10 класс

Проект к главе 3 «Имитационное моделирование»

Теоретический минимум

Теоретический материал учебника 10 класса, посвященный **моделированию,** раскрывает суть основного метода познания информатики и применение системного подхода, широко используемых в других научных дисциплинах. В дальнейшем понятие *модели* используется постоянно, либо при описании того или иного способа организации автоматизированной обработки информации, либо как способ описания сложного объекта.

В любой вводной статье или лекции, посвященной моделированию в курсе информатики, всегда говорится, в частности, об огромном количестве практических задач прогнозирования и управления, которые решаются с помощью этого метода.

К сожалению, подтвердить эту абсолютно верную мысль на практических примерах в школьном курсе затруднительно – методы и средства, которые доступны обычно школьникам, позволяют решать только очень упрощенные или очень узкие задачи. Задачи же, сопряженные с реальной практикой, с помощью этих методов в школе решать трудно (например, численное решение оптимизационной задачи – предмет отдельного курса в высшем учебном заведении) – недостаточно времени и математической подготовки.

Приблизить материал к практическим задачам из реальной жизни можно, применив один из самых старых и мощных подходов к созданию компьютерных моделей – ***имитационное моделирование***.

В рамках темы **«Моделирование»** авторами рассматриваются не только традиционные вычислительные модели, но и модели имитационные: агентные, дискретно-событийные и системно-динамические.

Для практикума одна из ведущих мировых компаний-разработчиков средств имитационного моделирования, абсолютный лидер российского рынка, **компания XJ Technologies (AnyLogic),** предоставляет специально адаптированную к условиям школьного курса версию среды AnyLogic, позволяющую создавать, демонстрировать и исследовать широкий спектр моделей из самых разных областей практической деятельности. Использование этой среды позволяет не только теоретически обсудить важность и возможности методов моделирования, но и продемонстрировать их важность и возможности для решения практических задач.

Для знакомства с теорией и практикой имитационного моделирования предлагается рассмотреть возможности среды на содержательных задачах, а также выполнить часть практикума по данной теме в виде двух практических работ.

Описание метода

Суть этого подхода: имитировать поведение системы, представив ее как множество простых взаимодействующих объектов, моделируя их поведение и взаимодействие и рассматривая результаты для всей системы в целом. Объект мы описываем как набор параметров и способов их изменения.

Применяя такой подход, мы можем:

- изучить поведение системы, не имея общих аналитических соотношений;

- проследить поведение системы в динамике, т. е. на каждом шаге процесса;
- исследовать поведение системы, в которой возможны случайные вариации поведения объектов.

Существует несколько видов таких моделей:

1. ***Дискретно-событийные модели.*** Такие модели описывают поведение системы как набор последовательных событий. Например, так можно описать рост пшеницы (процесс состоит из хорошо известных стадий) или работу учреждения (его работа описана инструкциями и регламентами). В такой системе описываются состояния (например, ожидание поступления на склад запасной части) и события, т. е. изменения состояния.
2. ***Агентные модели.*** Такие модели построены в виде набора взаимодействующих отдельных объектов (агентов), каждый из которых имеет какие-то заданные правила поведения и на основе поступающих сведений принимает решение об их применении. Например, такой агент может описывать поведение особи в стаде или человека в толпе. Поведение всей системы будет складываться как результат взаимодействия агентов, но спрогнозировать его математическим соотношением трудно. Например, поведение толпы на выходе из здания.
3. ***Модели системной динамики.*** Этот метод предназначен для исследования поведения сложных систем с большим количеством обратных связей и зависимых параметров, например крупного города, большого производства, демографической ситуации. При таком виде моделирования систему представляют в виде взаимодействующих объектов. Каждый объект на самом деле может быть очень крупной системой, но при моделировании мы считаем его единым целым, пропускающим через себя, формирующим или поглощающим потоки разного рода (финансовые, материальные и другие).

Имитационные модели - один из самых распространенных методов анализа и предварительной проверки сложных систем, принятия решений в условиях множества разнородных связанных факторов, оптимизации расписания и т.д.

Строить такие модели можно различными способами. В особо сложных и специфических задачах (например, при моделировании транспортной системы города) такие модели разрабатываются как специализированные программы, но в целом ряде случаев можно разработать ее в рамках системы создания таких моделей.

Разберем несколько примеров таких моделей.

Первый пример - **моделирование потока пешеходов с помощью агентной модели**.

Предположим, что у нас есть какой-то путь пешеходов (подземный переход, магазин и т.д.). Нас интересует – каким образом повлияет установка витрины (или создание маленького магазинчика на этом проходе) на движение пешеходов?

Определить эти параметры с помощью математической модели очень трудно, но можно описать среду движения, пешехода (и описать его движение) и создать имитационную модель, в которой пронаблюдать за движением пешеходов.

Мы создаем модель, в которой пешеходы - это точки (точнее, круги), перемещающиеся по плоскости. Правила движения пешехода сравнительно просты – пешеход движется с постоянной скоростью (примерно от 0,5 до 1 метра в секунду) к заданной цели. Если он видит препятствие (стену) или медленно движущегося пешехода, он старается его обойти. Время в модели считается в условных единицах, для определенности будем считать их секундами.

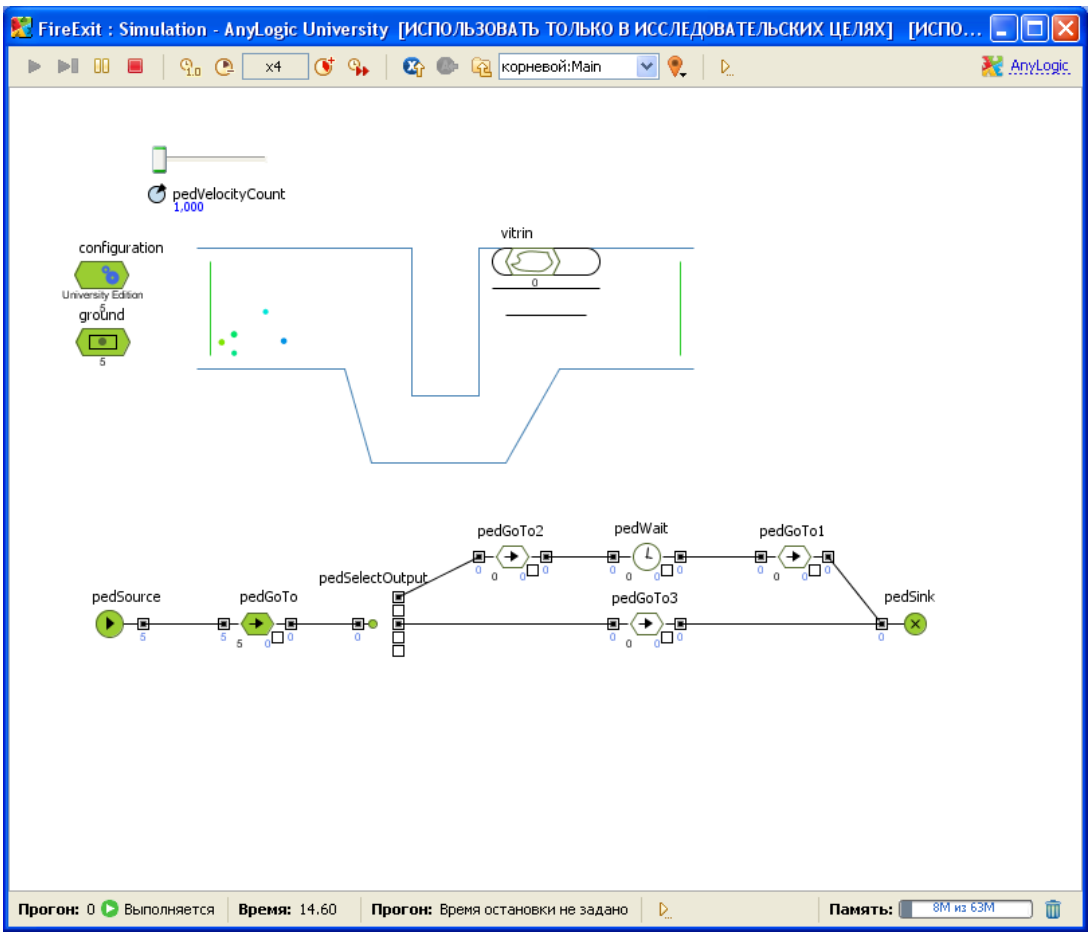


Рис. 3.3. Демонстрационное окно модели

Модель движения пешеходов мы создадим с помощью специализированной среды имитационного моделирования[1], в ней уже есть специально подготовленные средства (мы разберем их использование в практикуме) для моделирования пешеходов. Модель будет представлять собой программу, которая обеспечит учет перемещения пешеходов с заданной скоростью, сбор статистики и визуализацию происходящих событий.

В окне модели видно:

1. Среду перехода. Пешеходы появляются в левой части и проходят направо с обычной скоростью до выхода. После поворота они видят с левой стороны прохода витрину и могут либо свернуть к ней, либо пройти на выход.
2. Алгоритм работы: пешеходы появляются (pedSource) с некоторой частотой – по умолчанию 1000 в час, идут (pedGoto) до некоторого места, где замечают витрину (отмечено чертой) и принимают решение (pedSelectOutput) идти дальше (большая часть, 7 из 10 пешеходов) или подойти к витрине (3 из 10). Итоговая цель пешеходов – выход (pedSink)
3. Параметр, которым мы можем управлять – количество пешеходов в час.

