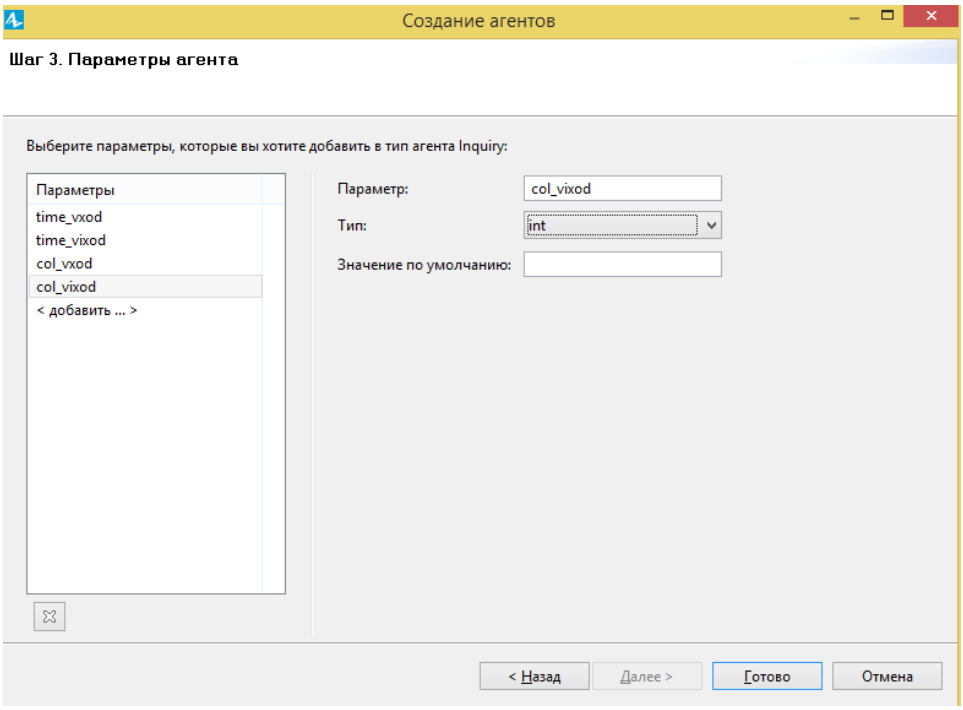


Рис. 62. Типы данных

Так как в поле **Значение** по умолчанию не устанавливаются никакие значения, то всем параметрам будет установлен 0.

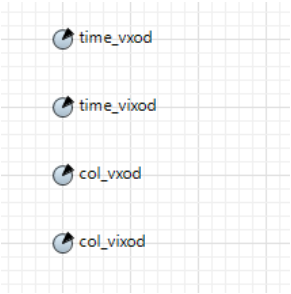


Рис. 63. Inquiry

Щёлкните кнопку **Готово**. Вы увидите окно, в котором будут показаны автоматически созданные параметры нестандартного типа заявок Inquiry (рис. 63).

Добавление элементов статистики

Для сбора статистических данных о времени обработки запросов сервером необходимо добавить элемент статистики. Этот элемент будет запоминать значения времени для каждого запроса. На основе этого он предоставит пользователю стандартную статистическую информацию (среднее, минимальное, максимальное из измеренных значений, среднеквадратичное отклонение и т. д.).

Добавьте элемент **Данные Гистограммы** (рис. 64):

- измените **Имя** на time_obrabotki;
- сделайте **Кол-во интервалов** равным 50;
- задайте **Нач. размер интервала** 0.01.

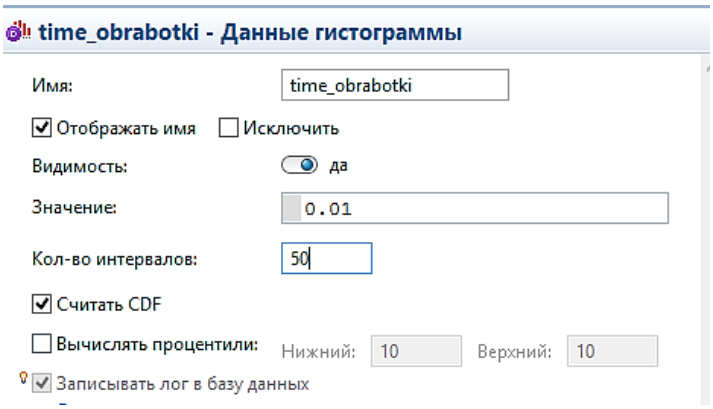


Рис. 64. Элемент сбора статистики о времени обработки запросов

Добавьте элемент **Данные Гистограммы** (рис. 65):

- измените **Имя** на ver_obrabotki;
- сделайте **Кол-во интервалов** равным 50;
- задайте **Нач. размер интервала** 0.01.

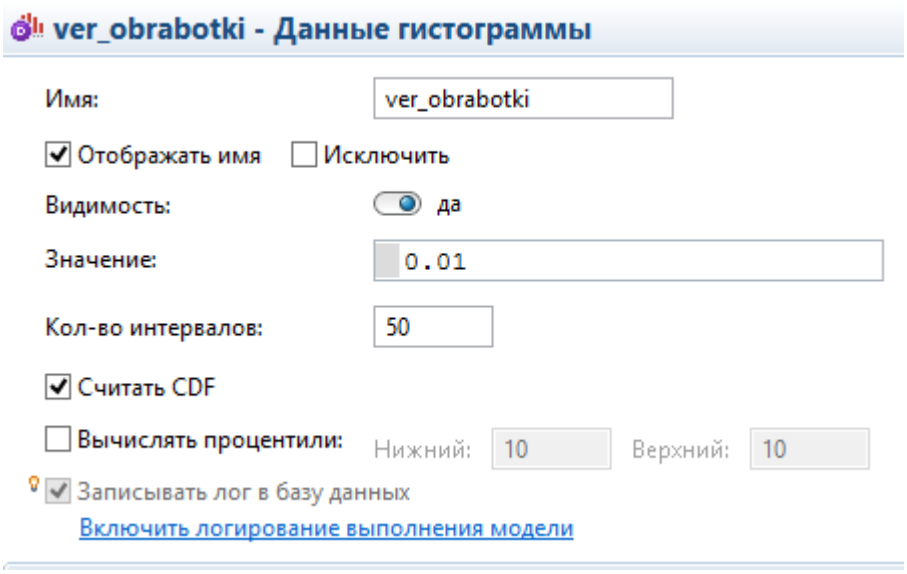


Рис. 65. Элемент сбора статистики о вероятности обработки запросов

Изменение свойств объектов диаграммы

Для того чтобы создавать заявки нестандартного типа, как в нашем случае Inquiry, необходимо поместить вызов конструктора этого типа в поле **Новая заявка объекта source**. Но, несмотря на то, что заявки в потоке теперь будут типа Inquiry, остальные объекты диаграммы будут продолжать их считать заявками типа Entity. По-

