Изучение имитационного моделирования в AnyLogic в углубленном курсе информатики

И.А. Калинин,
ФГБОУ ВПО Гос. ИРЯ им. А.С. Пушкина,

Н.Н. Самылкина,
ФГБОУ ВПО МПГУ,

А.И. Калугин,
ФГБОУ ВПО МПГУ

В предыдущей статье этого цикла [6] мы рассказали вам о мощном методе исследования и поиска решений — имитационном моделировании. Продолжая этот рассказ, мы рассмотрим сегодня еще несколько моделей.

Начнем с одного из видов агентной модели — заодно вспомнив основы работы среды AnyLogic.

Напоминаем, что бесплатная версия продукта для использования в школах и лицензионное соглашение можно найти по адресу: *http://metodist.lbz.ru/authors/informatika/8/*.

Простейшая модель распространения эпидемии

Одна из ежегодных проблем крупного города — эпидемии гриппа. Нам требуется создать модель, которая позволит изучить действенность некоторых типовых мер по борьбе с эпидемиями — вакцинации, карантина и т.д.

Для решения этой задачи мы воспользуемся агентной моделью. Моделировать пространство целого города и поведение людей для нас не имеет смысла — поскольку для нас в этой модели не важно, как люди перемещаются по городу. Нам важно только, как численно распространяется заболевание — а для этого “гонять” человечков не надо.

При этом фактически агент будет просто менять свое состояние последовательно: здоров, заражен, болен и выздоровел. Для простоты будем считать, что выздоровевший больше не заболеет.

Мы будем считать, что заболеть человек может двумя путями: “случайно” (т.е. без непосредственного

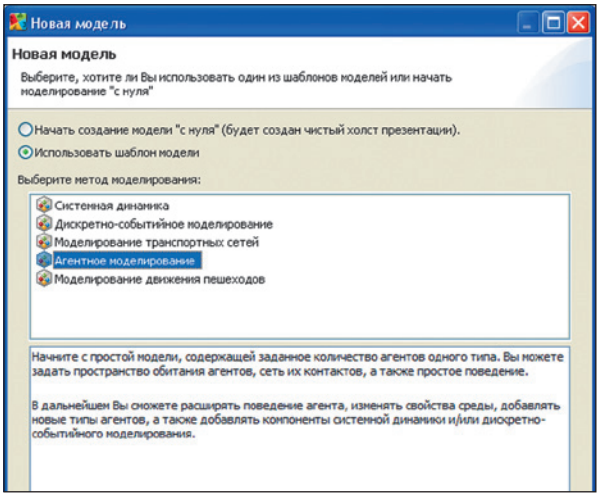
контакта с заболевшим) и при непосредственном контакте с зараженным или больным. Так что для нас существенными параметрами болезни будут:

1) вероятность заражения “из воздуха”;
2) инкубационный период;
3) среднее время течения заболевания.

Это, конечно, очень упрощенная модель[1], но для проверки основных предположений она вполне годится.

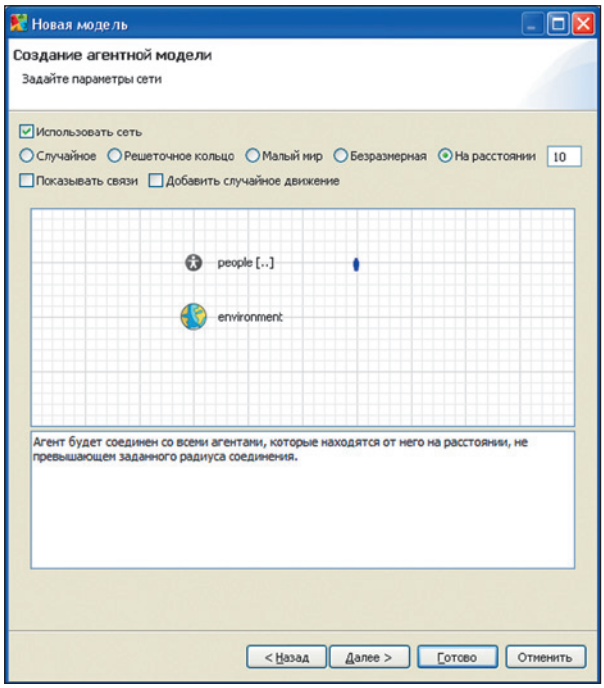
Ключевой вопрос, который мы будем исследовать с помощью нашей модели, это вопрос о параметрах, влияющих на скорость и широту распространения болезни. Опять-таки, упрощая, можем считать ключевой точкой каждого эксперимента момент времени, в который больных больше всего.

Для создания модели мы воспользуемся мастером. Обратимся к команде “Файл/Создать/Модель”, на первом шаге укажем название “Epidemic”, на втором шаге — что за основу берем агентную модель:



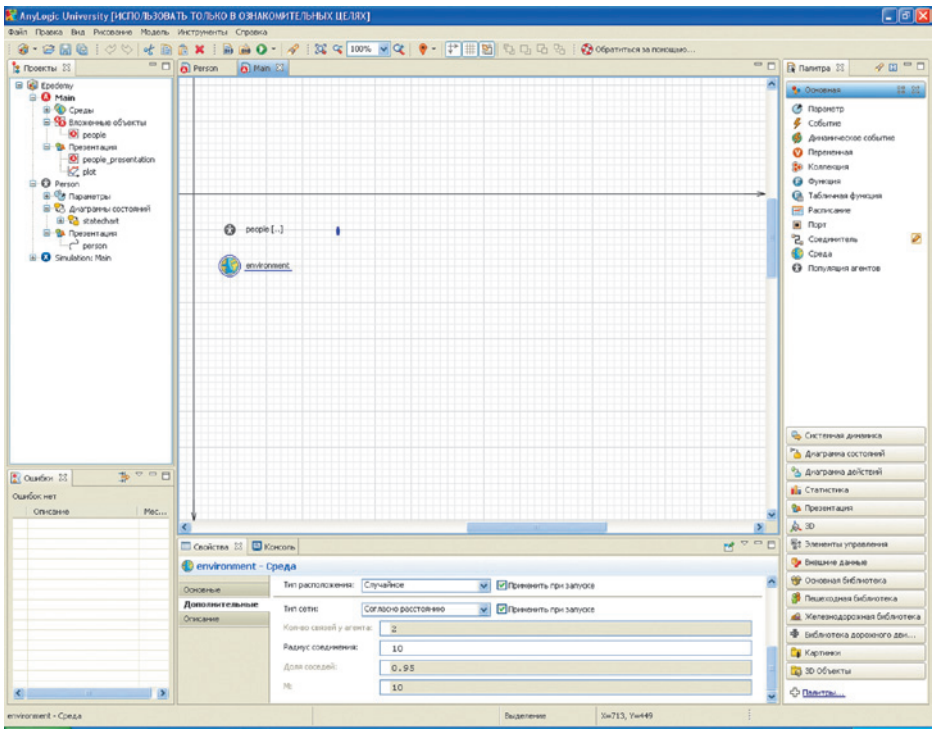
На следующем шаге укажем, что нам понадобится 500 агентов, далее — что распределены они будут случайно, размер пространства — 400 × 400 (то есть ничего не будем менять).

На шестом шаге укажем, что контактировать объекты будут на расстоянии до 10 единиц:



Больше ничего добавлять не будем и создадим модель — нажмем кнопку “Готово”.

Получим примерно следующее:



[1] Вот пример более аккуратной реализации подхода: *http://ipi-cpl.ru/about/articles/kondratyev_epidemic_ntv_2010*.

Мы создали модель, в которой при запуске будет создано 500 агентов на пространстве 400 на 400 единиц. Каждый агент будет “связываться с соседями на расстоянии 10 единиц. Распределятся они по всему пространству случайно, но происходить с ними пока ничего не будет — мы никаких действий им не приписали.

Наша модель, напомним, устроена таким образом, что перемещения агентов нас не интересуют, и фактически агенты будут просто менять свое состояние. Чтобы определить, какие состояния когда будут возникать, мы должны открыть объект “Person” в графическом редакторе и создать диаграмму смены состояний, StateChart.

Наша диаграмма будет показывать последовательность состояний во время заболевания: сначала человек здоров, потом заражен (но еще не знает об этом и свободно перемещается), болеет и, наконец, выздоравливает.

Откроем палитру “Диаграмма состояний” и перетащим на рабочее пространство объекта блоки. Блокам дадим названия: healthOk (Здоров), infected (Заражен), sick (Болен) и cured (Вылечен). Чтобы их было легче различать, сменим цвет заливки.

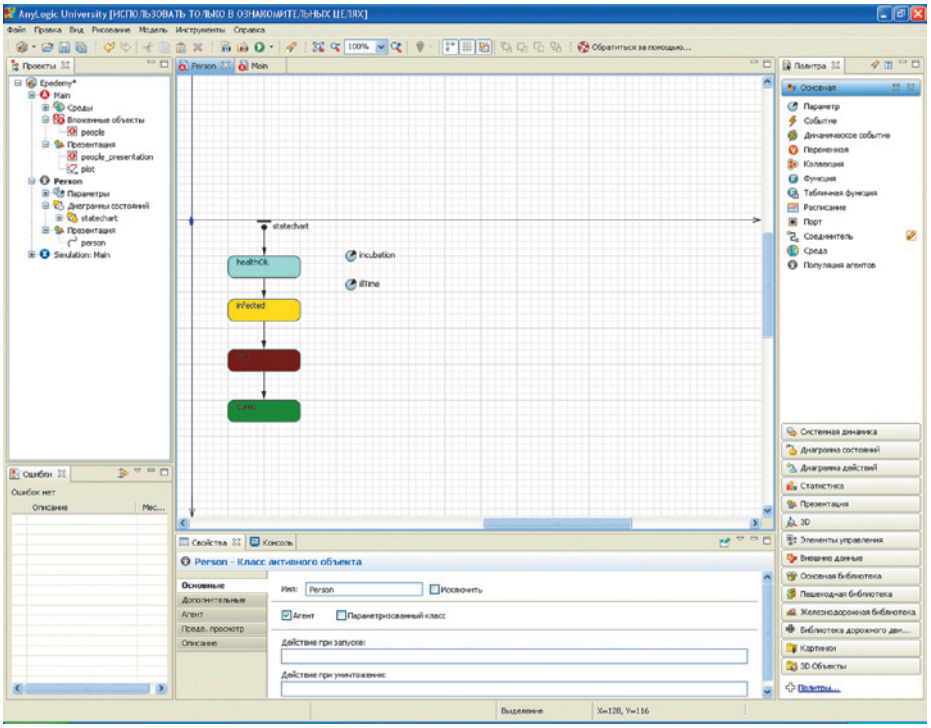
Для простоты предположим, что:

1. Выздоравливают все[2].
2. Время вылечивания не меняется.
3. Выздоровевший повторно не болеет.