Запустим модель и посмотрим результат. Для 1000 пешеходов в час никаких затруднений нет:

[1] Использован пробный вариант среды имитационного моделирования AnyLogic v.6.5 University

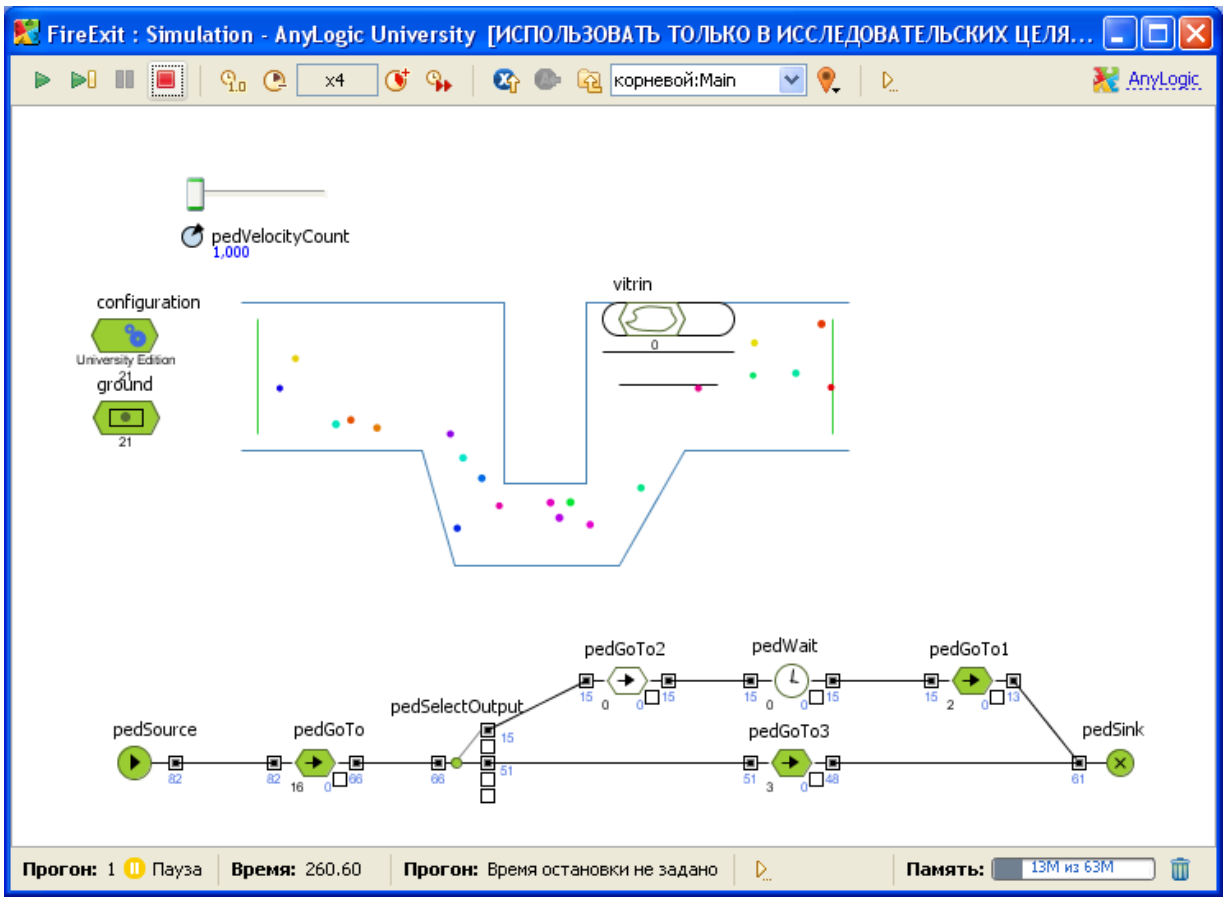


Рис. Исследование модели: 1000 пешеходов в час

В переходе одновременно находится примерно 20 человек (подсчет статистики ведется на всех узлах).

Для 6000 пешеходов в час на повороте появляется затруднение, но в целом пропускной способности достаточно:

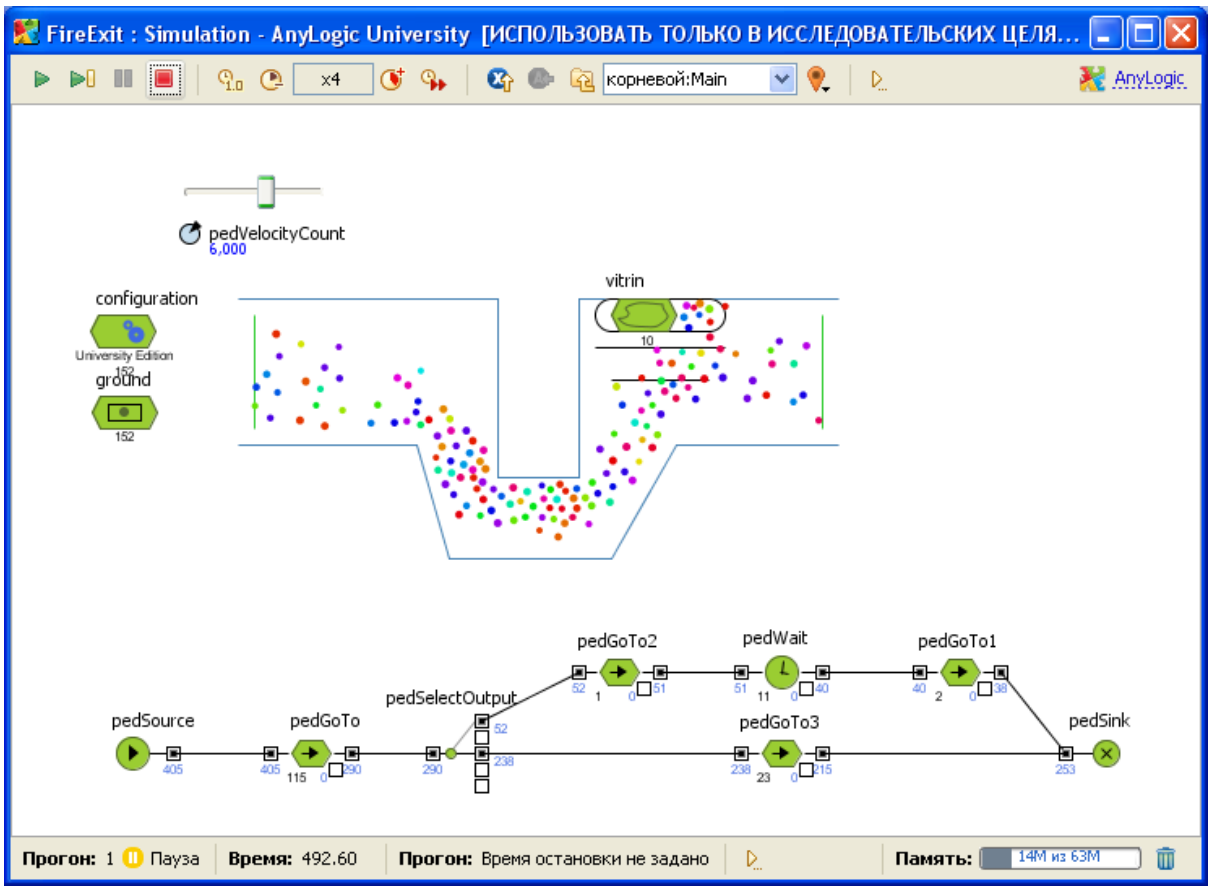


Рис. Исследование модели: 6000 пешеходов в час

А теперь, при том же потоке пешеходов изменим модель – сделаем витрину более заметной, видимой почти от выхода. Для этого мы просто сдвинем место принятия решения о повороте.

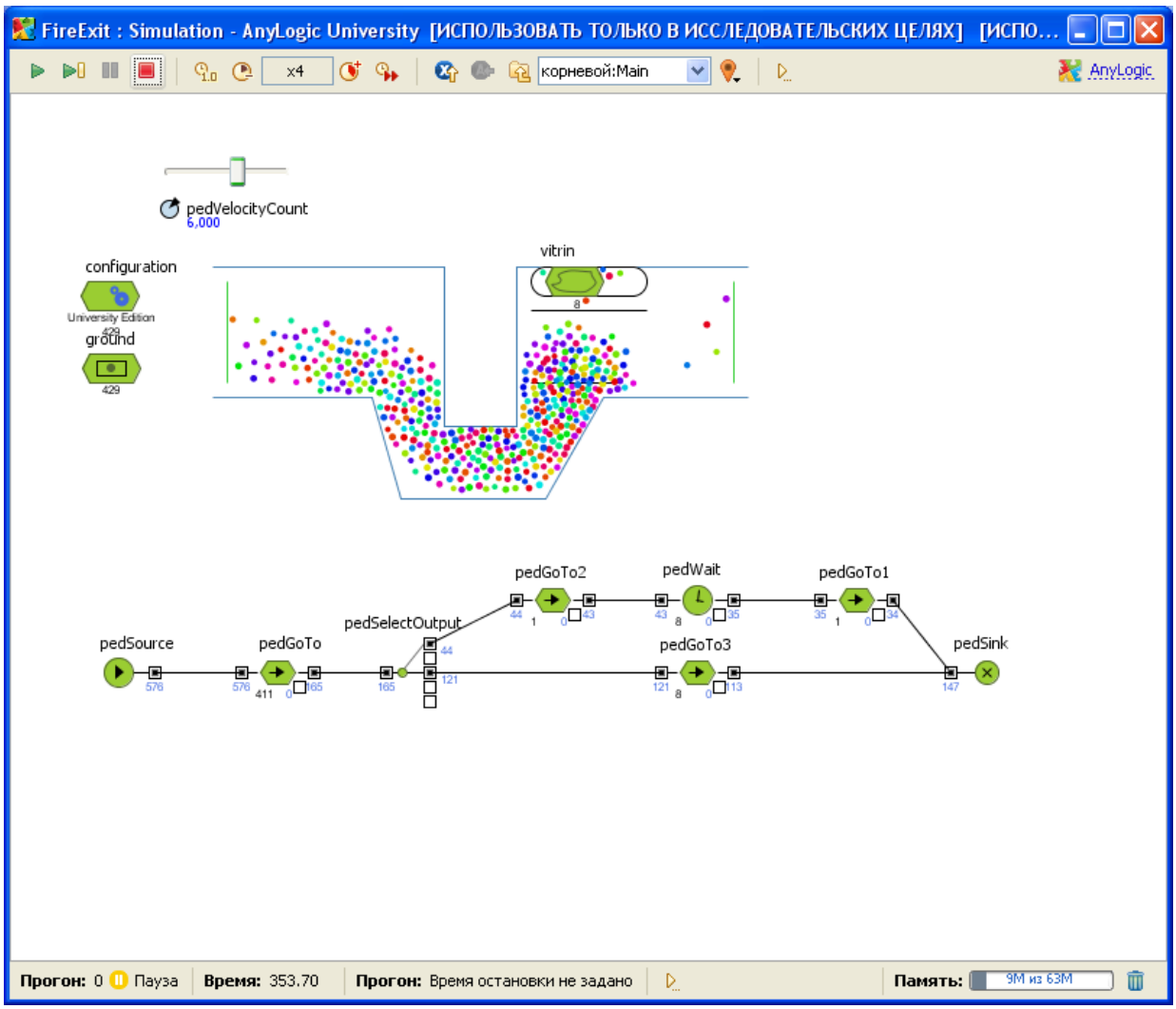


Рис. Исследование модели: 6000 пешеходов в час, витрину замечают от поворота

Хорошо видно, что возникла пробка. Ее можно увидеть и численно: разница между вошедшими (576 на входе) и вышедшими (147 на выходе) стала гораздо больше, хотя к самой витрине народу подошло меньше (43 вместо 51).