этому они не позволят явно обращаться к дополнительным полям класса Inquiry. Для того чтобы разрешить доступ к полям нестандартного типа заявки в коде динамических параметров объектов потоковой диаграммы, нужно указать имя нестандартного типа заявки в качестве типа заявки этого объекта.

В рассматриваемой потоковой диаграмме с учётом блока source всего пять объектов. Измените их свойства. Измените свойства объекта source (рис. 66):

1. Измените на Inquiry в поле **Тип заявки**. Это позволит напрямую обращаться к полям типа заявки Inquiry в коде динамических параметров этого объекта.

2. Выберите из выпадающего списка Inquiry() в поле **Новый агент**. Теперь этот объект будет создавать заявки необходимого типа Inquiry.

3. Введите agent.time_vxod=time(); в поле **Действия при выходе**. Код будет сохранять время создания заявки-запроса в параметре time_vxod указанного типа заявки Inquiry.

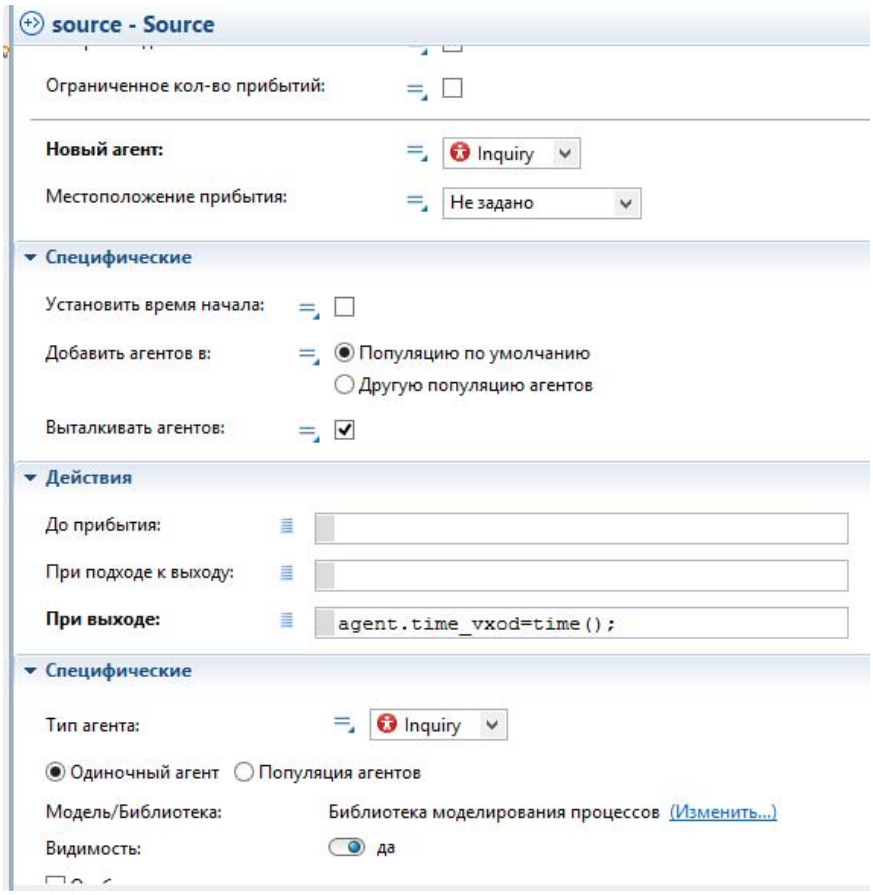


Рис. 66. Тип заявки

Функция time() возвращает текущее значение модельного времени.

1. Измените свойства объекта queue: введите Inquiry в поле **Тип агента**.

2. Измените свойства объекта delay: введите Inquiry в поле **Тип агента**.

3. Измените свойства объекта sink1: введите Inquiry в поле **Тип агента**.

4. Измените свойства объекта sink: введите Inquiry в поле **Тип агента** и введите в поле **Действие при входе** следующие коды:

а) time_obrabotki.add(time()-agent.time_vxod);

Этот код добавляет время обработки одного запроса в объект сбора данных гистограммы time_obrabotki. Данное время определяется как разность между текущим модельным временем time() и временем входа запроса в модель. add – встроенная функция добавления элемента в массив;

б) agent.col_vixod=sink.count();

agent.col_vxod=source.count();

Эти коды заносят количество запросов, вошедших в блок sink и вышедших из блока source соответственно. count() – встроенная функция этих блоков, возвращает количество вошедших в блок sink и количество вышедших из блока source заявок;

в) ver_obrabotki.add(agent.col_vixod/ agent.col_vxod);

Этот код добавляет относительную долю обработанных запросов в объект сбора данных гистограммы ver_obrabotki при поступлении каждого обработанного запроса в блок sink. На основе множества таких относительных долей определяется математическое ожидание вероятности обработки запросов сервером.

Запустите модель. Появится сообщение об ошибке. Щёлкните **Продолжить**. Появится второе сообщение (рис. 67).

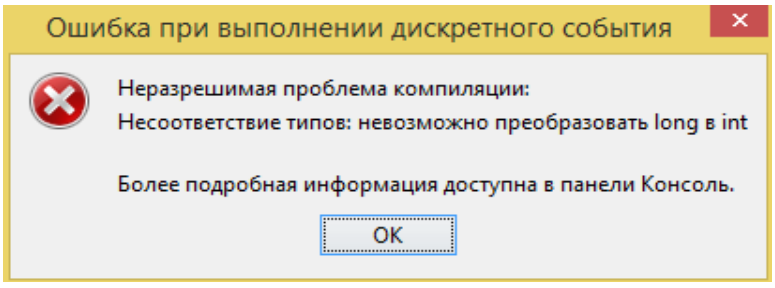


Рис. 67. Ошибка при выполнении дискретного события

Мы установили тип int для col_vxod и col_vixod. Изменим этот тип на double.