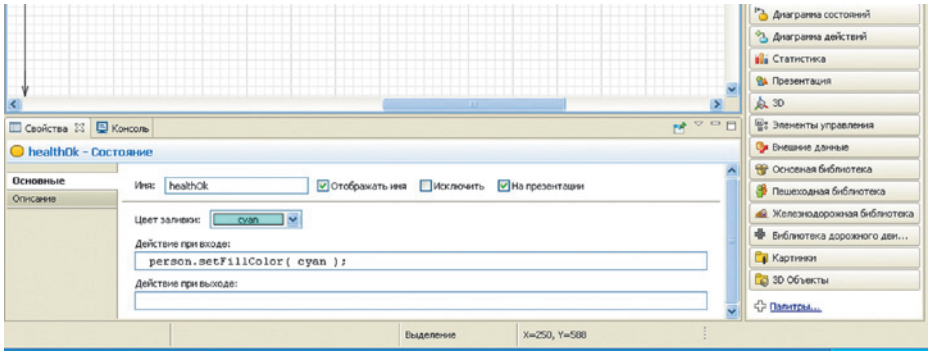
Чтобы изучать влияние параметров на модель, из палитры “Основная” перетащим два объекта-параметра:

- incubation (инкубационный период), зададим ему значение целого типа 5;
- illTime (время болезни), зададим ему значение 10.

Блоки свяжем между собой соединителем-стрелкой из палитры “Диаграмма состояний”. Получится примерно следующее:



Стоит заметить, что цвет блоков на состояние знака-человечка в рабочей модели не повлияет. Чтобы следить за происходящим, мы добавим на каждом состоянии команду, меняющую цвет в анимации:

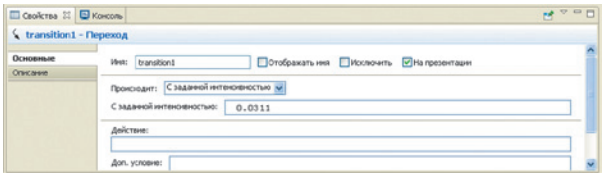


[2] К сожалению, в реальном мире даже для эпидемии гриппа это не так.

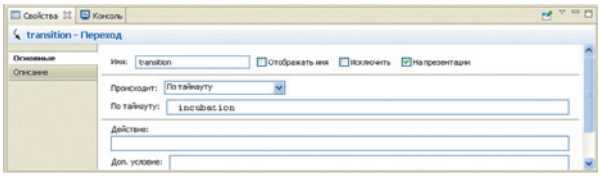
Напишем для каждого блока действие при входе в него. Для первого блока: person.setFillColor(cyan);

Аналогично — для остальных блоков.

Смена состояния будет происходить по стрелкам-связям. Стрелка от “Здоров” к “Заражен” — первичное заболевание будет происходить из-за заражения случайно (без прямого контакта), то есть просто с некоторой заданной интенсивностью. Зададим для стрелки частоту “0.0311”:



Второй переход — от состояния “Заражен” к состоянию “Заболел” будет определяться уже только временем инкубационного периода. Выберем для стрелки параметр “По тайм-ауту”, но вместо конкретного значения сошлемся на имя параметра:



Также поступим и для последнего перехода.

Обратите внимание: пока наша модель никак не зависит от количества контактов и близости людей друг к другу, единственный способ “заразиться” — это попасть под распределение.

Вернемся к общему экрану (закладка Main).

Чрезвычайно неудобно следить за распространением эпидемии “вручную”. Воспользуемся для этих наблюдений средствами автоматического сбора статистики.

Во-первых, включим сбор статистики у агентов. Выберем объект people, переключимся на закладку “Статистика”.

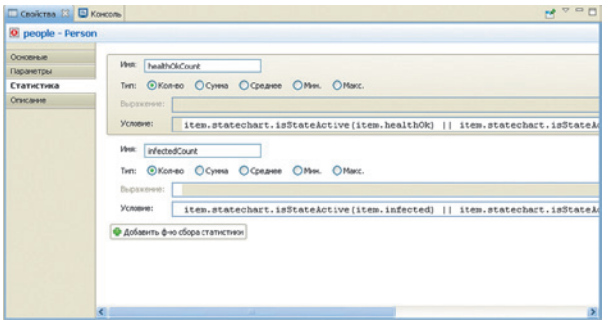
Изначально там ничего нет, но у нас есть возможность добавить счетчики. Сначала подсчитаем количество здоровых людей:

- Добавим функцию сбора статистики.
- Назовем ее healthOkCount.
- Подсчитывать будем количество.
- Условие подсчета будет состоять из двух: человек либо находится в состоянии “здоров”, либо “выздоровел”. Вот как выглядит условие “Здоров”: item.statechart.isStateActive(item.healthOk). Соединим его с условием “выздоровел” с помощью связки “или”. Названия статусов мы задали сами.

Получим примерно вот что:

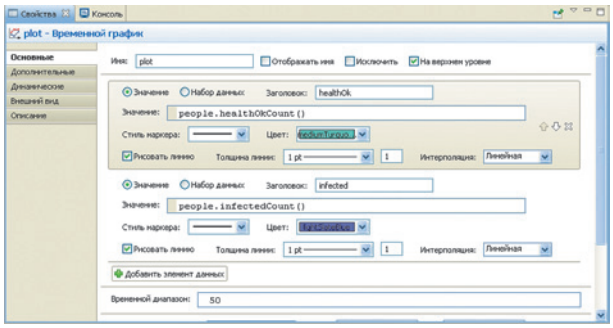
item.statechart.isStateActive(item.healthOk) ||
item.statechart.isStateActive(item.cured)

- Аналогично добавим счетчик infectedCount — с условием “заражен” или “болен”.

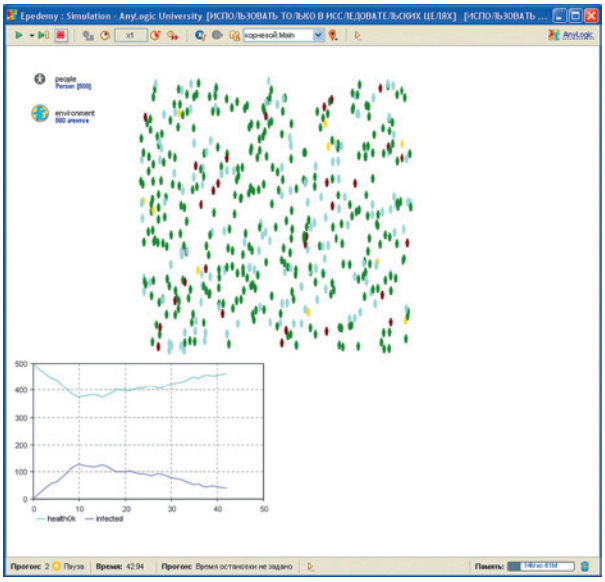


Теперь разместим средство просмотра: временной график.

- Перетащим необходимый объект из палитры “Статистика” на рабочее пространство.
- На закладке “Основные” этого объекта добавим элемент данных (нажмем кнопку).
- Элемент назовем healthOk, получать будем значения из нашего объекта people с помощью заданной нами функции сбора статистики — people.healthOkCount().
- Точно так же добавим второй элемент данных — infected.



Запустим модель и увидим примерно такую картину:

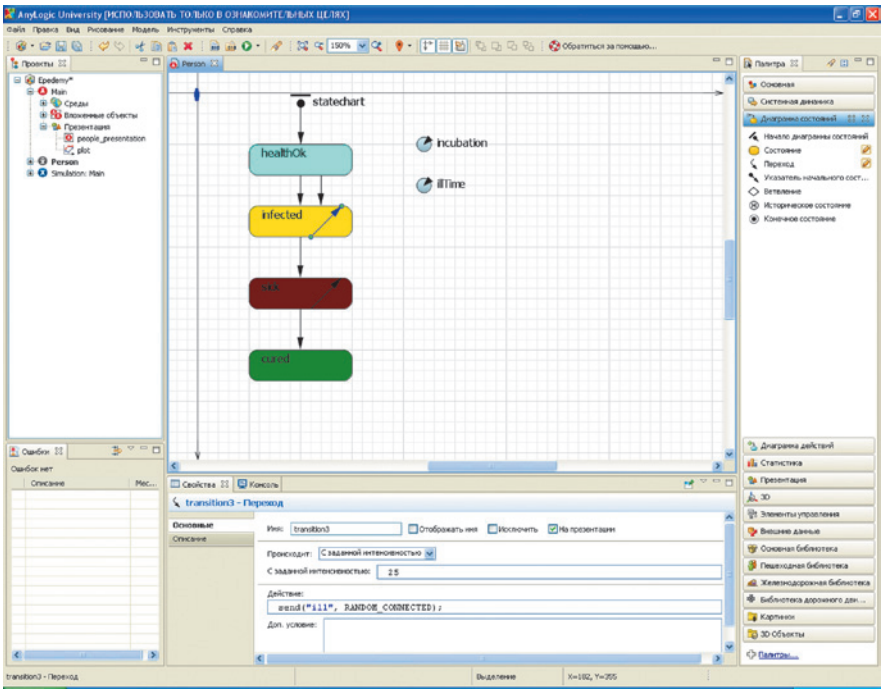


Как видим, максимум заболевших — на 10-й день (удивительно, но это — время инкубационного периода и периода явной болезни, после которого люди начинают выздоравливать). Пока что мы фактически моделируем незаразную болезнь — по-

скольку заболеваемость от собственно заболевших или зараженных не зависит никак.