Причина пробки – время, которое люди потратили, чтобы осмотреться и принять решение. Их движение только немного замедлилось, но и этого хватило.

Стоит обратить внимание, что привлекательность витрины мы не меняли, только заметность.

Для создания более наглядной модели в качестве подложки можно использовать план здания, а стены-препятствия расставить точно так, как они стоят в реальности.

Конечно, создать такую модель для конкретных условий можно только тщательно исследовав поведение настоящих пешеходов: как они движутся, сколько из них смотрит на витрины, как изменяется при этом их движение. Именно поэтому такие параметры – предмет многочисленных и довольно сложных исследований.

Аналогичный подход можно применить к очень многим процессам, например, создав модели машин, веществ и аппаратов, исследовать организацию производства.

Такие модели – мощный и универсальный способ исследования влияния изменений в поведении отдельных объектов-агентов на всю модель. В математической

модели, например, было бы очень сложно увидеть, как повлияет изменение тактики поведения жертв на популяцию хищников. Иногда очень простые изменения параметров приводят к существенно другим результатам, что в формулах можно учесть только после того, как эффект будет замечен.

Второй упомянутый нами вид моделей - **модели дискретно-событийные.**

В агентной модели агент (точнее - программный код, моделирующий его деятельность изменением параметров), как мы помним, принимал решения о своих действиях, исходя из восприятия окружения (то есть получая разрешенные ему сведения о состоянии среды). При моделировании же многих организованных процессов все решения уже приняты заранее, то есть известно, в каком порядке, кто и как должен действовать.

Моделируя такие системы, понятием агента (и самим агентным подходом) не пользуются, а используют понятие ***заявки на обслуживание*** – обращения к системе для достижения какого-то результата. Порядок обработки заявки задается в системе в виде некоторой последовательности событий: появления заявки, передачи заявки от одной стадии к другой, задержки (моделирующей выполнение стадии), накопления заявок в очереди.

При возникновении таких событий также оценивается достаточность ресурсов, например, занят или свободен какой-либо аппарат или комната.

Время в таких системах чаще всего представляется в виде условных дискретных шагов, поскольку заявка переходит из одной стадии в другую сразу, то есть дискретно.

Такой подход позволяет проанализировать работу системы под заданной нагрузкой, определить заранее «узкие» места, оценить количество необходимых ресурсов и качество работы системы в целом.

Рассмотрим такую модель на примере упрощенного **представления транспортного маршрута.**

Наш маршрут состоит из трех остановок: начальной, промежуточной и конечной. Маршрутное транспортное средство (будем его в дальнейшем называть маршруткой) имеет 12 мест и ходит по некоторому расписанию. В маршрутку можно посадить не более 12 человек.

На первой остановке люди только садятся, на промежуточной – выходят (если это предполагалось) и садятся (если есть места), а на конечной – только выходят.

Для нас основным критерием работы линии будет количество ожидающих посадки людей на остановках. Еще один критерий, часто используемый в таких системах – время, проведенное заявкой в системе.

Вот рабочее окно нашей модели:

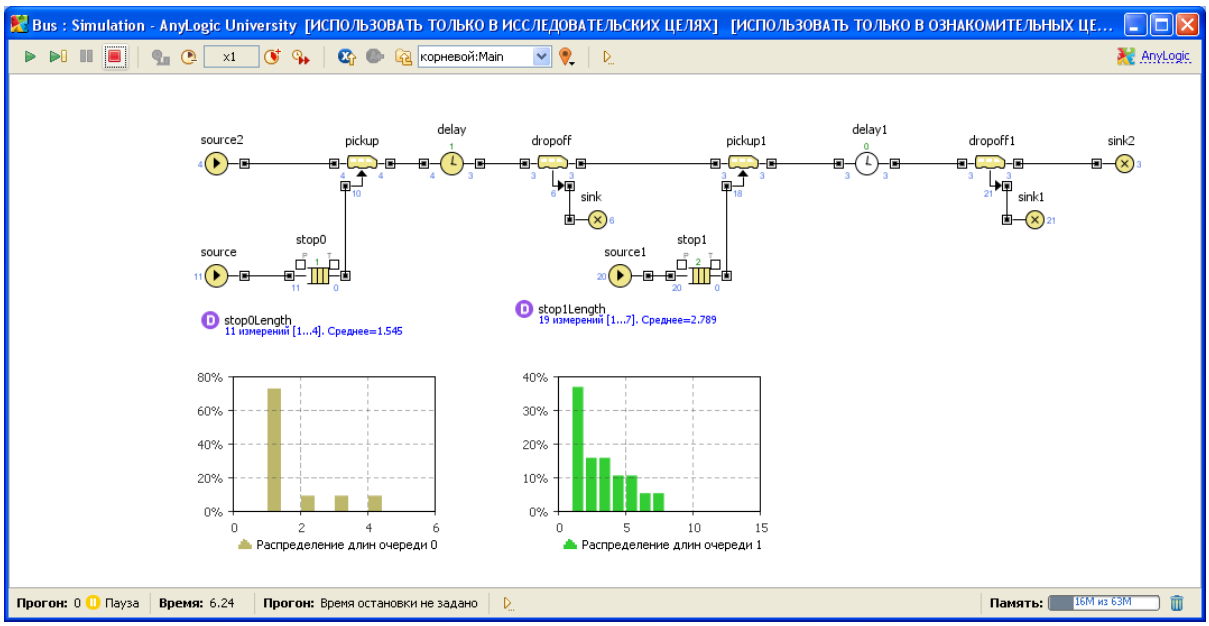


Рис. Транзакционная модель маршрута

В этой модели, в отличие от предыдущей, мы не используем анимацию для визуализации, поскольку нас интересуют только численные оценки. Длины очередей на каждом шаге мы фиксируем в специальных списках и отражаем на гистограммах, оценивающих длину двух очередей.

На этом рисунке видно, что длина очереди на первой остановке не превышала 4, а на второй, гораздо более интенсивной, доходит и до 7. Насколько такое допустимо – вопрос изначально заданных критериев.

Если пассажиры не будут учитывать расписания маршруток, то при длительном перерыве в их движении (в нашем случае – от 18 до 30 точки отсчета) ситуация может выглядеть так:

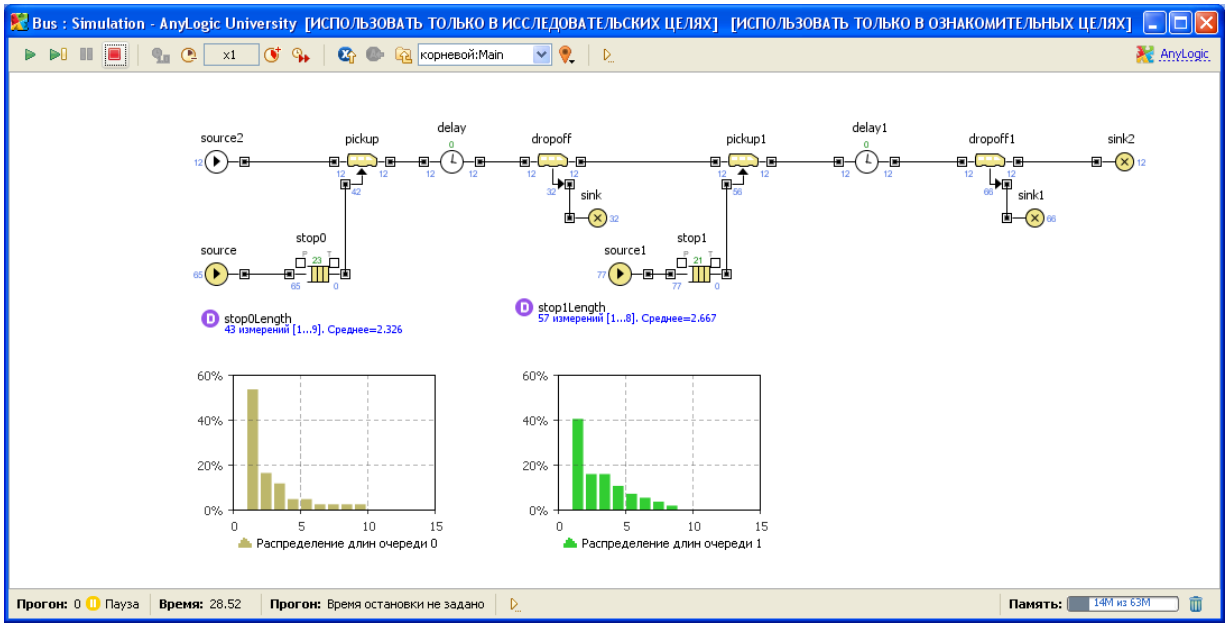


Рис. Исследование модели

То есть на остановках начнут скапливаться ожидающие маршрутку (23 на первой остановке и 21 на второй, поскольку там последняя маршрутка прошла позже). Чтобы это скомпенсировать, потребуется больше, чем 4 маршрутки, потому что не обязательно выходят все люди и постоянно подходят новые. При необходимости можно подобрать более адекватное расписание или увеличить количество маршруток.