Удаление и добавление новых полей типа заявок

Обратите внимание, что вам не пришлось использовать поле time_vixod, так как вместо него была использована функция time(), возвращающая, как известно, текущее значение модельного времени.

Удалите поле time_vixod.

Из процедуры удаления поля следует, что можно вводить новые дополнительные поля нестандартного типа заявок.

Итак, все условия задачи выполнены. Для того чтобы наблюдать за работой модели, установите, что время остановки модели не задано. Запустите модель (рис. 68).

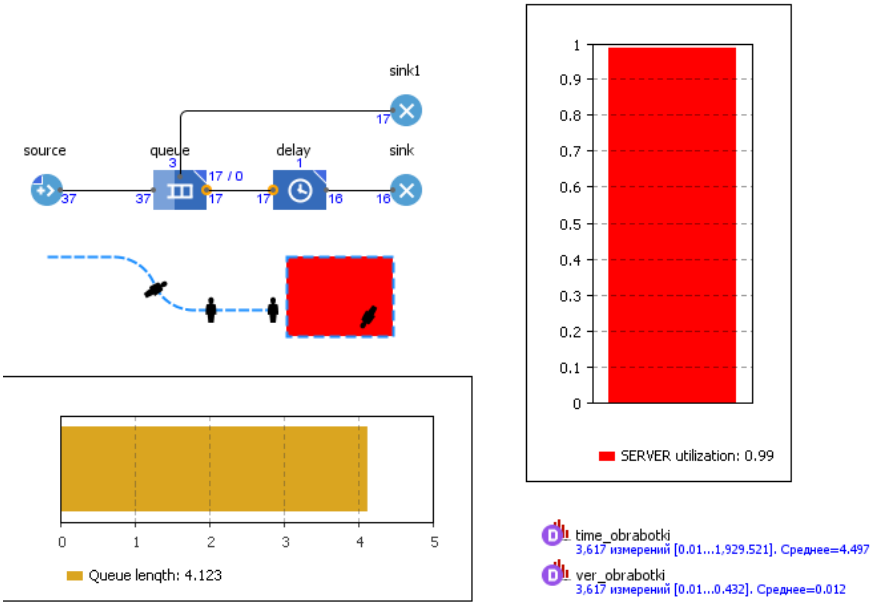


Рис. 68. Фрагмент работы модели сервера

Добавление параметров и элементов управления

Активный объект может иметь параметры. Обычно их используют для задания статических характеристик объекта. Но значения параметров при необходимости можно изменять во время работы модели. Для этого нужно написать код обработчика события, т. е. действий, которые должны выполняться при изменении значения параметра.

1. Создайте параметр time_mean объекта delay.
2. Добавьте объект **Параметр**.
3. Перейдите на панель **Свойства** (рис. 69).
4. В поле **Имя** введите имя параметра time_mean (среднее время). По этому имени параметр будет доступен из кода.
5. Задайте тип параметра double.

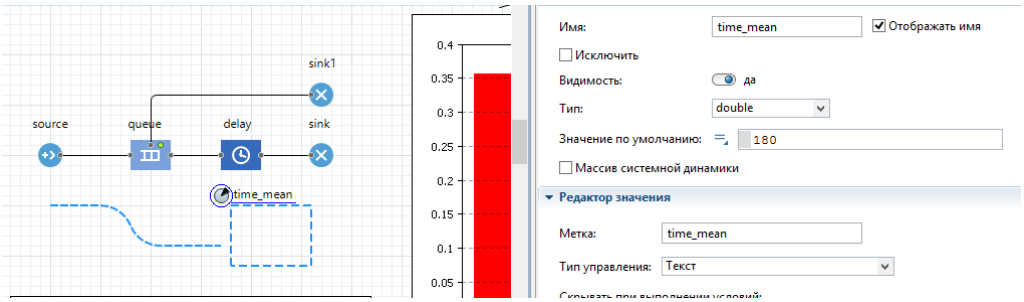


*Рис. 69. Объект **Параметр***

6. В поле **Значение** по умолчанию установите 180. Если значение не задано явно, по правилам Java оно будет равно нулю.

7. Выделите объект delay.

8. На панели **Свойства** в поле **Время задержки** вместо выражения exponential(1/180.0) введите выражение exponential(1/time_mean).

9. Если необходимо изменить среднее время обработки запросов time_mean в ходе моделирования, используйте для этого элемент управления – бегунок.

10. Добавьте элемент **Бегунок**.

11. Поместите бегунок под параметром time_mean, чтобы было понятно, что с помощью этого бегунка будет меняться среднее время обработки запросов объектом delay.

12. Если необходимо варьировать среднее время от 1 до 300, то введите 1 в поле **Минимальное значение**, а 300 – в поле **Максимальное значение** (рис. 70).

13. Установите флажок **Связать с** и в активизированное поле введите time_mean.

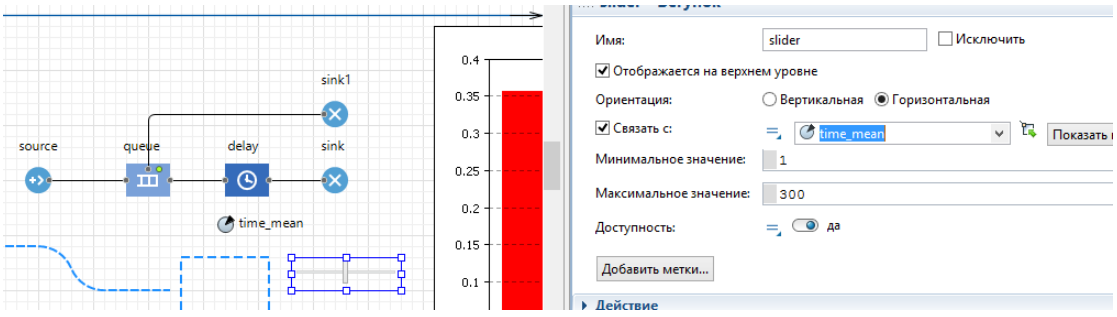


Рис. 70. Элемент «Бегунок»

Для изменения ёмкости буфера в ходе моделирования также используйте бегунок.