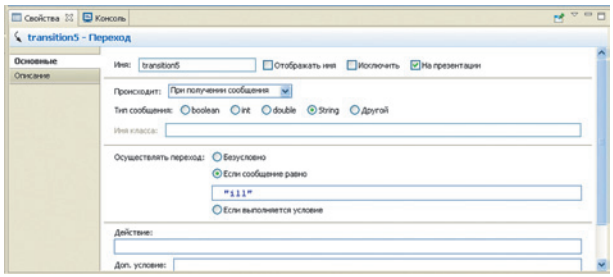
Доработаем нашу модель, учтя распространение инфекции от заболевших. Сделаем мы это с помощью механизма рассылки сообщений.

В диаграмме мы добавим два внутренних (внутри блоков) перехода — внутри состояний `infected` и `sick`. У этих переходов мы с заданной интенсивностью будем рассылать сообщения “ill”.

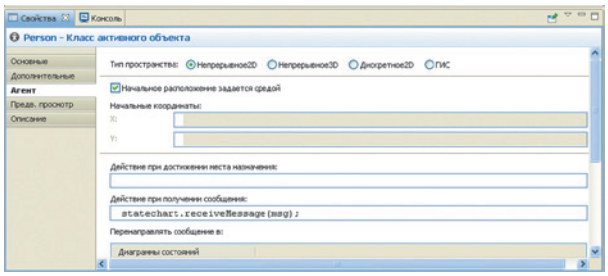


Рассылка сообщений будет случайной, поскольку мы не знаем, с кем контактирует человек во время болезни.

Для отработки сообщения мы добавим второй переход от состояния “Здоров” к состоянию “Заражен”, который будет происходить при получении сообщения “ill”:



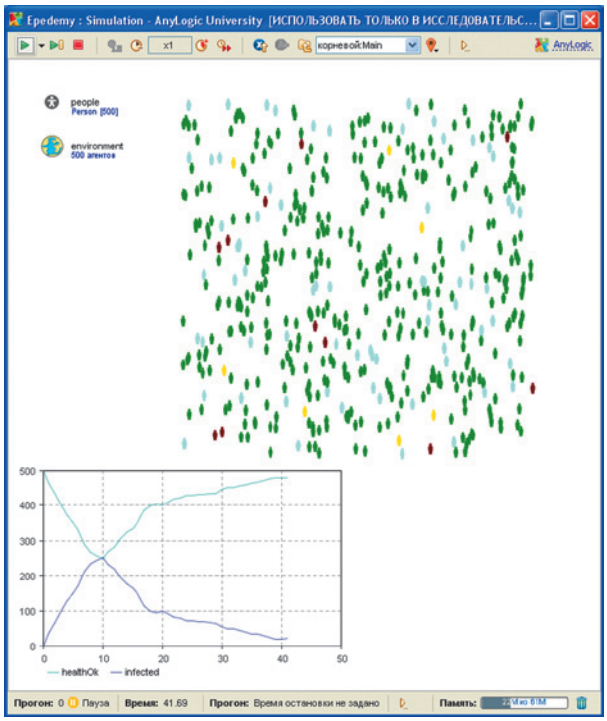
Сообщение будет приходить случайным людям, находящимся на расстоянии 10 единиц (именно это мы задали при создании модели). Чтобы сообщения приходили агентам, зададим свойства класса-агента: выберем в дереве класс person и зададим на странице агент действие при получении сообщения:



Действие statechart.receiveMessage(msg); передает сообщение в диаграмму смены состояний.

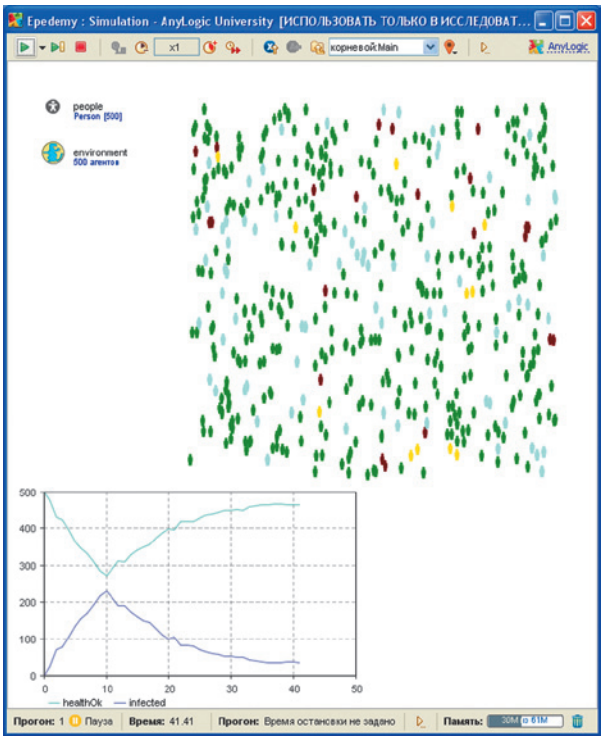
Для начала предположим, что больные люди передвигаются по пространству точно так же, как здоровые и зараженные, — то есть поддерживают обычную частоту контактов (25 сообщений в единицу времени).

Выполним прогон и увидим совсем другую картину:



К моменту начала снижения количества заболевших (10 дней) их количество становится равно количеству здоровых — 250 человек. В прошлый раз оно не успело подняться и до 200.

Введем ограничение: пусть заболевшие сидят в карантине, то есть посылают только три сообщения “болеть”.



Как видим, улучшение есть, но не принципиальное[3]. Если сделать несколько прогонов, то будет видно, что в обоих вариантах ситуация может быть и хуже, и лучше — в зависимости от распределения людей.

Можно предположить, что существенная разница возникнет в том случае, когда будет несколько “ареалов обитания” людей, — тогда рост очага заболевания будет гораздо ниже. В пределах же уже зараженного поселения статистическая разница будет невелика.

Сделаем нашу модель более подходящей к картине распространения серьезной инфекции:

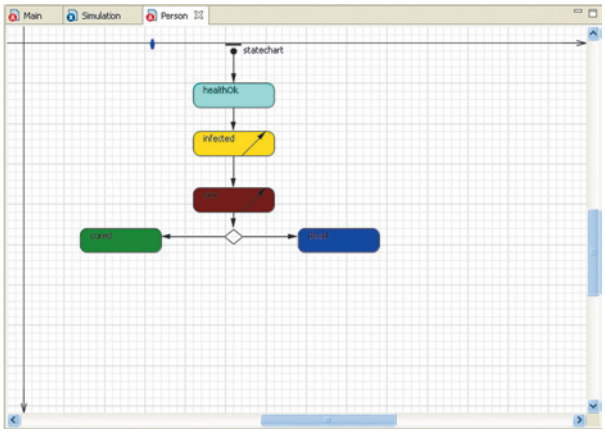
1. Источником заражения так или иначе всегда является один человек-— носитель. Уберем “фоновое” заражение.

2. Если лечения нет, то в распространении эпидемии возникает понятие “смертность”.

Чтобы исследовать поведение заболевания в нашей модели, без необходимости каждый раз менять параметры вручную, мы вынесем эти параметры на стартовую страницу и будем их менять.

Первые изменения коснутся главной страницы (Main). Перенесем на нее параметры incubation и illTime и добавим новые параметры: contactNorm (со значением 25) и sickNorm (со значением 3) — количество контактов для “свободного” и “заболевшего” людей, а также параметр distance — дистанция, на которой болезнь будет действовать.

Теперь внесем изменения в диаграмму смены состояний. Во-первых, изменим структуру:



1. Уберем первый переход от состояния “Здоров” к состоянию “Заражен”.

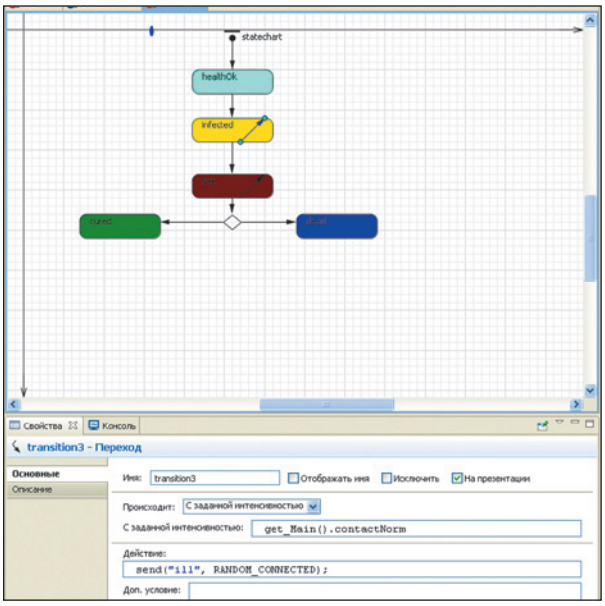
2. Добавим после состояния “Болен” элемент ветвления и от него два исхода: “Выздоровел” и “Умер”.

Условия перехода описываются в параметрах стрелок, выходящих из разветвления. Условие перехода к состоянию “Здоров” будет таким: uniform() < 0.98.

То есть если случайное число, равновероятно попадающее в промежуток от 0 до 1, должно оказаться меньше 0,98, — в 98 случаях из 100 человек выздоравливает.

Второе условие мы не будем задавать — под него попадут все остальные.

3. Там, где мы задавали значения частот и таймаутов, в соответствующих полях переходов мы должны будем сослаться на значения параметров из основного объекта. Делается это примерно так: get_Main().contactNorm, — так ссылаются из подчиненного объекта на параметр в основном.



[3] Авторам кажется, что это неплохо согласуется с историей распространения, например, чумных эпидемий. И “успешностью”, например, английских запираний.

Раньше заболевание начиналось с заразившихся “из воздуха”, теперь у нас таких не осталось. Чтобы эпидемия стартовала, в свойствах главного объекта (Main) мы укажем действие — посылку сообщения “ill” первому по списку агенту. Он и будет нашей стартовой точкой. Для этого запишем в поле “Действие при запуске” следующее:

people.get(1).statechart.receiveMessage(“ill”);

И, наконец, нам нужно будет дать возможность задавать значения параметров на стартовой странице.