Эта модель упрощена, поэтому мы не оцениваем задержку при входе и выходе пассажиров[2], не рассматриваем работу станций-пересадок, возврат маршрутки назад, реальное время маршруток в пути. Очевидно, что при более тщательном подходе можно достаточно точно описать работу транспортной сети города и, оценив интенсивность появления пассажиров на остановках, спрогнозировать ее поведение.

Таким образом, можно моделировать работу учреждений, складов, транспортных сетей, систем обслуживания, конвейерных производств, оценивать достаточность ресурсов и скорость их работы. В отличие от агентного подхода, этот способ требует существенно меньших ресурсов, что позволяет строить и изучать достаточно сложные модели.

В целом ряде случаев применение обоих видов моделей – хорошая альтернатива реальным «пробам». Например, значительно целесообразнее проверять достаточность пожарных водоемов, помп и расчетов на сетевой транспортной модели, а не в условиях реальных пожаров. Хотя, в последние несколько лет решения, как кажется авторам, принимаются именно на основе практического опыта и внесистемных мотивов. Реальные потери, конечно, значительно убедительнее.

Особый вид имитационных моделей - **системной динамики,** разработан и используется для прогнозирования развития крупных систем, в которых поведение отдельных объектов взаимозависимо, причем связей много. Чаще всего такие системы рассматриваются в финансово-экономической сфере и в производстве.

Существенная особенность таких моделей - что в них мы рассматриваем не модели объектов, а взаимосвязь параметров.

В моделях системной динамики всю систему представляют в виде набора взаимосвязанных объектов следующих видов:

Название объекта	**Назначение**
Уровень	Переменная, накапливающая изменения.
Связь	Обозначение влияния переменных.
Связь с задержкой	Влияние с запаздыванием.
Поток	Переменная, изменяющая значения уровней.
Поток с регулятором	Изменение значение уровней с регуляцией. Изменение значение потока на основании вспомогательной переменной.
Вспомогательные переменные	Переменные-параметры, определяющие поведение, но сами ни на что не влияющие.
Константы	Неизменные (во время симуляции) значения.
«Облако» («Озеро»)	Неисчерпаемый источник или поглотитель.

В качестве примера мы попробуем **исследовать работу таможни в некоторой стране.** *Рассмотрим типичную для таможни ситуацию, когда часть товара поставляется в обход таможни и контролирующие органы страны полагают, что бюджет недополучает с этого налоги.*

[2] В ряде городов введены в действие системы автоматического контроля пассажиров. В результате время посадки 15 человек может достигать 10-12 минут, что конечно стоит учесть в модели. Аналогично будет действовать и условие «оплата водителю при входе/выходе».

Предлагается идея: вместо того, чтобы усиливать борьбу с контрабандой (это долго, дорого и не гарантирует успеха), государство увеличивает налог на товар, компенсируя свои потери.

Поставщики товара поднимают цену при продаже, чтобы скомпенсировать выросший налог.

Известно, что в таких случаях контрабанда возрастает, потому что становится выгодной большему количеству поставщиков и потребителей.

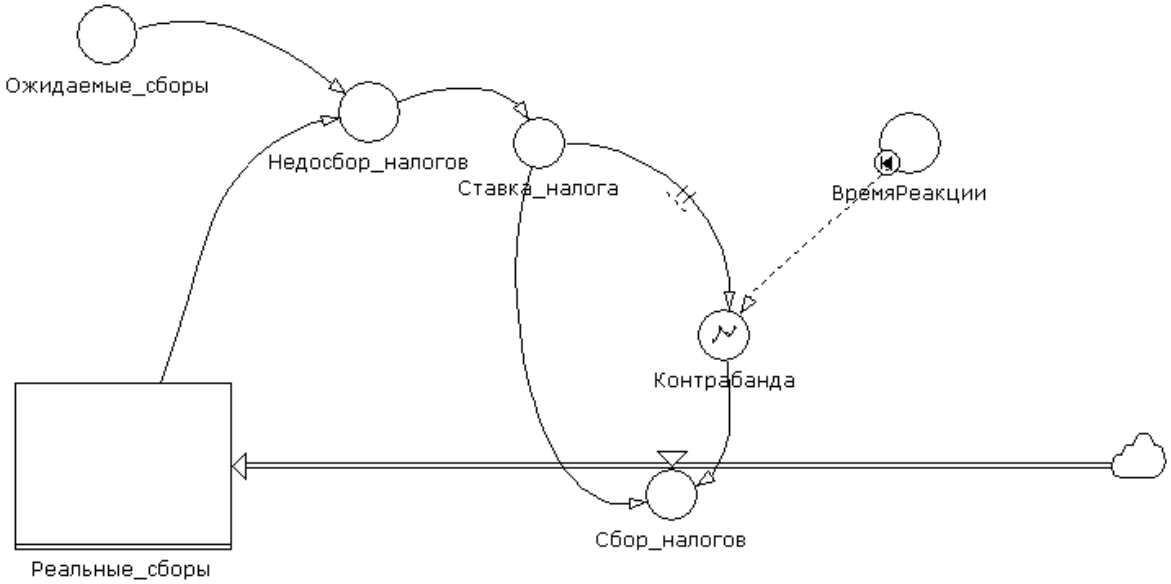
Возникает вопрос: сумеет ли государство скомпенсировать потери таким образом?

Составим модель: есть поток товара (который мы укажем в денежном эквиваленте - USD, Условных Системных Деньгах), с которого по заданной ставке взимается налог. Налоги определяют уровень реальных сборов.

Зная объем ввоза (предположим, он не меняется), мы предполагаем уровень сборов и можем оценить недосбор налогов. Исходя из недосбора, мы вычислим новую ставку налога.

Изменяя ставку, мы повлияем и на сбор налогов, и на уровень контрабанды. Уровень контрабанды изменится не сразу, а через некоторое время, требуемое для реакции системы.

Составим модель[3]:



Год	Контрабанда	Недосбор_налогов	Реальные_сборы	Ставка_налога
0	USD1,10	USD0,00	USD80,00	USD0,00
20	USD2,19	USD5,39	USD74,61	USD4,31
40	USD2,44	USD6,07	USD73,93	USD4,86
60	USD2,49	USD6,21	USD73,79	USD4,97
80	USD2,50	USD6,24	USD73,76	USD4,99
100	USD2,50	USD6,25	USD73,75	USD5,00

[3] Модель подготовлена в демонстрационной версии PowerSim Studio 8

«Проиграв» эту модель для 100 условных лет, мы выясним удивительную вещь – реальные сборы налогов не выросли, а упали. Можно экспериментировать с коэффициентами и усложнять модель, но суть дела не изменится: простой рост налогов при нежелании или неспособности бороться с контрабандой, только ухудшит ситуацию.

Такие сравнительно простые схемы встречаются достаточно часто, и имеют свои названия. Приведенная в примере схема называется «Решение, обреченное на неудачу».

Модель можно усложнить, введя уровень борьбы с контрабандой, и учесть в нем влияние на деятельность контролирующих органов возрастания объема сборов, влияние такой борьбы на время и стоимость транспортных перевозок, другие параметры.

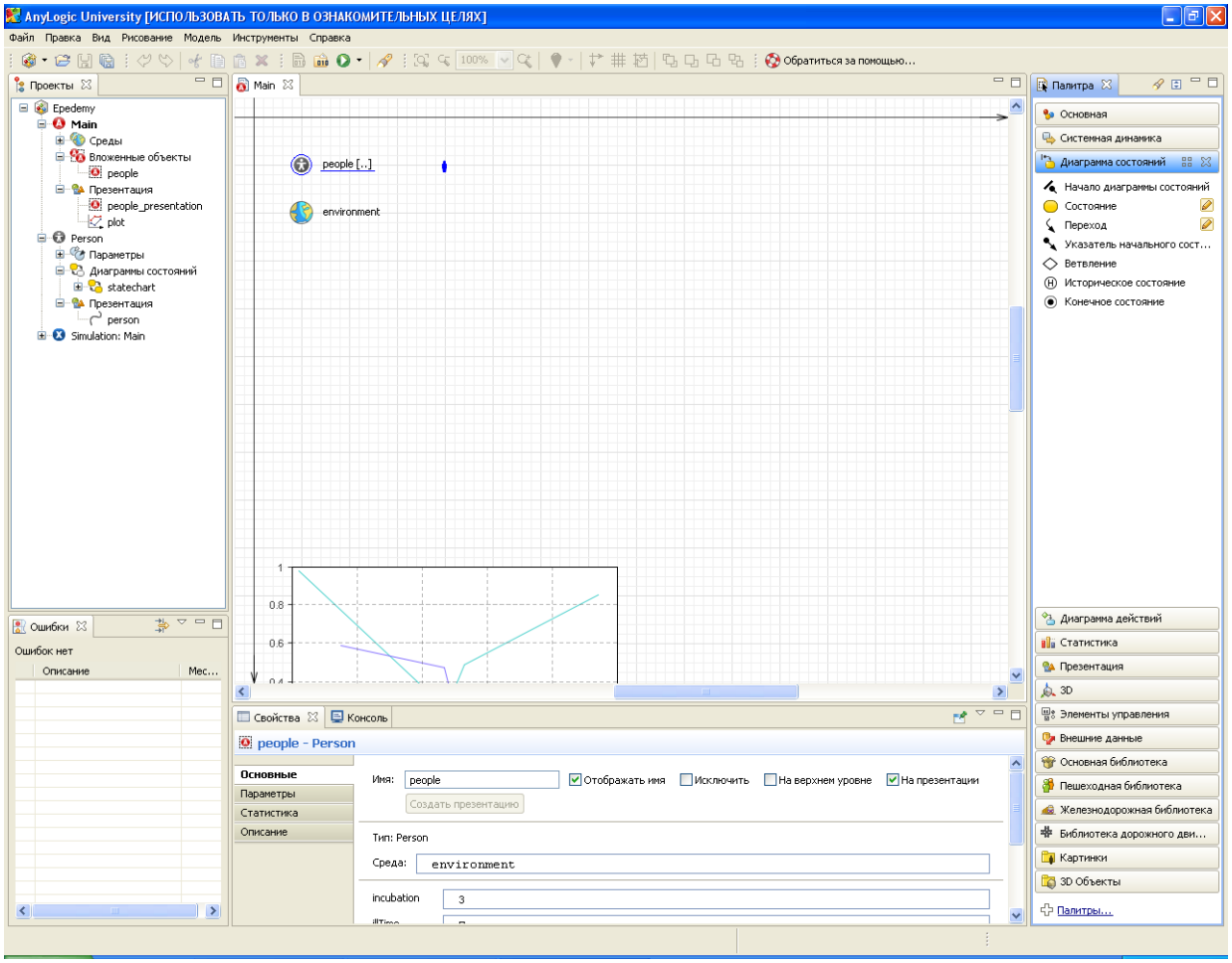
Подобные схемы (иногда, большие и сложные) позволяют изучать взаимное влияние происходящих социальных процессов и используют для принятия управленческих решений.