1. Добавьте ещё один бегунок под элементом queue.

2. Если необходимо варьировать ёмкость буфера от 0 до 15 запросов, то введите 15 в поле **Максимальное значение**.

3. Установите флажок **Связать с** и в активизированное поле введите queue.capacity.

4. Запустите модель. Теперь в процессе моделирования можно изменять ёмкость входного буфера и среднее время обработки запросов с помощью бегунков.

Существует и другой способ изменения свойств объектов во время выполнения модели: нужно щёлкнуть по элементу, войти в режим редактирования и ввести новое значение в одной из закладок всплывающего окна инспекта. Поэтому заранее не нужно продумывать, значения каких параметров планируется изменять, и добавлять специальные элементы управления (например, бегунки).

Изменение значения в окне инспекта поддерживается для следующих элементов: простая переменная, параметр, накопитель и для следующих типов: численные; логический (boolean); текстовый (String).

Удалите элемент **Бегунок** для параметра time_mean. Запустите модель и приостановите её. Щёлкните по значку параметра time_mean. Введите новое значение: 240.0. Закройте инспект. Рядом с элементом **Параметр** вы увидите введённое вами значение 240.0. Запустите модель с новым свойством объекта delay (рис. 71).

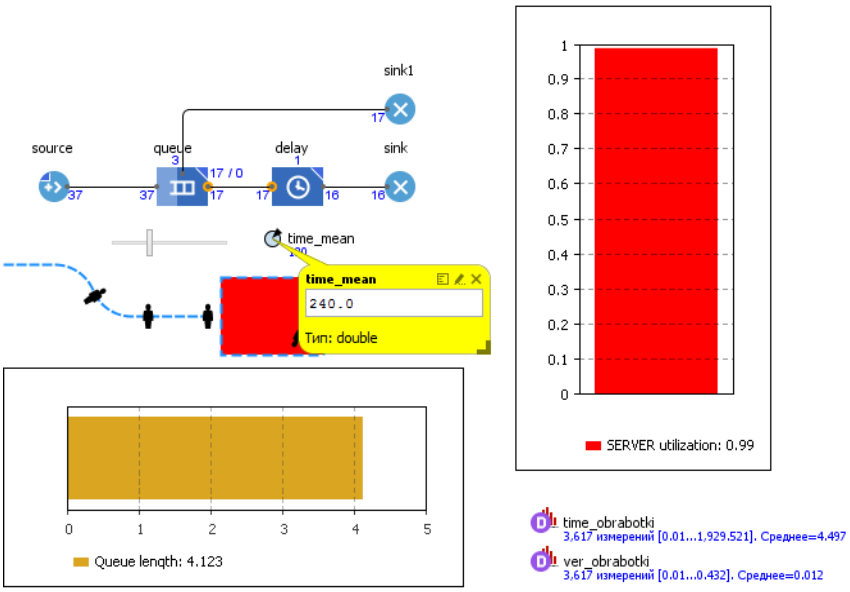


Рис. 71. Ввод нового значения параметра в окно инспекта

Добавление гистограмм

Добавим на диаграмму рассматриваемого потока гистограмму, которая будет отображать собранную временную статистику.

1. Добавьте элемент **Гистограмма**.

2. Укажите, какой элемент сбора данных хранит данные, которые вы хотите отображать на гистограмме: щёлкните кнопку **Добавить данные** и введите в поле **Данные** имя соответствующего элемента: time_obrabotki. Установите **Отображать среднее**.

3. В поле **Заголовок** введите Histogram Time obrabotki (рис. 72).

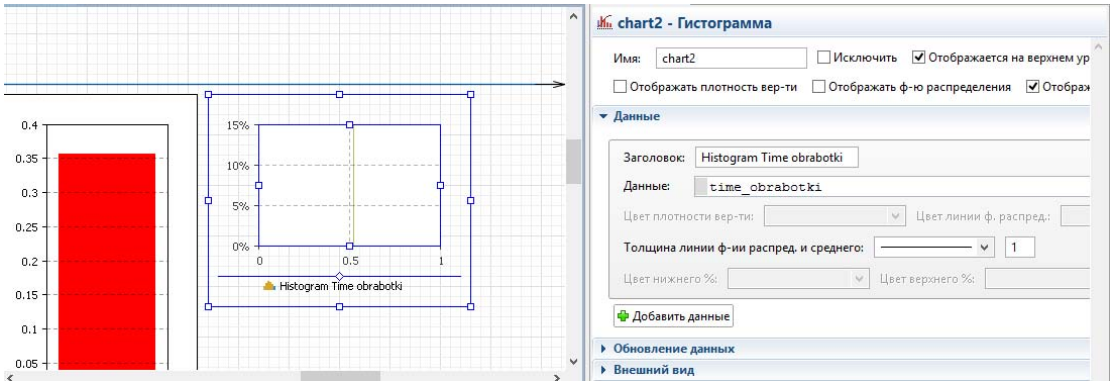


Рис. 72. Histogram Time obrabotki

Добавим на диаграмму потока гистограмму, которая будет отображать собранную вероятностную статистику.

4. Добавьте элемент **Гистограмма**.

5. Щёлкните кнопку **Добавить данные** и введите в поле **Данные** имя элемента: ver_obrabotki. Установите **Отображать среднее**.

6. В поле **Заголовок** введите Histogram Ver obrabotki.

7. Запустите модель. Фрагмент работы показан на рис. 73.