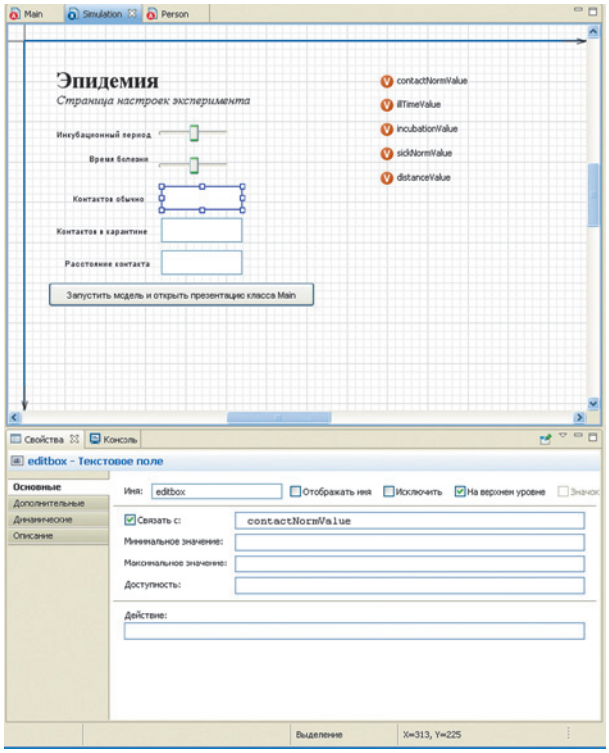
Для этого откроем в графическом редакторе страницу симуляции (simulation) и добавим на нее несколько элементов:

1. Добавим переменные (не параметры, а именно переменные!) из палитры основные. Назовем их так же, как параметры, только с припиской value.

2. Добавим элементы ввода. Сами элементы мы добавим из палитры “Элементы управления”, а подписи к ним — из палитры “Презентация”. Текст подписей наберем как значение свойства Текст.

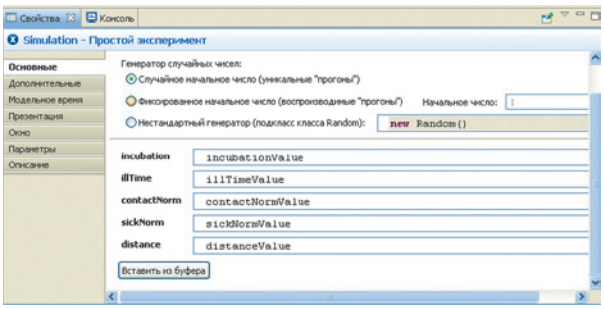
3. Свяжем элементы управления с переменными. Для этого поставим в их свойствах галочку “Связать с” и напишем имя соответствующих переменных.

Должно получиться примерно вот что:



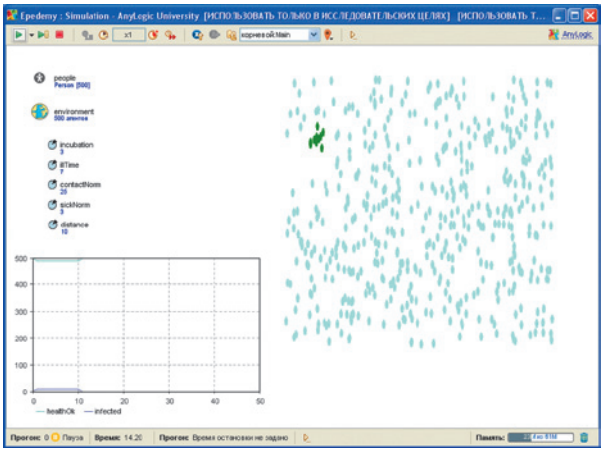
И последнее, мы свяжем параметры с переменными.

4. Выберем в дереве объектов Simulation и на странице Основные зададим в строках появившихся параметров значения переменных:



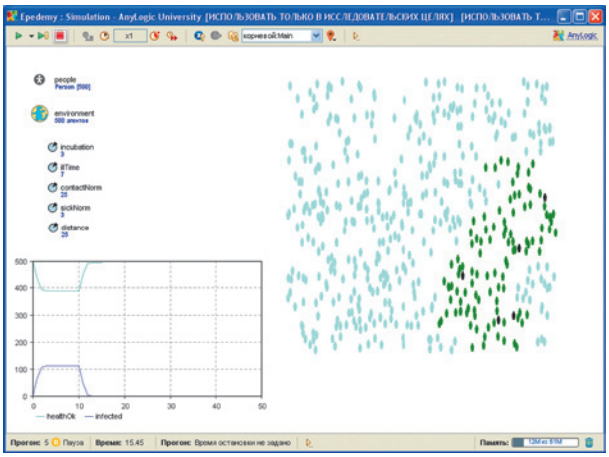
Элементы для сбора и отображения данных о смертности можете сделать сами, как было описано выше.

Теперь выполним несколько прогонов с разными значениями параметров. С параметрами, изначально заданными нами, картина скорее всего будет примерно такой:



Как видим, картина довольно резко изменилась. Болезнь проявилась только на небольшой группе людей. Причина очень проста — остальные не оказались в “радиусе заражения”.

Если же радиус увеличить до 25, картина будет примерно такой:



То есть количество заболевших растет. При этом значения количества контактов, сроков заболевания и т.д. на максимальное количество пораженных будет влиять мало. Основным фактором, как мы и предсказывали, оказывается “площадь контактов”, то есть фактически — террито-

рия плотного расселения. Таким образом, ввод карантина — мера целесообразная, но только для целых районов (то есть областей, не связанных с внешним миром). Внутри пораженного района нам не удастся уменьшить количество заболевших, поскольку для большинства болезней у нас есть период, в котором болезнь еще не заметна, но уже заразна.

Задания

Самостоятельно усовершенствуйте модель, чтобы изучить влияние таких параметров:

1. Влияние плотности расселения (например, в результате увеличения площади).

2. Появление лечения — как варианта, при котором время болезни и смертность сокращаются.

3. Как будет развиваться ситуация, если приобретенный иммунитет окажется временным?

4. Предположим, что мы провели вакцинацию — то есть примерно половина людей не может заболеть (на какой-то промежуток времени). Как это повлияет на распространение болезни во время конкретной эпидемии? В долгосрочной перспективе?

Системно-динамическое моделирование

При организации работы компании сотовой связи в крупном городе один из существенных вопросов — сроки и условия наращивания мощности базовых станций сети. От этого показателя напрямую зависит качество обслуживания клиентов.

Если компания не предпринимает никаких особых усилий по привлечению клиентов, то их приток зависит от количества тех, кто уже обслуживается компанией, — сюда войдут и стандартные затраты на рекламу (фиксированный процент выручки с клиента), и те, кто придет по рекомендации.

Модернизировать станции слишком часто нельзя, это приведет к большим расходам, а оборудование не успеет окупиться.

Если мощность недостаточна, появляются недовольные клиенты. Недовольные компанией клиенты отказываются от ее услуг. Некоторое количество недовольных есть всегда, но если качество работы падает — их становится больше.

Выяснив, что мощность недостаточна, мы начинаем модернизацию — но она тоже потребует времени, а результат далеко не сразу окажет влияние на мнение (и динамику) клиентов.

Основным критерием успешности работы для нас будет отсутствие падения общего количества клиентов, то есть такая организация дела, при которой приток клиентов больше, чем потеря. Основная единица времени для нас в этих подсчетах — неделя.

Вопрос: какое время задержки допустимо? Насколько нужно повышать мощность оборудования?

Структура сети, поведение конкретных клиентов, затраты на рекламу — не могут оказывать принципиального влияния на ситуацию, поскольку *никак не влияют на взаимозависимости*. Нас фактически интересуют статистические параметры. Даже модернизация никогда не даст скачкообразного эффекта — потому что никогда не будет выполняться разом во всей сети.

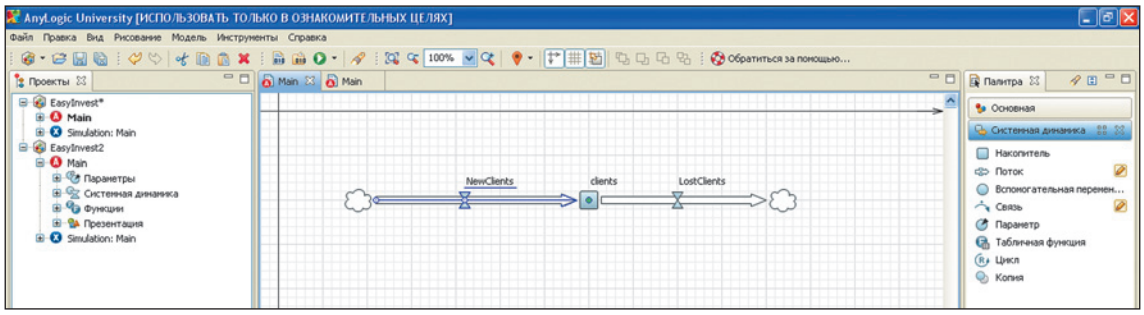
Как уже рассказывалось в учебнике [5], для изучения влияния параметров на сложный процесс в крупных системах, как правило, нет смысла (а то и возможности) моделировать точное поведение каждого участника процесса.

Один из наиболее мощных инструментов для решения задач такого рода — метод, получивший название “Системно-динамическое моделирование” ([2], [3]). Его мы и применим.

Создадим новую чистую модель — не используя шаблон. Для создания модели воспользуемся палитрой “Системная динамика”. Основой нашей модели будет поток клиентов, который мы будем накапливать в компании.

- Поместим на рабочее поле модели объект “Накопитель” и назовем этот объект Clients. Изначально (по умолчанию) у нас будет 100 условных клиентов.
- Разместим два объекта “Поток”: приток клиентов и их потерю. Приток — NewClients, отток — LostClients.
- Конечную точку притока и конечную точку оттока поместим в накопитель — так, чтобы соединитель принял вид, как на иллюстрации.