Системно-динамическое моделирование применяется для исследования и предсказания поведения сложных систем (например, социальных отношений) и активно используется для принятия решений.

Практикум

Чтобы практически освоить основные виды имитационного моделирования и познакомиться с возможностями этого метода, мы рассмотрим построение таких моделей с использованием среды AnyLogic. Компания производитель этой среды – российская компания XJ-Technologies предоставляет для ознакомления пробную версию (http://www.anylogic.ru/downloads), которой нам более чем достаточно. Перед началом работы рекомендуем установить её на своем компьютере.

Среда это объектно-ориентированная, и для создания моделей опирается на язык Java. Особенности языка нам изучать не потребуется, для работы достаточно общих представлений об объектно-ориентированном подходе и синтаксисе языка С.
Перед началом работы кратко опишем интерфейс среды. Основное рабочее окно среды выглядит примерно так:



С левой стороны вверху – дерево, в котором показаны открытые сейчас модели (на иллюстрации – только одна, Epedemy), а в моделях – объекты, из которых она состоит. Объекты размещены на двумерной плоскости – рабочем пространстве. В открытой сейчас модели мы видим объект-пространство – Main, объект-агент Person и стартовую страницу модели, на которой можно будет размещать настройки перед прогонами – Simulation.

Объекты размещаются в рабочем пространстве – как в графическом редакторе. Объекты можно добавлять, перетаскивая их из палитр в правой части (сейчас открыта палитра «Диаграмма состояний»)

Объекты в модели имеют свойства и методы, которые можно описывать, задавая им значения в нижней центральной части рабочего окна. Наша задача с точки зрения среды делится на две фазы: подготовка модели (описание объектов и их взаимодействия) и «прогон» - запуск модели.

Начнем с наиболее иллюстративного вида имитационных моделей – агентных моделей, на котором и покажем, как именно это делается.

Задача 1. Изучение движения учащихся через турникеты с помощью агентной модели.

В школе №0 планируется повысить уровень обеспечения безопасности.

В первую очередь, специалисты рекомендовали ограничить проход в школу посторонних, для чего на входе установят два турникета и начнут проверять всех входящих с помощью специальной карты.

По мнению директора школы реальное применение такого подхода приведет к существенным затруднениям перед началом занятий.

Как подтвердить или проверить эти предположения?

Решение:

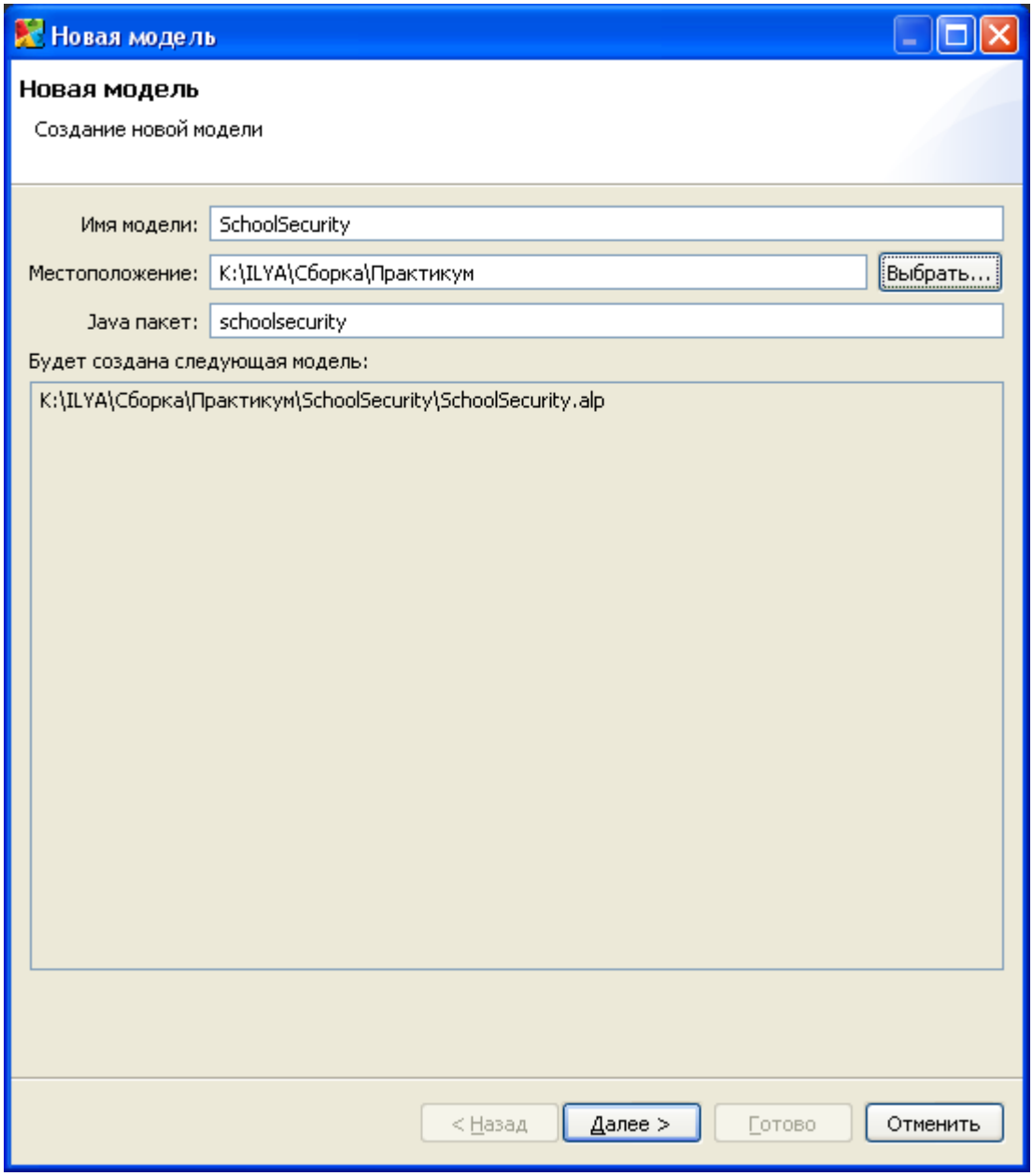
Создадим модель, показывающую первый этаж школы с турникетами и агентами-пешеходами, единственной задачей которых будет – пройти через турникет.

При этом мы учтем, что проход через турникет (а точнее – проверка) занимает некоторое время, и время от времени будет попадаться человек без документов (например, без карточки), который будет возвращаться назад.

Время прохода может меняться (человек проверяет документы медленнее, чем автомат – карточки), вероятность появления «неправильного» документа тоже.

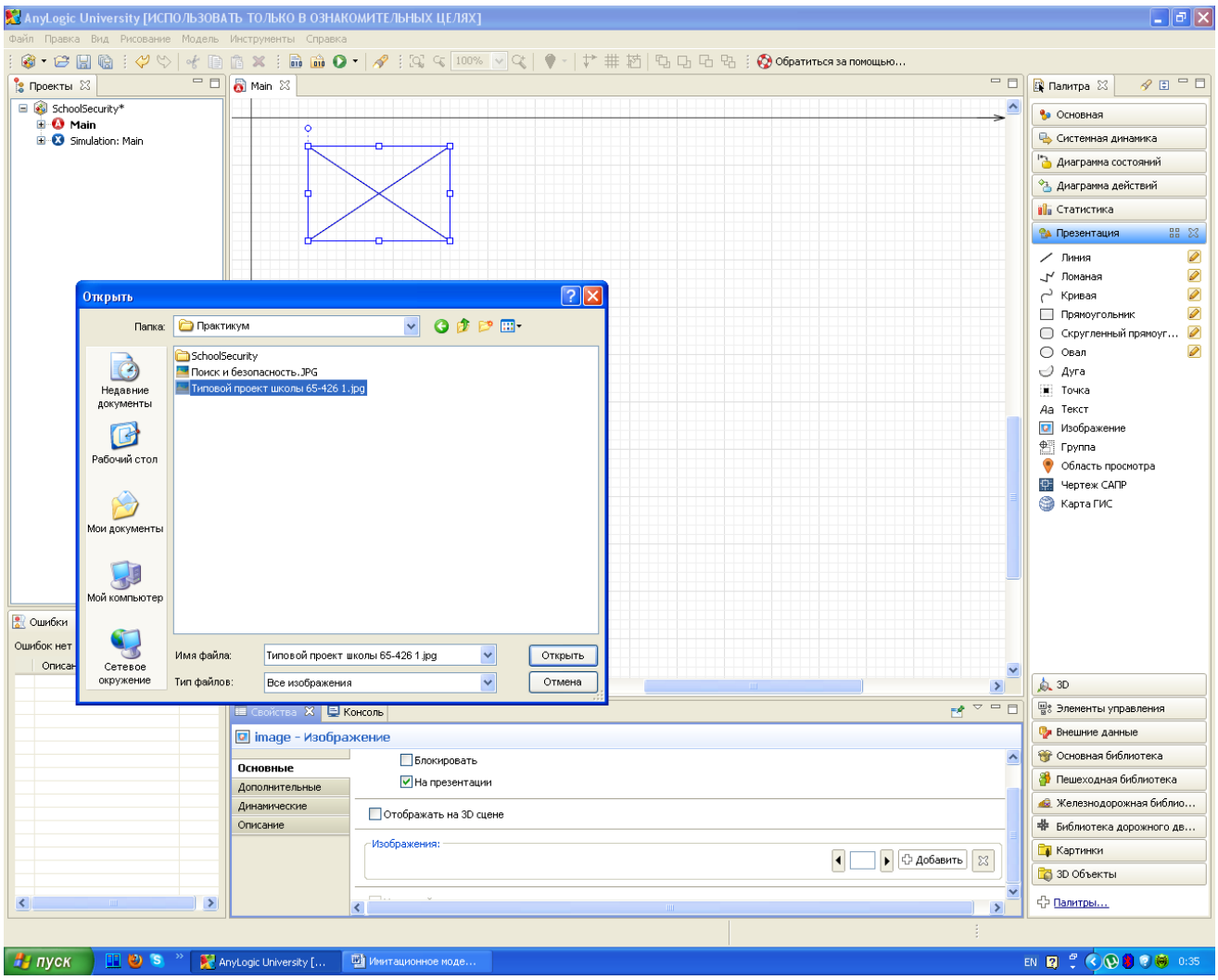
Чтобы модель выглядела реалистично, найдем и загрузим типовой план первого этажа школы. Авторы использовал план из типового проекта №45-621/1