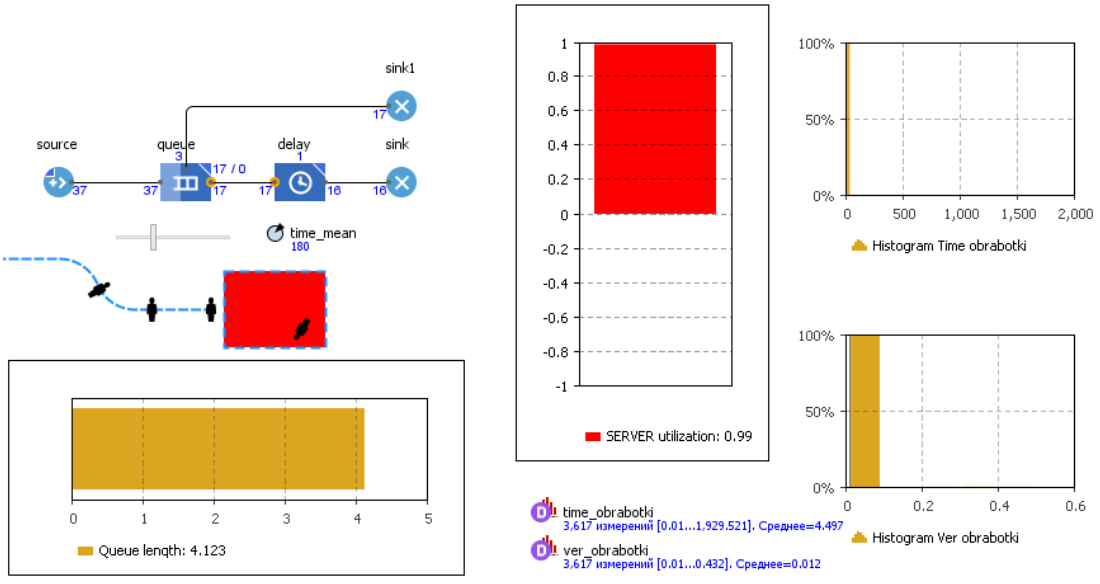


Рис. 73. Фрагмент работы модели с элементом управления и гистограммами

Обратите внимание, что после нового запуска модели time_mean =180, хотя ранее мы изменили его значение на 240.

Изменение времени обработки запросов сервером

Построенная модель соответствует постановке задачи. В ней, с целью упрощения процесса построения первой модели, время обработки запросов сервером было принято распределённым по показательному (экспоненциальному) закону со средним значением T2 = 3 мин.

Однако в модели время обработки поступающих запросов зависит от производительности сервера $Q = 6 \cdot 10^5$ оп/с и вычислительной сложности запросов, распределенной по нормальному закону с математическим ожиданием $S_1 = 2 \cdot 10^7$ оп и среднеквадратическим отклонением $S_2 = 2 \cdot 10^5$ оп.

Кроме того, в модели определяется среднее количество запросов, обработанных за время моделирования 3600 с.

Внесите в модель изменения для аналогичного расчёта времени обработки запросов.

1. Удалите элемент **Параметр** с именем time_mean, элемент **Бегунок** для элемента queue.
2. Добавьте три элемента **Параметр**.
3. В поле **Имя** каждого из элементов введите S1_, S2_ и Q_ соответственно. Выберите **Тип double**.
4. В поле **Значение по умолчанию** каждого из элементов введите 60000000, 200000 и 600000 соответственно.
5. Перетащите элемент **Переменная**.
6. В поле **Имя** укажите KolZap.
7. Выделите объект delay. В поле **Время задержки** вместо exponential(1/time_mean) введите (normal(S2_,S1_))/Q_
8. Выделите объект sink. В поле **Действие** при входе к имеющемуся там коду добавьте код KolZap=sink.in.count()/9604.0; (рис. 74).

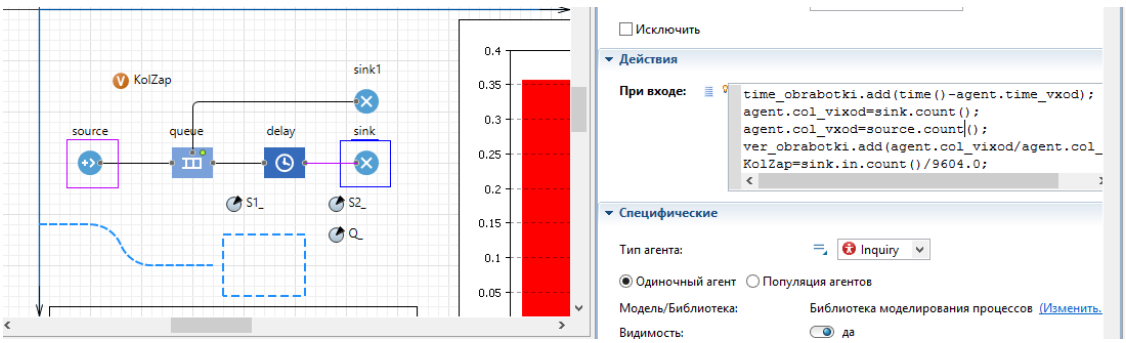


Рис. 74. Переменные

Для получения результатов моделирования с доверительной вероятностью α = 0,95 и точностью ε = 0,01 нужно выполнить 9604 прогона модели:

$$N = t_{\alpha}^{2}\frac{\rho(1-\rho)}{\varepsilon^{2}} = 1{,}96^{2}\frac{0{,}5^{2}}{0{,}01^{2}} \approx 9604,$$

где t_{α} – табулированный аргумент функции Лапласа, t_{α} = 1,96; p – ожидаемая вероятность исхода события, в данном случае вероятность обработки запросов сервером.

Расчёт проведен для так называемого «худшего» случая, т. е. в предположении, что ожидаемая вероятность обработки запросов p = 0,5.

Увеличим время моделирования в AnyLogic-модели в 9604 раза. Так как статистические данные о количестве обработанных запросов собираются за всё время моделирования, увеличенное в 9604 раза, то для получения среднего значения это количество нужно разделить на 9604, что и предусмотрено в коде.

9. Показатели моделируемой системы нужно определить в течение 3600 с, поэтому время моделирования в AnyLogic составит 3600 × × 9604 = 34574400 единиц модельного времени.

10. В панели **Проект** выделите Simulation. На странице **Модельное время** в поле **Установить** выберите **В заданное время**.

11. В поле **Конечное время** установите 34574400.

12. Запустите модель и дождитесь окончания моделирования.

Результаты моделирования приведены на рис. 75.