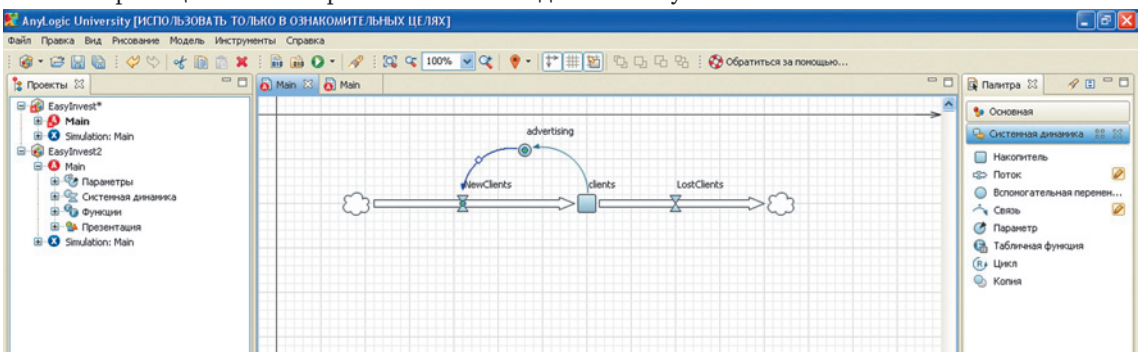
Обратите внимание — фактически мы предполагаем, что у нас есть бесконечный источник клиентов. В реальных условиях это, конечно, несколько не так.



На приток влияет реклама. Мы не изучаем никаких ее параметров, поэтому сведем все к одной вспомогательной переменной.

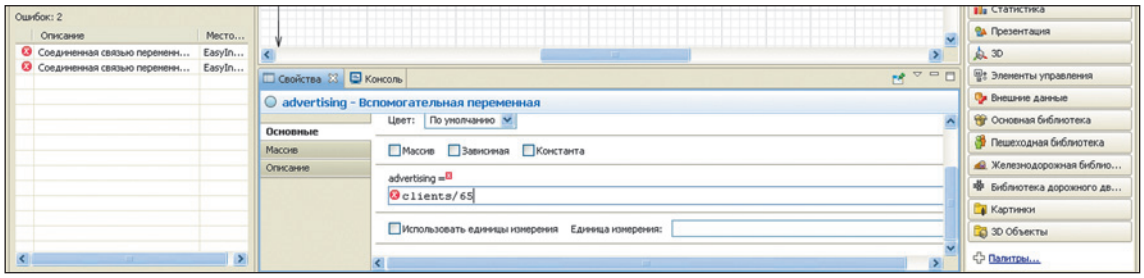
• Поместим на рабочее поле переменную, назовем ее advertising.

• Эту переменную мы свяжем с клиентской базой и притоком новых клиентов с помощью связей — которые тоже перетащим из палитры. Имена связям давать не нужно.

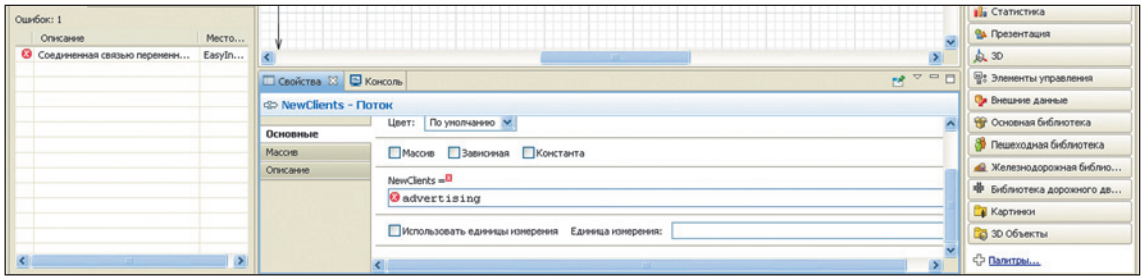


При этом возникнут два предупреждения: о том, что связь есть, но формулы для ее учета не заданы.

• Зададим значение этой переменной как формулу: clients/65



• Таким же способом зададим значение притока новых клиентов:



Сообщения об ошибках исчезнут.

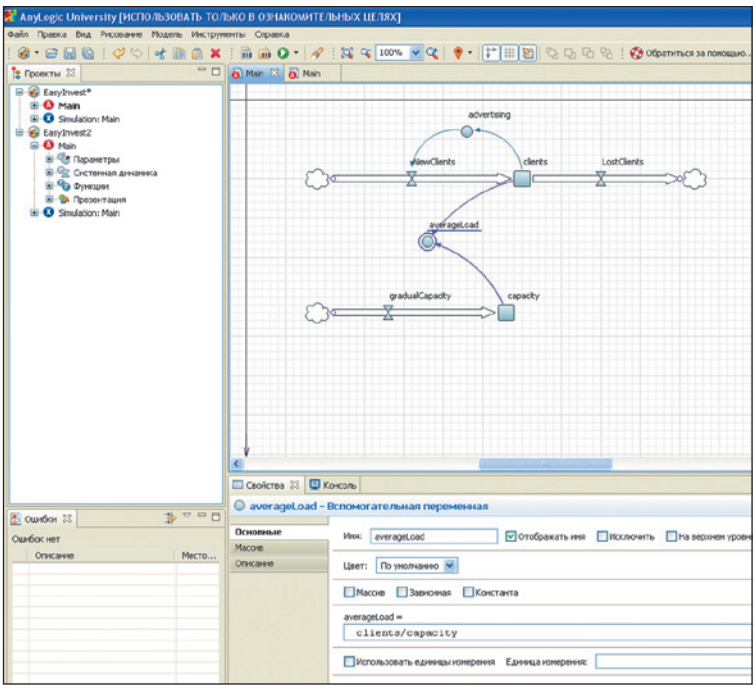
Теперь перейдем к самому наращиванию мощности. Создадим вторую линию:

• Зададим имеющуюся мощность накопителем, который назовем capacity — вместимость. Изначальная мощность сети — 60 условных клиентов.

• Мощность будет наращиваться потоком gradualCapacity — из бесконечного источника. То есть мощность мы наращиваем в зависимости только от наших решений.

Имеющаяся мощность по обработке обращений пользователей будет влиять на среднюю загрузку системы, которую мы вычислим как отношение мощности к количеству клиентов (clients/capacity).

• Добавим промежуточную переменную, свяжем ее с имеющимися параметрами и зададим формулу.



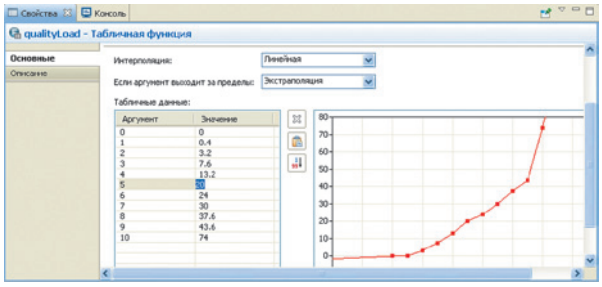
Теперь опишем соотношение качества с нагрузкой.

Качество обслуживания для клиентов — это нагрузка, но не текущая, а ранее работавшая, создавшая впечатление у клиентов.

При этом нам надо будет учесть, что зависимость эта не линейная и заданная некоторыми статистическими наблюдениями.

Для этого мы создадим специальную функцию, в которой зададим зависимость не формулой, а набором значений:

• Перетащим для этого из палитры компонент “Табличная функция”, назовем ее qualityLoad и зададим следующие параметры:



• Создадим еще одну промежуточную переменную — quality, и формула для этой переменной будет такой: qualityLoad(delay(averageLoad,10)).

То есть оценка качества в виде количества “уходящих” клиентов будет даваться с задержкой на 10 единиц времени. Переменную мы свяжем с переменной средней загрузки и потоком уходящих клиентов. В “вентиле” уходящих клиентов мы просто напишем ее название.

• Для оценки и принятия решения о модернизации мы введем параметр standart — задающий нагрузку, при которой необходимо принять решение о модернизации. По умолчанию примем 1,7 — то есть примерно тогда, когда у нас за каждый цикл начнут уходить два клиента.

• Параметр мы свяжем с текущей нагрузкой через промежуточную переменную defectCapacity (недостаток мощности), вычислив ее как разницу между загрузкой и стандартом.

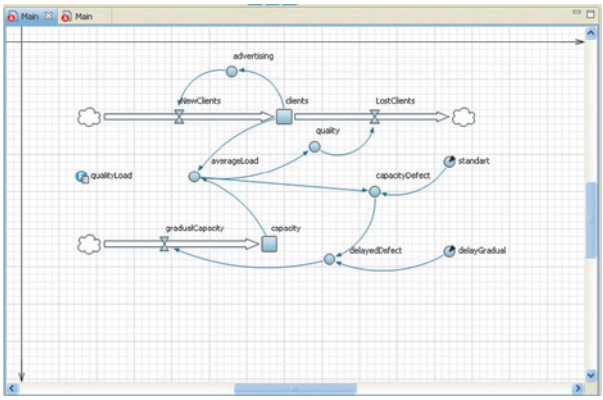
Влиять на увеличение пропускной способности она будет тоже с задержкой:

• Создадим параметр delayGradual. По умолчанию примем ее за 30 целых единиц.

• Создадим промежуточную переменную для роста пропускной способности с задержкой delayedDefect, определим ее формулой: delay(capacityDefect,delayGradual).

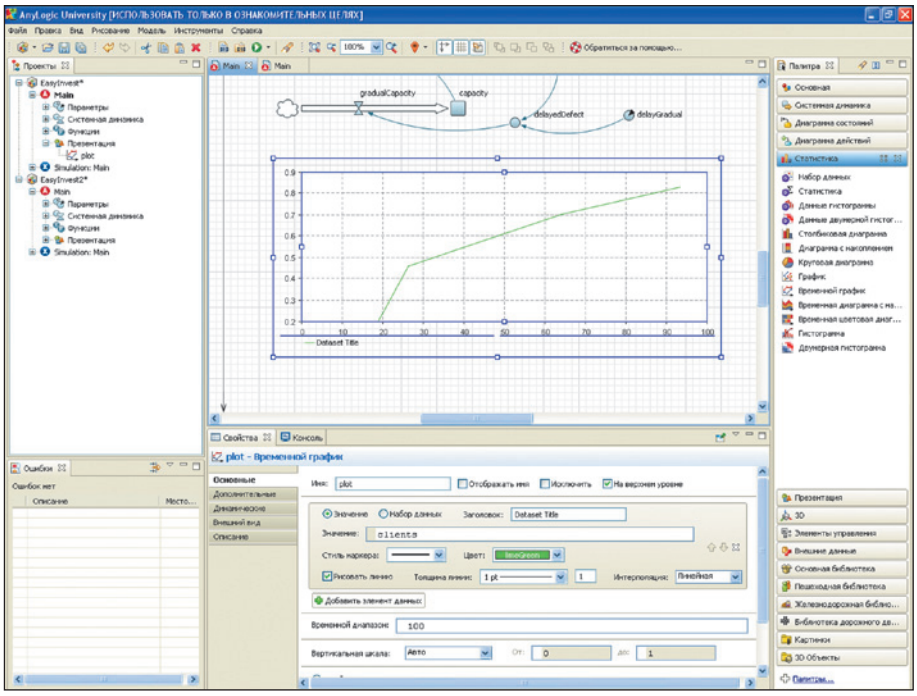
• Свяжем ее с текущим дефектом и потоком модернизации.