Запустим среду, и командой «Файл\Создать» создадим пустую модель:



На втором шаге укажем, что модель создаем «с нуля», не используя заготовки.

Возьмем за основу типовой проект школы 65-426/1. Найдем в сети план первого этажа, очистим его от лишних артефактов, и разместим на рабочем листе: из набора компонентов «Презентация», перетащим на рабочее поле объект «Изображение» и добавим в его параметрах картинку плана этажа:



Вставленную картинку «растянем».

Теперь прямо поверх плана мы отметим стены (поскольку на картинке, автоматически, система стены выделить не может). Для этого построим ломаные вдоль стен. Обратите пожалуйста внимание – чтобы начать рисование ломаной нужно дважды щелкнуть по значку «карандаш» справа от объекта «Ломаная» в наборе компонентов «Презентация».

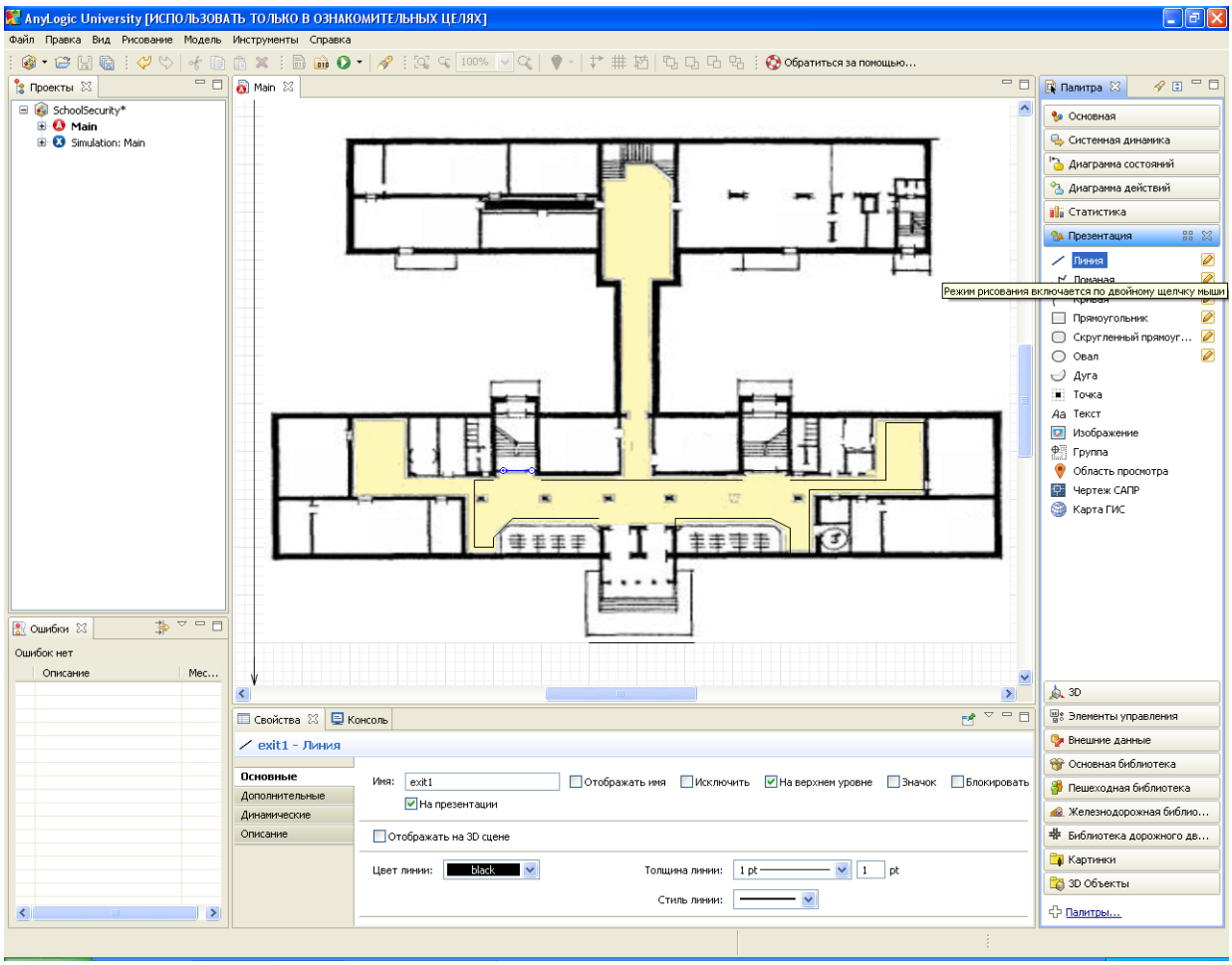
Чтобы сделать нашу ломаную «стенами», нам нужно объединить все элементы в группу – выберите их, щелкнув на них мышью с прижатой клавишей Shift, нажмите правую клавишу мыши и выберите в меню «Группа/Создать». Появившуюся группу назовем Walls.

Создадим стартовую линию – «источник» пешеходов. Линию мы расположим перед крыльцом – с точки зрения модели, нам не важно, откуда люди приходят к школе.

Чтобы её нарисовать, перетащим объект-линию, подгоним размеры и зададим линии имя: enter

Внутри школы отметим пару линий – выходов. На самом деле это, конечно не выходы, а просто подъемы на 2 этаж, но опять же – с точки зрения модели важно только то, что люди уходят с этажа. Схема, конечно, очень упрощена – в обычной ситуации людям надо еще, как минимум, раздеться.

Линии назовем exit1 и exit2, примерно так, как показано на иллюстрации:



Пока что на нашей модели ничего происходить не будет, поскольку мы ничего задающего движение пешеходов еще не сделали.

Построим общую логическую схему движения из готовых блоков. Блоки нужно перетащить по одному из раздела «Пешеходная библиотека» и расположить примерно так, как показано на рисунке.

Блоки будут такие:

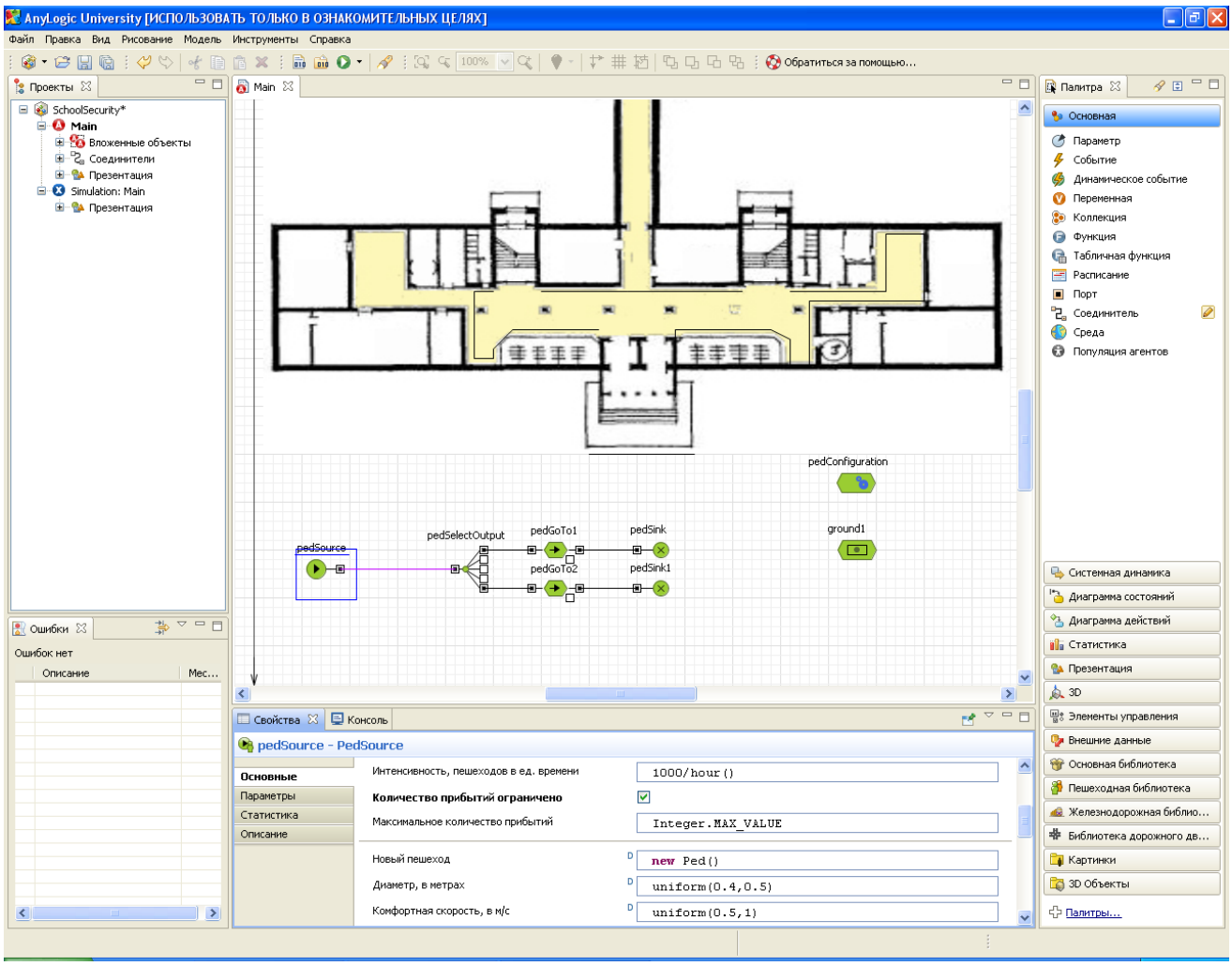
pedSource – источник пешеходов.