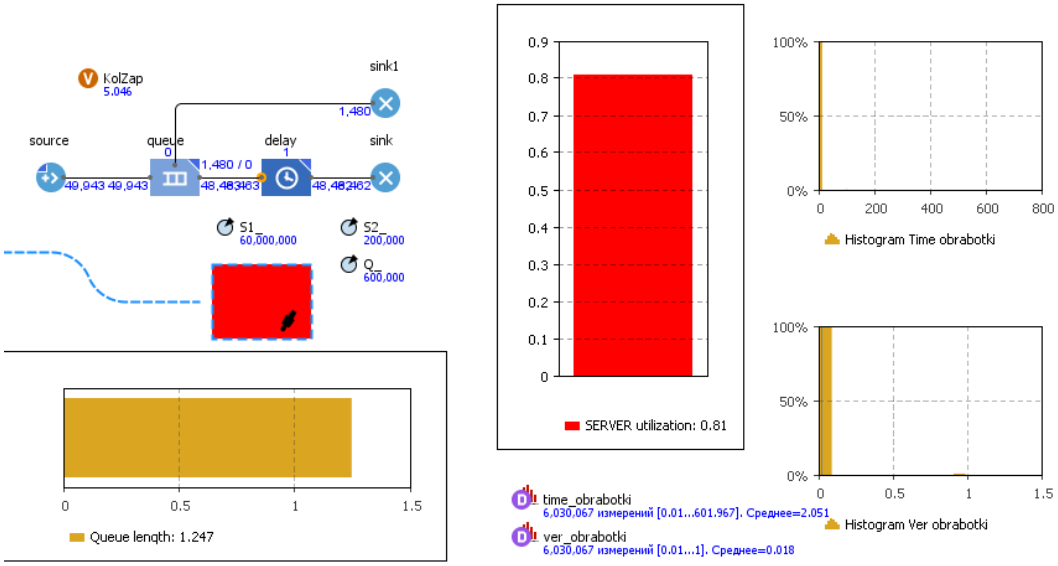


Рис. 75. Результаты моделирования обработки запросов сервером

Интерпретация результатов моделирования

Для проведения исследований на модели сделайте ещё несколько дополнений и изменений. Можно было бы обойтись и без них, но они улучшат эксплуатацию модели.

1. Из палитры **Основная** перетащите три элемента **Параметр** на диаграмму агента Main. Разместите их выше параметров S1_, S2_, Q_.

2. Выделите первый элемент **Параметр**. В поле **Имя** первого параметра введите timeMean – среднее время поступления запросов для обработки на сервере. Оставьте тип double.

3. В поле **Значение по умолчанию** введите 120.

4. Выделите объект source. В поле **Время между прибытиями** вместо 120.0 введите timeMean. Теперь при изменении среднего времени поступления запросов не придётся искать нужный код в свойствах объекта модели.

5. Выделите второй элемент **Параметр**. В поле **Имя** введите kolProg – количество прогонов модели. Как и в предыдущем случае, при корректировке числа прогонов код в свойствах искать будет не нужно. Оставьте тип double.

6. В поле **Значение по умолчанию** введите 9604.

7. Выделите объект sink.

8. В поле **Действие при входе** в имеющемся там коде замените последнюю строку на KolZap=round(sink.in.count()/kolProg);

Количество обработанных запросов KolZap сервером выдавалось с дробной частью. Теперь, вследствие применения процедуры round, количество запросов будет целым.

Как уже было отмечено, при изменении количества прогонов модели не нужно будет искать код для требуемой корректировки. Однако всё-таки потребуется изменить модельное время. Например, если нужно выполнять с моделью 1000 прогонов, то модельное время следует установить равным $3600 \cdot 1000 = 3600000$.

9. Выделите третий элемент **Параметр**. В поле **Имя** введите emkBuf – ёмкость в сообщениях входного буфера сервера. Установите тип int.

10. В поле **Значение по умолчанию** введите 5.

11. Выделите объект queue. В поле **Вместимость** введите emkBuf. (рис. 76).

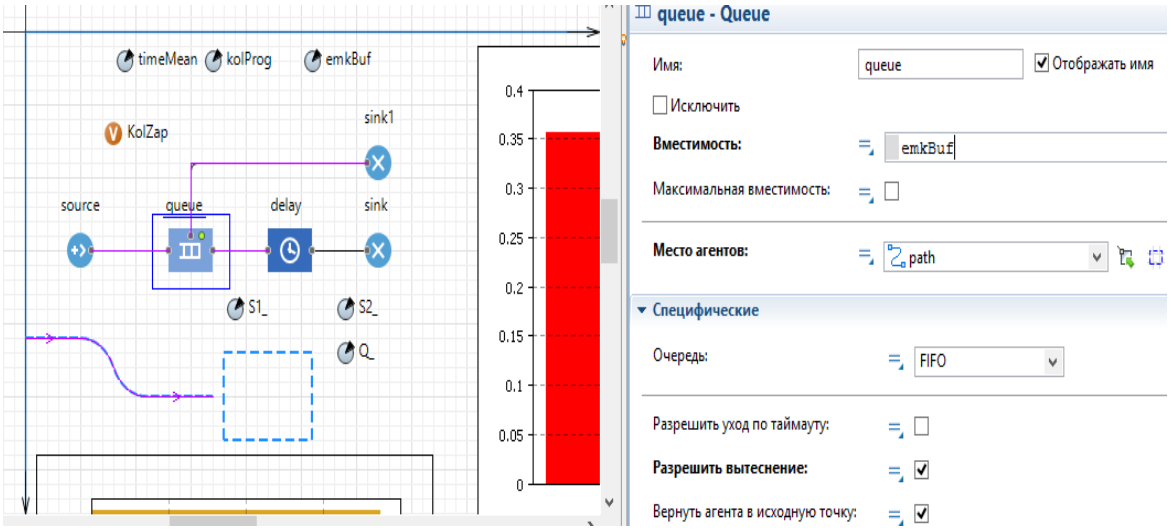


Рис. 76. Новые параметры

12. Запустите модель. Результаты моделирования показаны на рис. 77.

Проведём несколько экспериментов. Будем изменять *среднее время* поступления запросов timeMean в предположении, что с течением времени количество источников запросов будет расти, т. е. timeMean будет уменьшаться. В сторону увеличения будем изменять ёмкость входного буфера emkBuf и *производительность Q* сервера.