Итоговая для нас схема:

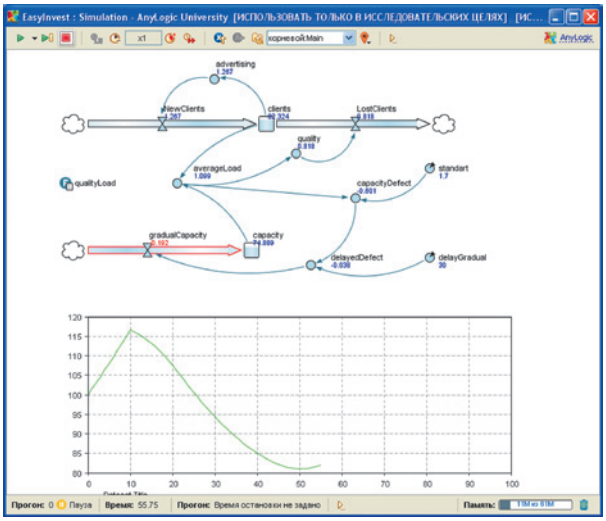


• На переменную delayDefect мы сошлемся в “вентиле” потока gradualCapacity — формулой delayedDefect*5, — поскольку, очевидно, будем наращивать мощность с запасом..

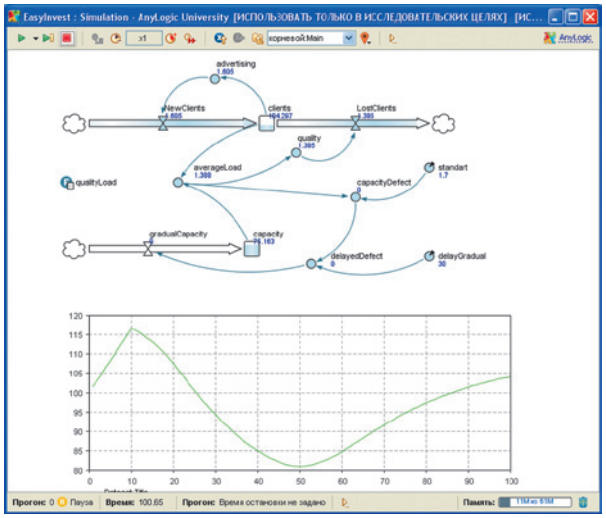
Чтобы отслеживать результаты работы, мы добавим на лист временной график, на котором отразим изменение количества клиентов со временем. Для этого достаточно в графике добавить одну серию данных, связанную с переменной clients.



Как ведет себя система при наших значениях по умолчанию? Запустим ее и обнаружим неприятную деталь:



Как только у нас начинает расти пропускная способность, вентиль “увеличения” начинает ее уменьшать. Очевидно, портить нашу сеть мы не будем. Изменим формулу вычисления дефекта на следующую: max(averageLoad-standart,0). То есть не расти она может, но уменьшаться не должна. Повторим прогон.



Изначально все неплохо. За два месяца количество клиентов возрастает примерно на 17%. Но потом происходит следующее: через десять недель клиенты обнаруживают, что сети на них явно не хватает, и начинают “голосовать ногами”. Несмотря на то что через 30 недель мы начинаем модернизацию, ситуация не улучшается до 50 недель. К этому моменту мы теряем 17% к изначальному количеству, и ситуация исправляется только к 85-й неделе, то есть более чем через год.

Очевидно, что с таким подходом к своей сети компания вряд ли обеспечит прибыль.

Задания

1. Изучите влияние имеющихся параметров на систему, добейтесь улучшения ситуации.

2. Почему мы не предлагаем варьировать в модели “рекламную мощность”, время задержки реакции клиентов?

3. Какой еще параметр можно добавить для управления системой — не меняя ее основной структуры?

Поиск оптимального решения

Любой магазин, любая организация, занимающаяся работой с потоком клиентов — будь то банковский офис или отделение единого информационно-расчетного центра, — рано или поздно сталкиваются с проблемой расчета необходимого количества сотрудников (кассиров, операционистов). Число работников при этом, с одной стороны, не должно быть очень маленьким, чтобы не собирать огромные очереди недовольных клиентов, но, с другой стороны, не должно быть и слишком большим, иначе организация понесет лишние расходы на зарплату. Расчет оптимального количества сотрудников, которое будет зависеть от интенсивности потока клиентов, среднего времени обслуживания и других параметров, является примером решения **задачи оптимизации**.

Задачу эту можно решить, подготовив и решив систему уравнений, однако учет всех параметров и связей даже такой простой системы, как описанная выше, может потребовать введения в модель большого количества переменных, что неизбежно приведет в итоге к достаточно громоздким вычислениям.

Куда более рациональным будет в этом случае прибегнуть к средствам компьютерного моделирования, позволяющим вместо сложных и не всегда возможных аналитических преобразований свести все к рутинным расчетам, то есть опереться на вычислительные мощности компьютера.

Пользователю в этом случае остается лишь задать объекты, параметры и связи, входящие в моделируемую систему, а также правильно интерпретировать полученные на выходе результаты. Рассмотрим пример решения подобной задачи в среде Anylogic.

1. Создание дискретно-событийной модели работы касс магазина

Для создания модели магазина самообслуживания нам понадобятся следующие элементы из блока “Основная библиотека”: *source* (источник заявок, т.е. посетителей магазина), delay (задержка заявок, связанная с тем, что посетителю требуется какое-то время на выбор покупок), service (обслуживание на кассовой линии), resourcePool (элемент, отвечающий за количество кассиров) и sink (уничтожение заявок).

Последовательно перетащим элементы из блока “Палитра” в рабочее окно модели и соединим их линиями так, как показано на *рис.* 1.

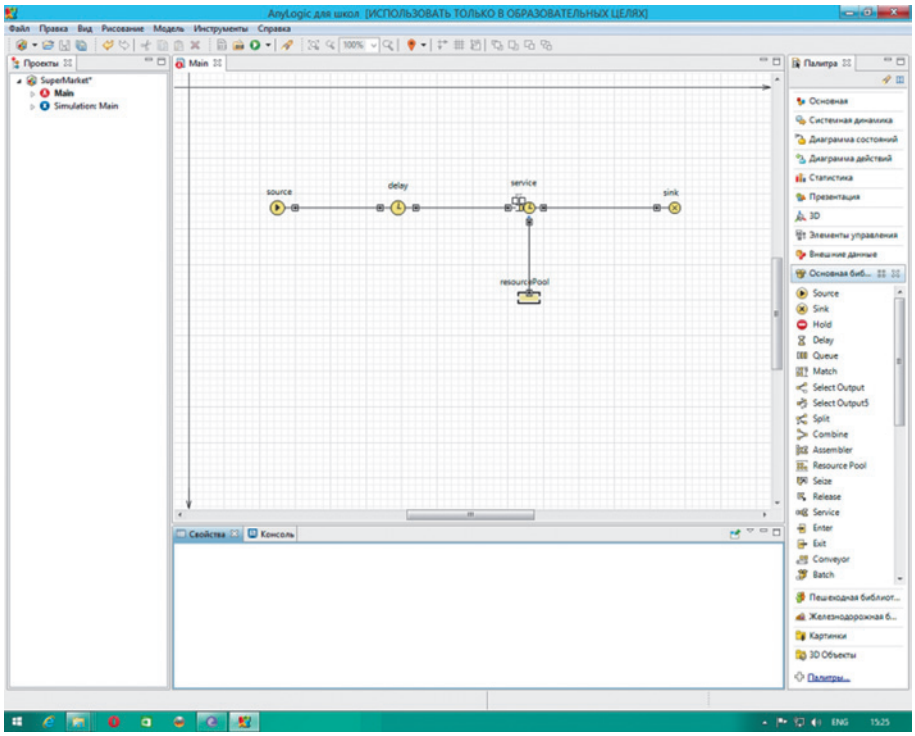


Рис. 1

Следующим этапом создания модели будет изменение свойств каждого из добавленных элементов.

1. Элемент source (*рис.* 2).

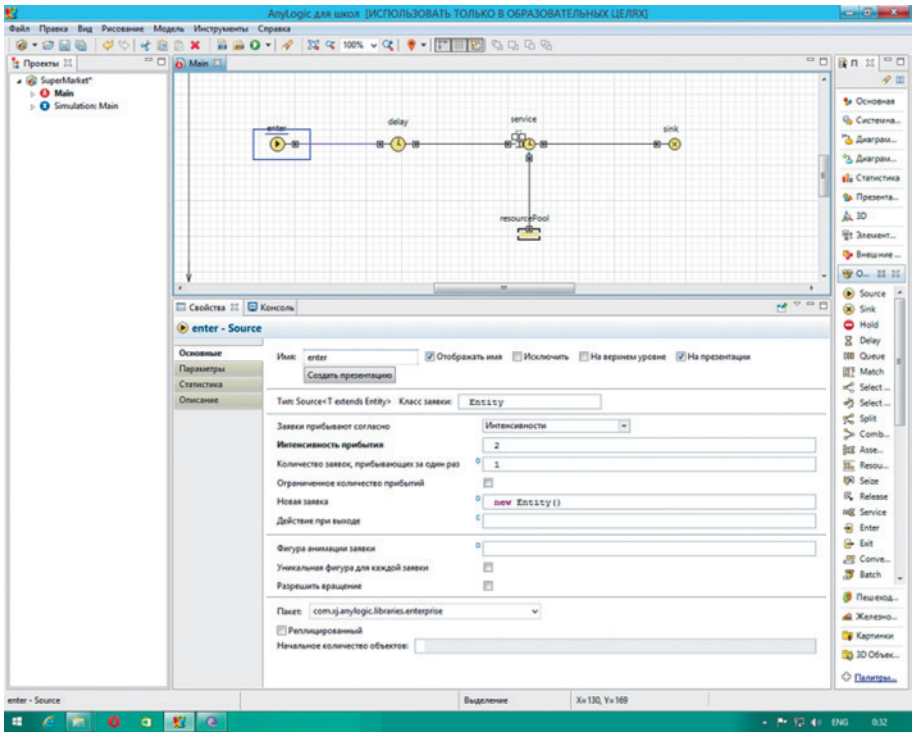


Рис. 2

• Так как данный элемент отвечает за появление заявок (покупателей в магазине), переименуем его в enter (вход).