pedSelectOutput – выбор выхода

pedGoto – движение пешехода. Этих блоков будет два – поскольку у нас два выхода.

Сейчас для нас не так важно, что происходит за пределами турникетов, поэтому мы не учитываем того, что люди могут от каждого входа пойти к любому выходу.

Блоки нужно связать между собой. Для этого щелкнем два раза на кружке-соединителе и протянем линию до соответствующего соединителя-приемника, там щелкнем второй раз.



Кроме них, нам на модели понадобятся еще два объекта без соединений:
pedConfiguration - конфигурация
pedGround – объект-этаж.

В параметрах этажа нам нужно указать:

- Имя этажа. Для этого в поле имя напишем «ground1»
- объект-стены. D разделе свойств нужно в поле «Стены» вписать название группы линий со стенами: Walls.
- Зададим параметры источника пешеходов:
- Ограничим количество пешеходов. Для этого поставим галочку «ограничено количество прибытий» и укажем в поле «Максимальное количество прибытий» значение 600.
- Укажем объект-Этаж. В поле этаж напишем «ground1»
- Место появления. В соответствующем поле напишем имя линии – enter.
- Укажем частоту появления новых пешеходов, исходя из следующих параметров: в школе учится 600 человек и попасть внутрь они должны за полчаса. То есть параметр «Интенсивность» должен иметь значение 1200 человек в час.

Укажем параметры «разветвления» pedSelectOutput – коэффициенты предпочтения. Указать нужно вероятность выбора каждого направления, и в нашем случае они равны – по 0.5 первому и пятому исходу (остальные мы не использовали).

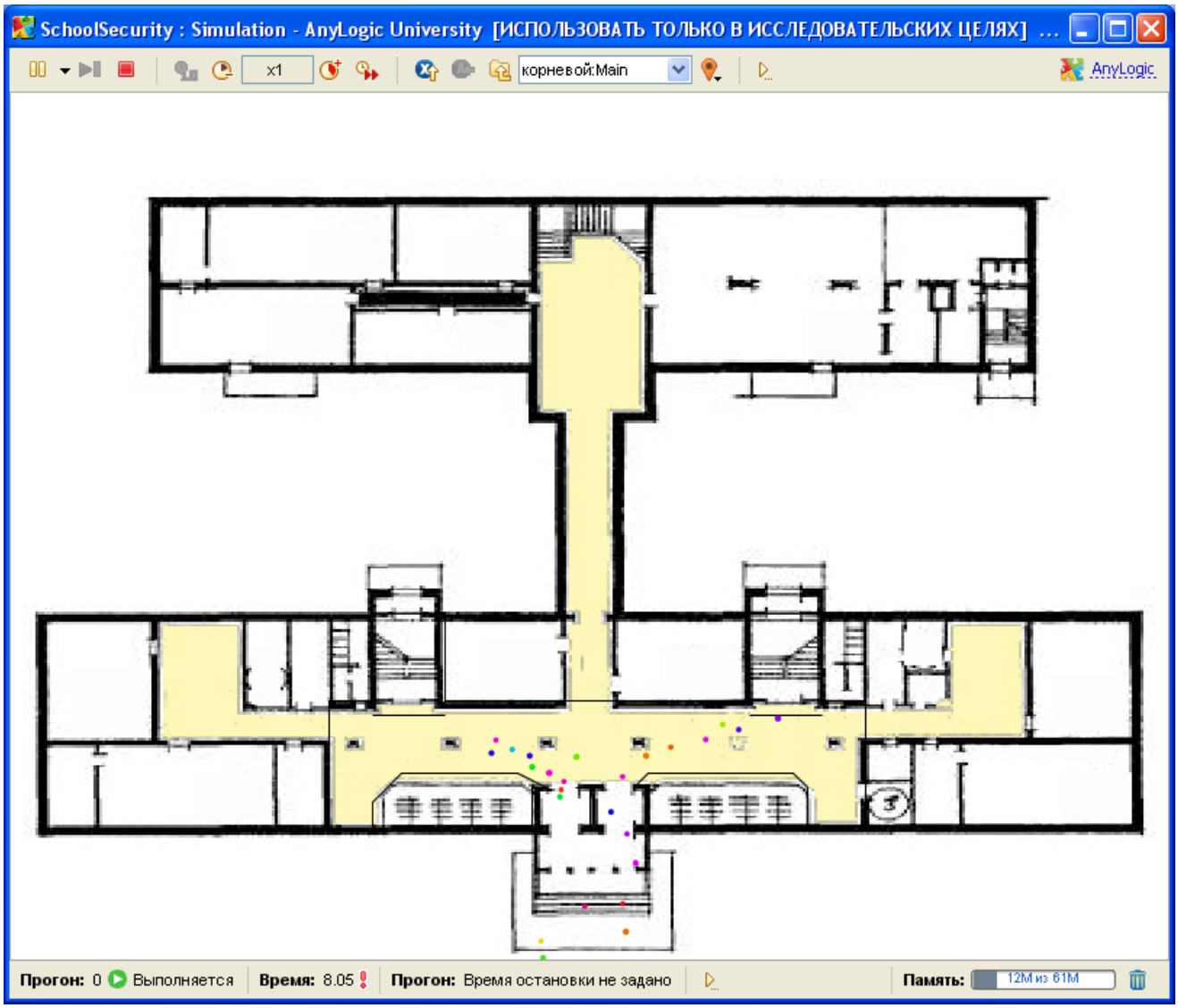
В каждом элементе движения pedGoto нужно указать цель – то есть линию выхода exit1 и exit2,

Запустим модель и, не меняя скорости, посмотрим сколько времени займет проход пешеходов.
Стоит заметить, что время в модели – «виртуальное». Фактически, одна единица времени – это одна минута. То есть нам не нужно наблюдать за прогоном полчаса, достаточно 30 секунд.

Во время прогона мы выясним несколько мелких огрехов:

- Пешеходы проходят через стенку между выходами и колонны, поскольку мы их не учли в границах. Дорисуем недостающие линии и добавим их в группу.
- В некоторых случая пешеход «не видит» свободного прохода к своему выходу и это вызовет ошибку. Сдвинем линии выходов чуть внутрь.

В нашей модели скорость движения пешеходов различается, поэтому для уверенности мы сделаем несколько прогонов.