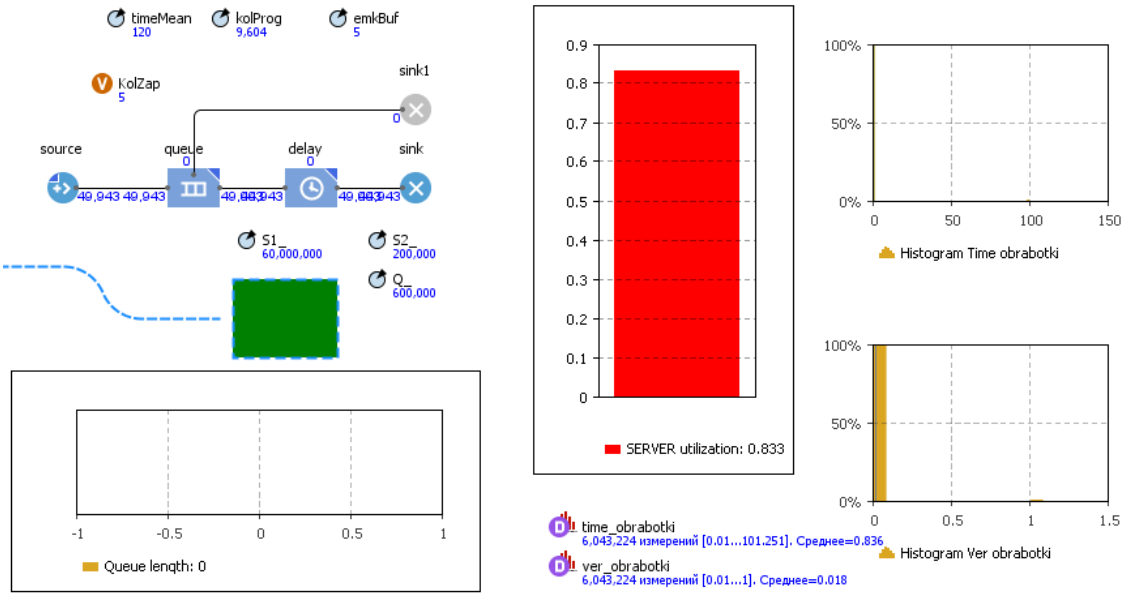


Рис. 77. Результаты моделирования при условиях постановки задачи

Контрольные задания

1. Проанализируйте методику разработки моделей в AnyLogic, разобрав ее на примере выполненной лабораторной работы.

2. Сервер обрабатывает запросы, поступающие с автоматизированных рабочих мест с интервалами, распределенными по показательному закону со средним значением 1,5 мин. Время обработки сервером одного запроса распределено по экспоненциальному закону со средним значением 4 мин. Сервер имеет входной буфер ёмкостью 7 запросов. Постройте имитационную модель для определения математического ожидания времени и вероятности обработки запросов.

3. Какие изменения необходимо внести в модель, чтобы представить работу двух параллельно работающих серверов? Проанализируйте результаты первого и второго эксперимента с одним и двумя серверами.

ЗАКЛЮЧЕНИЕ

Сегодня имитационное моделирование с большим успехом используется для анализа и проектирования сложных систем в различных областях человеческой деятельности. Имитационные модели дают возможность эффективно экспериментировать над системами в различных условиях, а программно-технические средства являются неотъемлемыми элементами данного процесса.

В условиях экономического кризиса, когда государству и предприятиям жизненно необходимо четко оценивать свои возможности и ресурсы для развития в ближайшей и долгосрочной перспективе, имитационное моделирование становится особенно актуальным и представляет собой универсальный подход для принятия решений в условиях неопределённости.

Освоив теоретические и практические навыки имитационного моделирования, будущие специалисты будут иметь возможность анализировать объекты исследования с учетом различных аспектов (временного, пространственного и логического), используя при этом объектно ориентированное программирование, что позволяет решать задачи высокого уровня сложности.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. Объект в ООП (Объектно-ориентированном программировании) [Электронный ресурс]. Режим доступа: http://itandlife.ru/programming/object-oriented-programming/obektno-orientirovannoe-prog-rammirovanie-oop-ponyatie-obekta/ (дата обращения: 25.07.2016).

2. *Маликов Р. Ф.* Практикум по имитационному моделированию сложных систем в среде AnyLogic 6 : учеб. пособие. Уфа : Изд-во БГПУ, 2013. 296 с.

3. Системный анализ сложных систем управления [Электронный ресурс]. Режим доступа: http://kiipt-uad.narod.ru/doc/SA/sa02.html (дата обращения: 26.07.2016).

4. *Абракитов В. Э.* На пути к научным открытиям : монография [Электронный ресурс]. Режим доступа: http://abrakitov.narod.ru/Monograph/1-6.htm (дата обращения: 30.07.2016).

5. Системный анализ процесса управления в сложных системах [Электронный ресурс]. Режим доступа: http://lektsii.com/6-37727.html (дата обращения: 30.07.2016).

6. *Климушев Н. К., Прудникова О. М.* Моделирование технологических процессов лесопромышленного производства : учеб. пособие. Ухта : УГТУ, 2003. 76 с.

7. Сложные системы в процессе построения имитационных моделей [Электронный ресурс]. Режим доступа: http://info-tehnologii.ru/IMIT_MOD/Sl_sis/index.html (дата обращения: 01.08.2016).

8. Особенности построения имитационных моделей [Электронный ресурс]. Режим доступа: http://info-tehnologii.ru/IMIT_MOD/Kl_IM_MOD/Postr_mod/index.html (дата обращения: 01.08.2016).

9. Концептуальное проектирование систем в AnyLogic и GPSS World [Электронный ресурс]. Режим доступа: http://www.intuit.ru/studies/courses/4818/1066/lecture/16367 (дата обращения: 05.08.2016).

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. *Кобелев, Н. Б.* Имитационное моделирование объектов с хаотическими факторами: учеб. пособие / Н. Б. Кобелев. – М. : КУРС : НИЦ Инфра-М, 2016. – 192 с. – ISBN 978-5-906818-20-1.

2. *Сосновиков, Г. К.* Компьютерное моделирование. Практикум по имитационному моделированию в среде GPSS World : учеб. пособие / Г. К. Сосновиков, Л. А. Воробейчиков. – М. : Форум : НИЦ Инфра-М, 2015. – 112 с. – ISBN 978-5-00091-035-1.