• Очевидно, что в разное время суток покупатели заходят с разной интенсивностью, но пока для простоты будем считать, что частота появления в магазине новых клиентов все время одинакова и равна двум человекам в одну минуту.

2. Элемент delay (*рис.* 3).

• Переименуем его в shopping, так как этот элемент отвечает за время, требующееся зашедшему в магазин человеку на выбор покупок.

• Для того чтобы выставить это время, используем функцию треугольного распределения triangular. Будем считать, что минимум на процедуру выбора потребуется минимум пять минут, максимум — один час, а наиболее вероятным укажем время в двадцать минут.

• Поле *максимальная вместимость* отметим галочкой, так как в условиях нашей модели будем считать магазин способным вместить сколько угодно покупателей.

3. Элемент service.

• Переименуем данный элемент в queue (очередь).

• В поле *количество ресурсов* оставим значение, равное единице, так как на обслуживание одного покупателя обычно требуется лишь один кассир (один ресурс).

• *Время задержки*, т.е. время, необходимое посетителю для оплаты покупок в кассе, укажем также при помощи треугольного распределения triangular(0.5, 1, 5).

• Поле *максимальная вместимость очереди* отметим галочкой, так как в процессе последующего эксперимента при определенных параметрах очередь может оказаться очень большой.

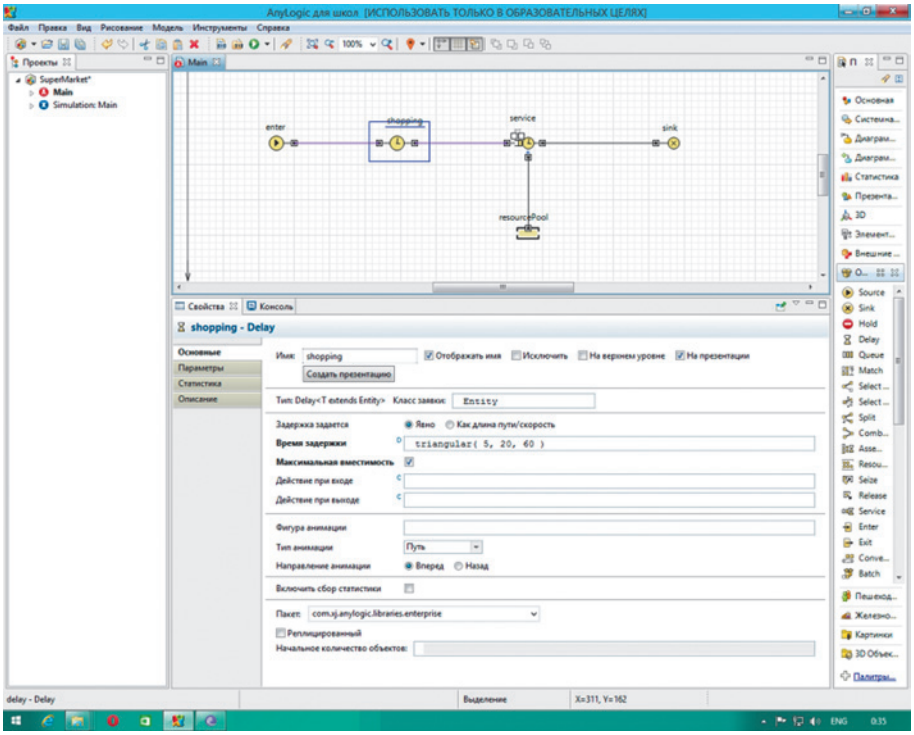


Рис. 3

4. Элемент resourcePool.

• Переименуем в cashiers (кассиры).

• *Количество ресурсов* (в нашем случае количество кассиров) укажем равным четыре. Впоследствии это число мы изменим на параметр, который будет изменяться в зависимости от интенсивности потока покупателей.

5. Элемент sink просто переименуем в exit (выход).

Запустим получившуюся модель. Зеленые цифры над элементами *shopping* и *queue* показывают, сколько человек в данный момент выбирает товар, а сколько уже стоит в очередях к кассам. Однако данное отображение не является для нас особенно информативным. Гораздо важнее для нас будет не текущий размер очередей, а их усредненное по времени значение. Для того чтобы собрать такую информацию, необходимо добавить в рабочее окно программы элемент *Статистика,* переименовав его в queueSize. В поле *значение* данного элемента пропишем формулу *queue*.queueSize()/4, где цифра четыре означает количество кассиров, так как необходимо одну большую очередь к кассам разделить на четыре равных по величине очереди. В свойствах queue разрешим сбор статистики, поставив соответствующую галочку.

Теперь мы можем наглядно отобразить средний размер очереди в кассу при помощи столбиковой диаграммы. Перетащим соответствующий элемент в рабочую область. В свойствах щелкнем на *добавить элемент данных* и зададим в поле *значение* формулу queueSize.mean(). Масштаб диаграммы выберете на свое усмотрение.

Запустив снова симуляцию модели, увидим на диаграмме, что средняя очередь постепенно растет и за 720 минут (12 часов) достигает величины порядка восемь человек (*рис.* 4). Если же изменить интенсивность прибытия покупателей с два человек в минуту на один, то увидим, что размер очереди с течением времени не воз-

растает и по величине близок к нулю. Очевидно, что для такой интенсивности потока клиентов четыре кассира — это слишком много.

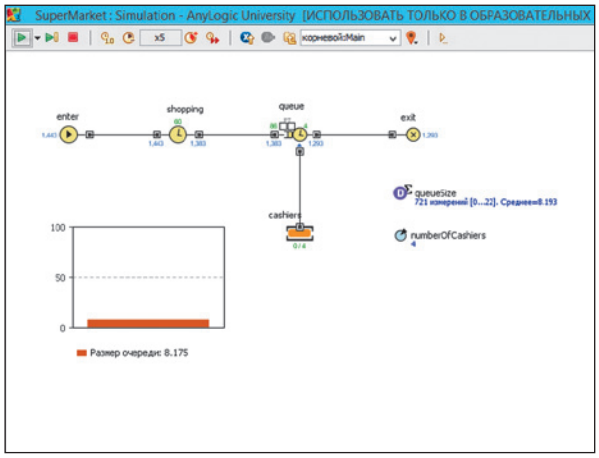


Рис. 4

Дополнительные задания к части 1

1. Построенная модель не ограничена по времени, в то время как большинство магазинов работают не круглосуточно, а с утра и до вечера. Попробуйте ограничить время работы модели одним днем с 10 утра до 10 вечера. Делается это во вкладке *Модельное время* в свойствах объекта Simulation.

2. В нашей модели интенсивность прибытия покупателей в магазин есть величина постоянная. Но в реальной жизни интенсивность зависит от времени суток (утром покупателей очень мало, а вечером, наоборот, много) и дня недели (будни или выходные). Попробуйте задать интенсивность, зависящую от расписания. Для этого перетащите в рабочее окно элемент *Расписание* вкладки *Основная,* задайте в его свойствах нужное расписание. Затем в свойствах объекта *Enter* укажите, что заявки должны прибывать согласно *интенсивности (по расписанию)*, и сошлитесь на объект *Расписание.*

2. Эксперимент по оптимизации.

Само собой, ставить четырех кассиров в магазин, где спокойно управляется с работой и меньшее число человек, никто не будет, так как это приведет к излишним затратам на заработную плату. Однако чрезмерная экономия на количестве работников повлечет за собой образование больших очередей, что может негативно отразиться на репутации магазина. Разумнее всего в этом случае найти оптимальное решение, т.е. минимальное количество кассиров, при котором средняя очередь не будет превышать заданное значение. Используем для этого *Оптимизационный эксперимент*, встроенный в Anylogic.

Наша цель — проведя большое количество прогонов модели, определить статистически оптимальное значение этого показателя. На основании полученных при этом данных специальный алгоритм OptQuest, один из лучших на сегодняшний день, определит оптимальное для этой модели решение. Алгоритм этот сложен, в нем используются и методы математической оптимизации, и нейронные сети, и эвристики поиска решения. Алгоритм запатентован, и его содержание правообладатель не публикует.

Конечно, сами прогоны, в отличие от наших экспериментов, показываться не будут — нас будет интересовать только результат, полученный в каждом из них.

Для начала добавим в рабочее окно модели параметр, перетащив его из палитры *Основная.* Переименуем его в numberOfCashiers. Зададим целочисленный тип (integer) и значение по умолчанию, равное *4.* Этот параметр в дальнейшем мы и будем оптимизировать. В свойствах объекта *cashiers* в поле *количество ресурсов* заменим число четыре на numberOfCashiers.