3. *Кобелев, Н. Б.* Имитационное моделирование : учеб. пособие / Н. Б. Кобелев, В. А. Половников, В. В. Девятков ; под общ. ред. д-ра экон. наук Н. Б. Кобелева. – М. : КУРС : НИЦ Инфра-М, 2013. – 368 с. – ISBN 978-5-905554-17-9.

4. *Девятков, В. В.* Методология и технология имитационных исследований сложных систем: современное состояние и перспективы развития : монография / В. В. Девятков. – М. : Инфра-М, 2013. – 448 с. – ISBN 978-5-9558-0338-8.

5. *Емельянов, А. А.* Имитационное моделирование экономических процессов : учеб. пособие / А. А. Емельянов, Е. А. Власова, Р. В. Дума ; под ред. А. А. Емельянова. – М. : Финансы и статистика, 2008. – 368 с. – ISBN 5-279-02572-0.

6. *Козлов, А. Ю.* Статистический анализ данных в MS Excel : учеб. пособие / А. Ю. Козлов, В. С. Мхитарян, В. Ф. Шишов. – М. : Инфра-М, 2014. – 320 с. – ISBN 978-5-16-004579-5.

7. *Лычкина, Н. Н.* Имитационное моделирование экономических процессов : учеб. пособие / Н. Н. Лычкина. – М. : Инфра-М, 2012. – 254 с. – ISBN 978-5-16-004675-4.

8. AnyLogic – многоподходное имитационное моделирование [Электронный ресурс]. – Режим доступа: http://www.anylogic.ru (дата обращения: 06.08.2016).

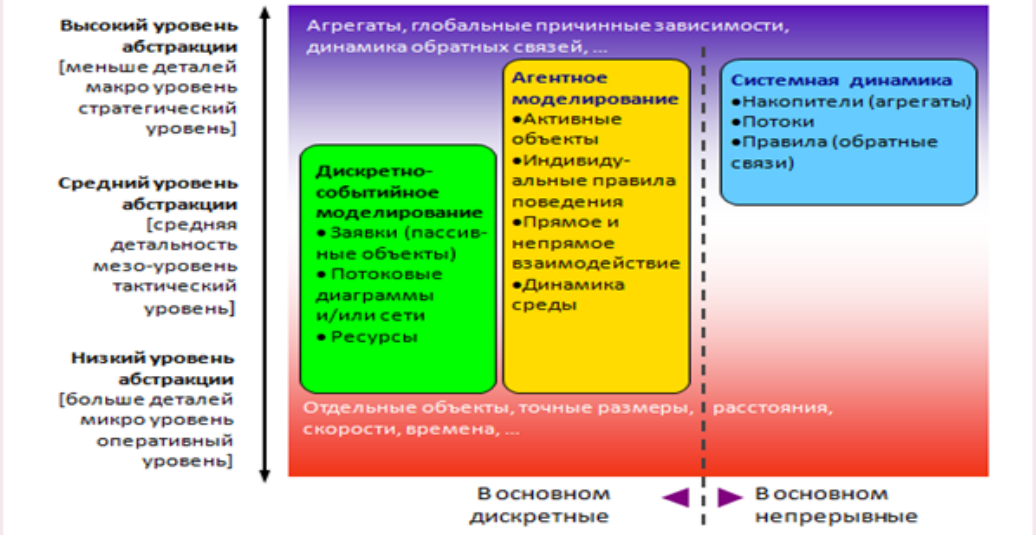
ПРИЛОЖЕНИЕ

Справочная информация по программе AnyLogic

Три подхода имитационного моделирования

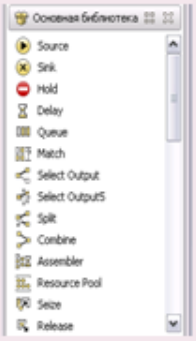
Продукт получил название AnyLogic, потому что он поддерживает все три известных метода моделирования:

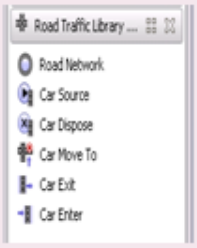
Парадигмы моделирования

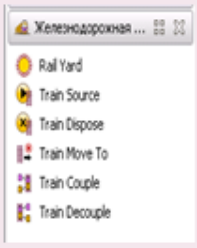


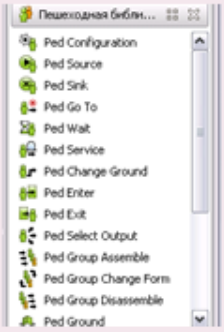
Встроенные библиотеки

- **Основная библиотека AnyLogic;**
- **Пешеходная библиотека AnyLogic;**
- **Железнодорожная библиотека;**
- **Библиотека Дорожного движения.**









Графическая среда AnyLogic

Графическая среда моделирования AnyLogic поддерживает проектирование, разработку, документирование модели, выполнение компьютерных экспериментов с моделью, включая различные виды анализа. Включает в себя следующие элементы:

- *Stock & Flow Diagrams*;
- *Statecharts*;
- *Action charts*;
- *Process flowcharts*.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ 5

ЛАБОРАТОРНЫЙ ПРАКТИКУМ 13

Лабораторная работа № 1. Первая модель на ANYLOGIC: «Создание модели мячика» 13

Лабораторная работа № 2. Стейтчарты: модель пешеходного перехода 29

Лабораторная работа № 3. Построение модели маятника 38

Лабораторная работа № 4. Модель обработки документов в организации 47

Лабораторная работа № 5. Модель обработки запросов сервером 52

ЗАКЛЮЧЕНИЕ 81

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ 82

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ 83

ПРИЛОЖЕНИЕ 84

Учебное издание

РЕМЕЗОВА Екатерина Максимовна

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В СРЕДЕ ANYLOGIC

Лабораторный практикум

Редактор Е. А. Лебедева
Технический редактор С. Ш. Абдуллаева
Корректор О. В. Балашова
Компьютерная верстка Е. А. Кузьминой

Подписано в печать 05.10.17.
Формат 60×84/16. Усл. печ. л. 5,11. Тираж 60 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.