Можно приступать к созданию нового эксперимента. Для этого необходимо щелкнуть правой кнопкой в левом окне на названии модели и выбрать *Создать — Эксперимент — Оптимизация.*

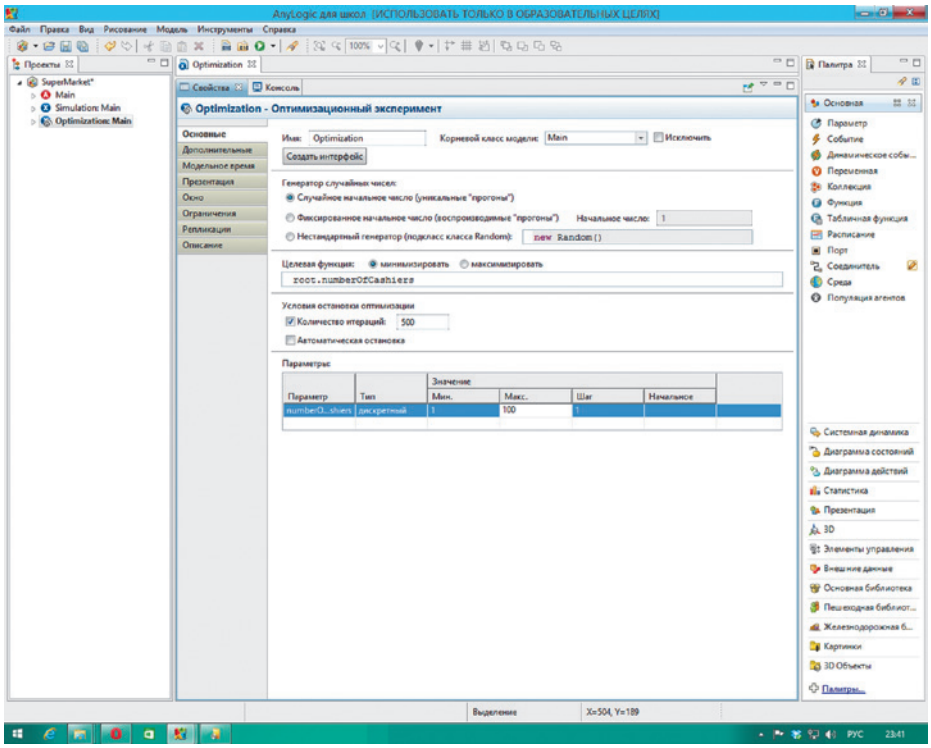


Рис. 5

Для проведения эксперимента необходимо указать целевую функцию, которую мы хотим минимизировать или максимизировать. В нашем случае это будет минимизация добавленного только что параметра numberOfCashiers. Поэтому на вкладке *Основные* свойств эксперимента укажем, что в качестве целевой функции мы хотим видеть параметр *root*.numberOfCashiers. Указатель root необходим, так как параметр расположен не в окне оптимизационного эксперимента, а в активном классе модели. Чуть ниже зададим дискретное изменение данного параметра от 1 до 100 с шагом 1 (*рис.* 5). Это означает, что эксперимент будет выполняться снова и снова, но количество кассиров будет меняться от одного до двадцати, и для каждого значения параметра numberOfCashiers будет измеряться среднее значение очереди.

Суть нашего эксперимента заключается в том, что необходимо найти то минимально допустимое количество работников, при котором средняя очередь в кассу не превысит заданного нами значения. Величина, которой мы ограничиваем очередь, в разных магазинах может быть очень и очень разной и определяется многими факторами, как, например, ценовой категорией магазина (вряд ли клиент дорогого магазина захочет стоять в длинной очереди) или наличием свободного пространства (при его отсутствии длинные очереди могут блокировать проход покупателей, еще не выбравших товар). Так или иначе, но перед запуском оптимизационного эксперимента мы должны точно знать, очереди какой длины являются для нашего магазина предельно допустимыми. Для определенности предположим, что число это будет равно пяти. Перейдем во вкладку *Ограничения* и потребуем, чтобы значение root.queueSize.mean() в процессе эксперимента не превышало пяти (*рис.* 6).

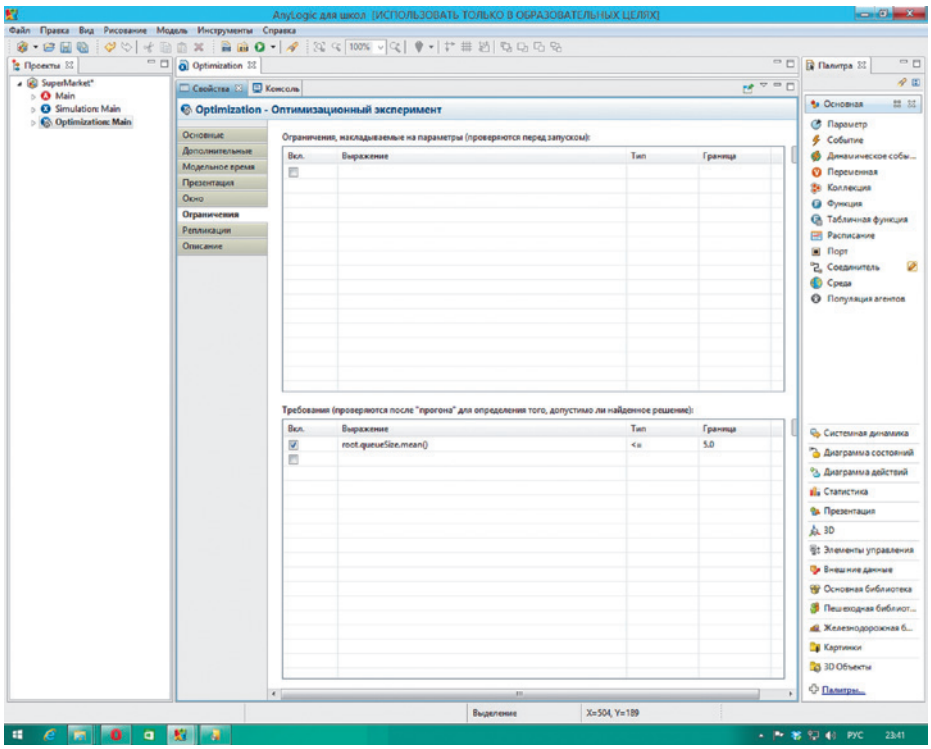


Рис. 6

Большая часть магазинов работает с 10 утра до 10 вечера. Поэтому и мы ограничим время выполнения эксперимента 12 часами (720 минут). Для этого перейдем на вкладку *Модельное время* и укажем *начальное время* — 0.0, а *конечное время* — 720.0.

После этого возвращаемся во вкладку *Основные* и нажимаем на кнопку *Создать интерфейс.*

Обратите внимание, что теперь, если щелкнуть на кнопке запуска модели, для запуска будут доступны два эксперимента. Один из них — Simulation — простой эксперимент по симуляции модели, который мы запускали ранее. Второй же эксперимент и есть оптимизационный. Запустим его.

Перед нами откроется новая форма, в которой щелкнем на кнопку *Запустить оптимизацию.* На *рис.* 7 показан результат выполнения такого эксперимента. На графике справа отображена зависимость значений оптимизируемого параметра от номера итерации (одна итерация — одно выполнение эксперимента при определенных значениях “перебираемого” параметра numberOfCashiers). Причем красным отображено лучшее недопустимое значение, т.е. значение, полученное без учета ограничений, наложенных на оптимизируемую модель. А синим — лучшее допустимое. Мы видим, что с увеличением количества итераций значение оптимизируемого параметра стремится к пяти. Это же значение показано в желтом окошке, сле-

ва от графика. Следовательно, для заданных нами начальных условий наименьшее возможное количество работников будет равным пяти.

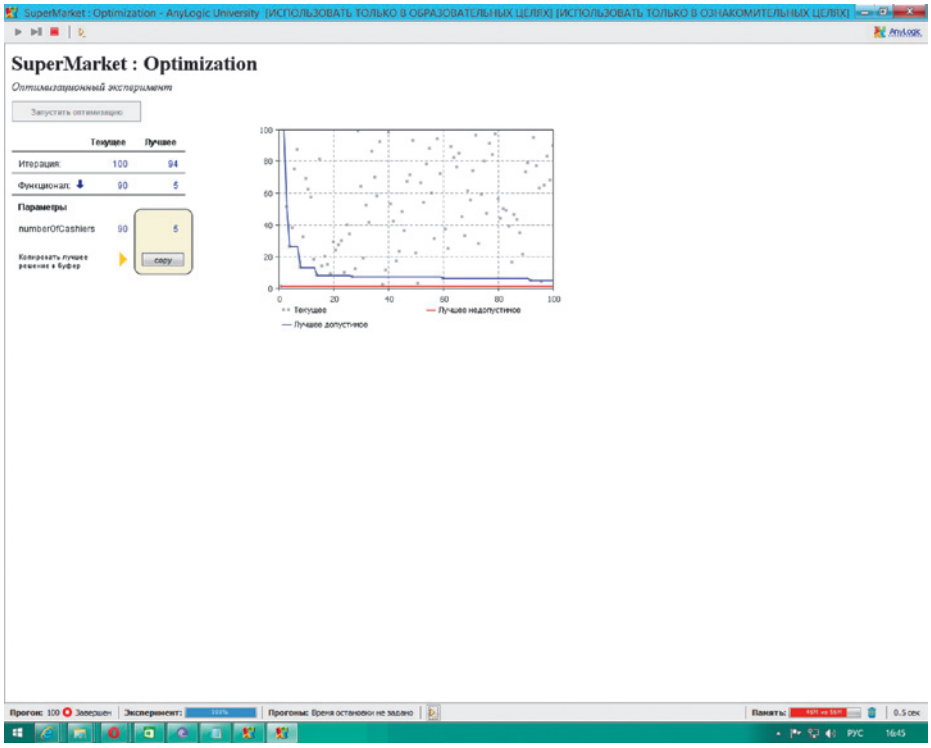


Рис. 7

Дополнительное задание к части 2

В процессе оптимизации может возникнуть необходимость в многократном изменении входных данных (у нас это интенсивность потока клиентов). При этом каждый раз закрывать эксперимент и изменять значение интенсивности вручную — достаточно долгая и неудобная для пользователя процедура. Гораздо проще было бы, если бы в окне эксперимента располагался ползунок, регулирующий входной параметр. Для того чтобы добавить его, перетащите *ползунок* из палитры *Элементы управления* и самостоятельно настройте его на управление потоком клиентов.

Литература

1. *Качала В.В.* Основы теории систем и системного анализа. М.: Горячая линия-Телеком, 2007.

2. *Форрестер Дж.* Основы кибернетики предприятия (индустриальная динамика). М.: Прогресс, 1971.

3. *Шеннон Р.* Имитационное моделирование систем — искусство и наука. М.: Мир, 1978.

4. *Медоуз Д.Х.* Азбука системного мышления. М.: Бином. Лаборатория знаний, 2010.

5. *Калинин И.А., Самылкина Н.Н.* Информатика. 10-й класс. Углубленный уровень. Учебник. М.: Бином, 2013.

6. *Калинин И.А., Самылкина Н.Н.* Основы имитационного моделирования. М.: “Первое сентября: информатика”, № 5, 2013.