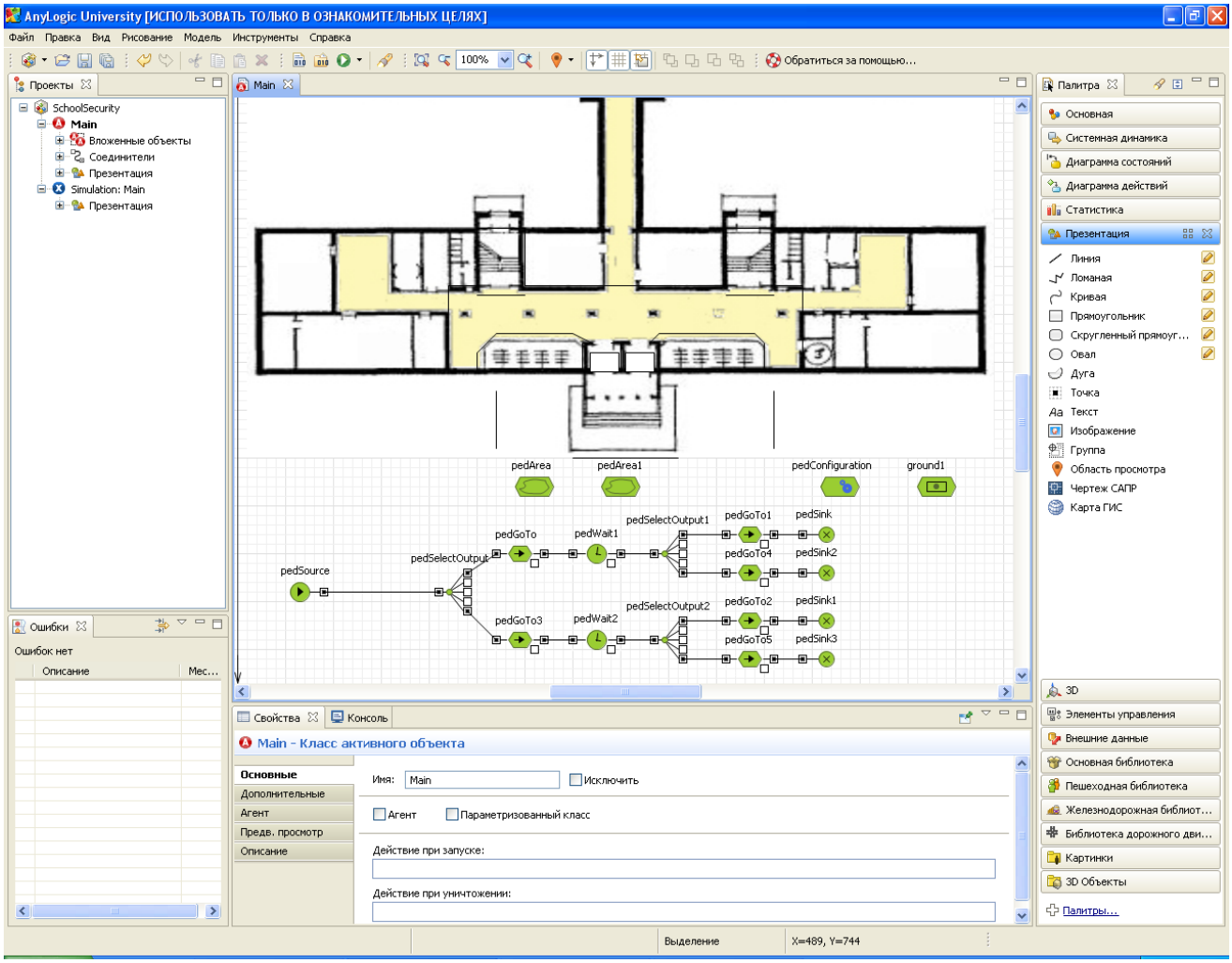


Результат будет вполне предсказуемым – никаких препятствий нет, 600 человек попадают внутрь примерно за 30 минут (то есть по дороге нет затруднений – время зависит только от частоты появления и скорости движения).

Дополним нашу модель. Теперь мы поставим два турникета (нарисуем линии gate1 и gate2), обозначим двумя прямоугольниками зоны ожидания обработки, и предусмотрим две линии – как цели для движения в случае отказа (например, забытой или отказавшей карты).
Время ожидания изменяется от 0,5 до 1 сек.

Для обработки ожидания нужно добавить два объекта-зоны: pedArea.

В схеме мы разобьем путь на три этапа: до турникета, ожидание пропуска и проход либо на этаж, либо на выход. Все объекты нужно связать, заполнив параметры.
В новых ветвлениях мы уже укажем неравное разделение: будем считать, что 95% посетителей имеют действительные карты (первый исход имеет коэффициент выбора 0,95), а 5% (0,05) – нет.

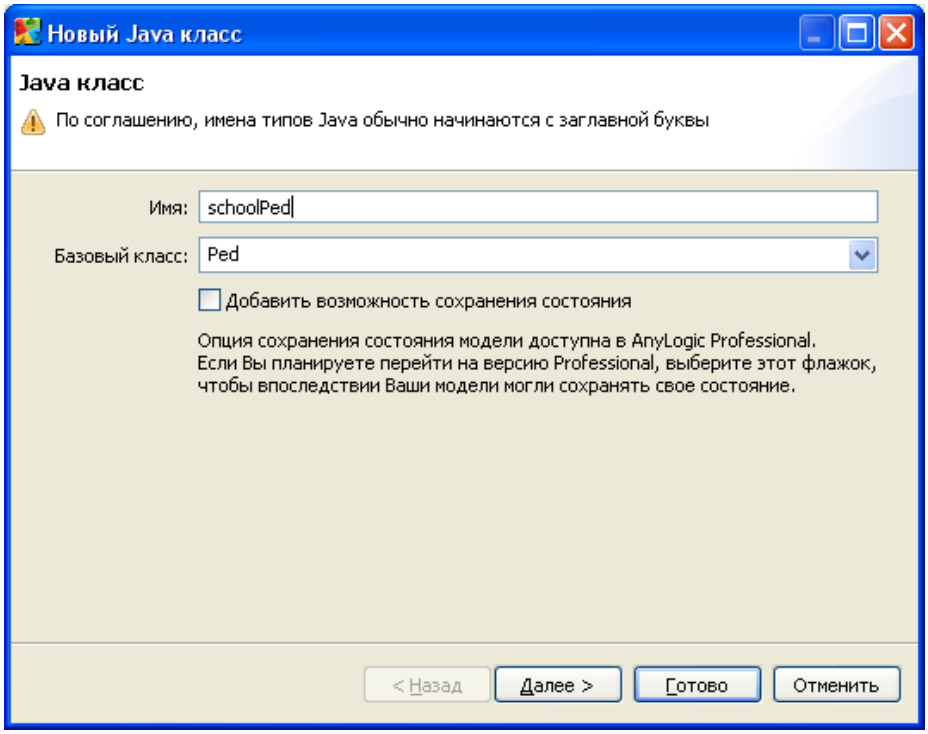


Снова запустим модель, и обнаружим, что время выросло, но не очень существенно: до 32 минут.

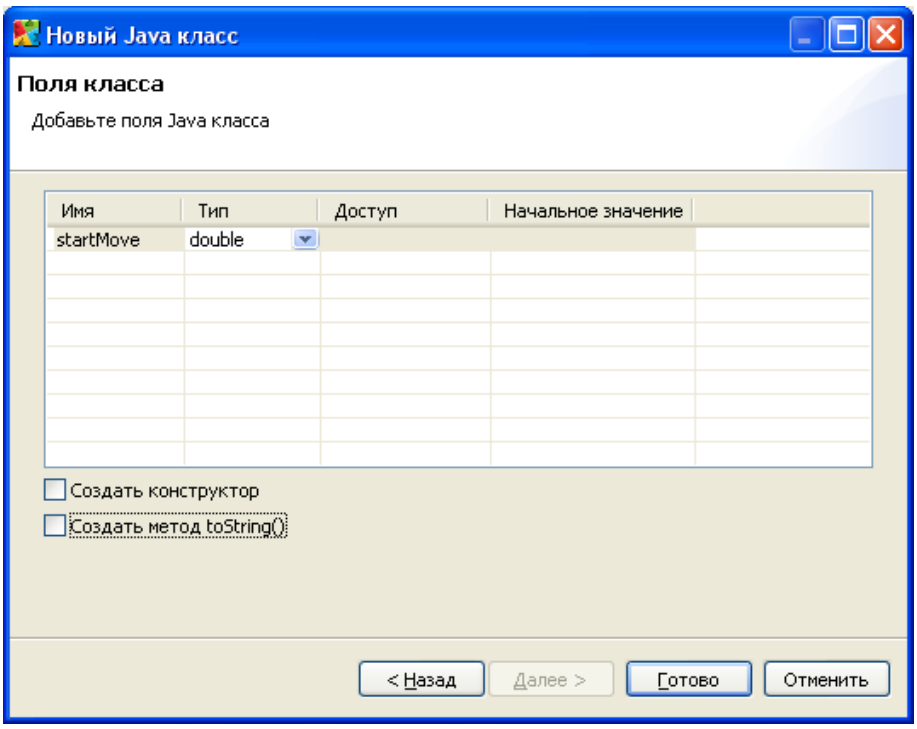
Усовершенствуем нашу модель, чтобы больше не оценивать время «на глаз». Для этого мы воспользуемся средствами сбора статистики и возможностью модифицировать класс «Пешеход».

Мы видели, что во время работы модели некоторые пешеходы могут «заблудиться» и застрять на этаже. На реальный результат они повлиять не могут, поэтому мы можем их отбросить. Параметр, который нас на самом деле интересует – это время, которое потребуется пешеходу чтобы пройти через этаж. Изначально время входа и выхода у пешехода не фиксируется, поэтому создадим нового пешехода, отмечающего время.
Для этого:

- На названии модели нажмем правую кнопку, в меню выберем пункт «Создать\Новый Java-Class»
- В появившемся окне укажем имя класса schoolPed и базовый класс – Ped



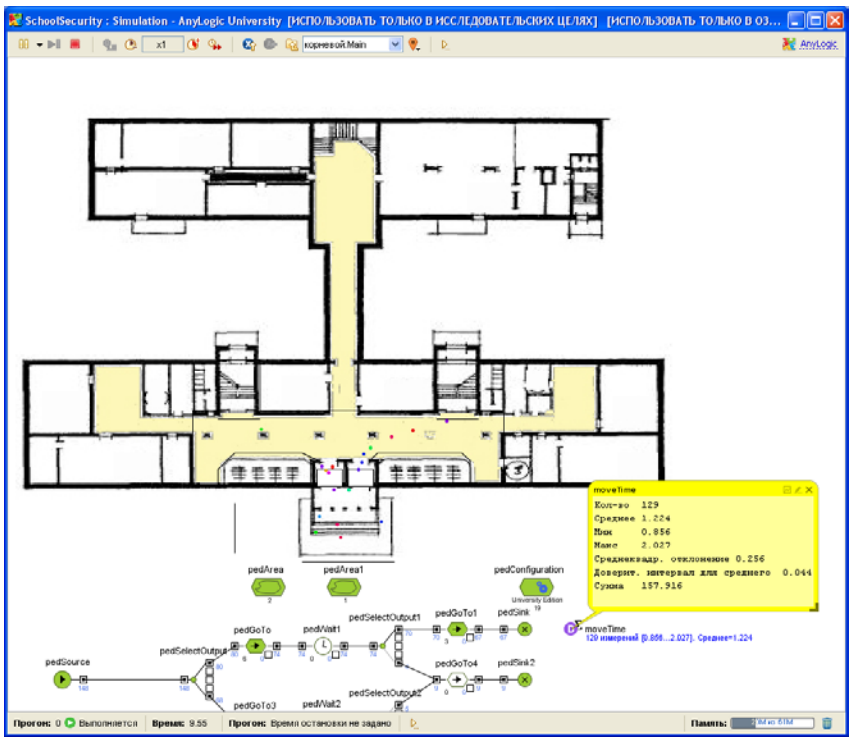
- В следующем окне добавим параметр startMove – там будет храниться время входа. Тип этого параметра – double, поскольку именно так обрабатывается модельное время.

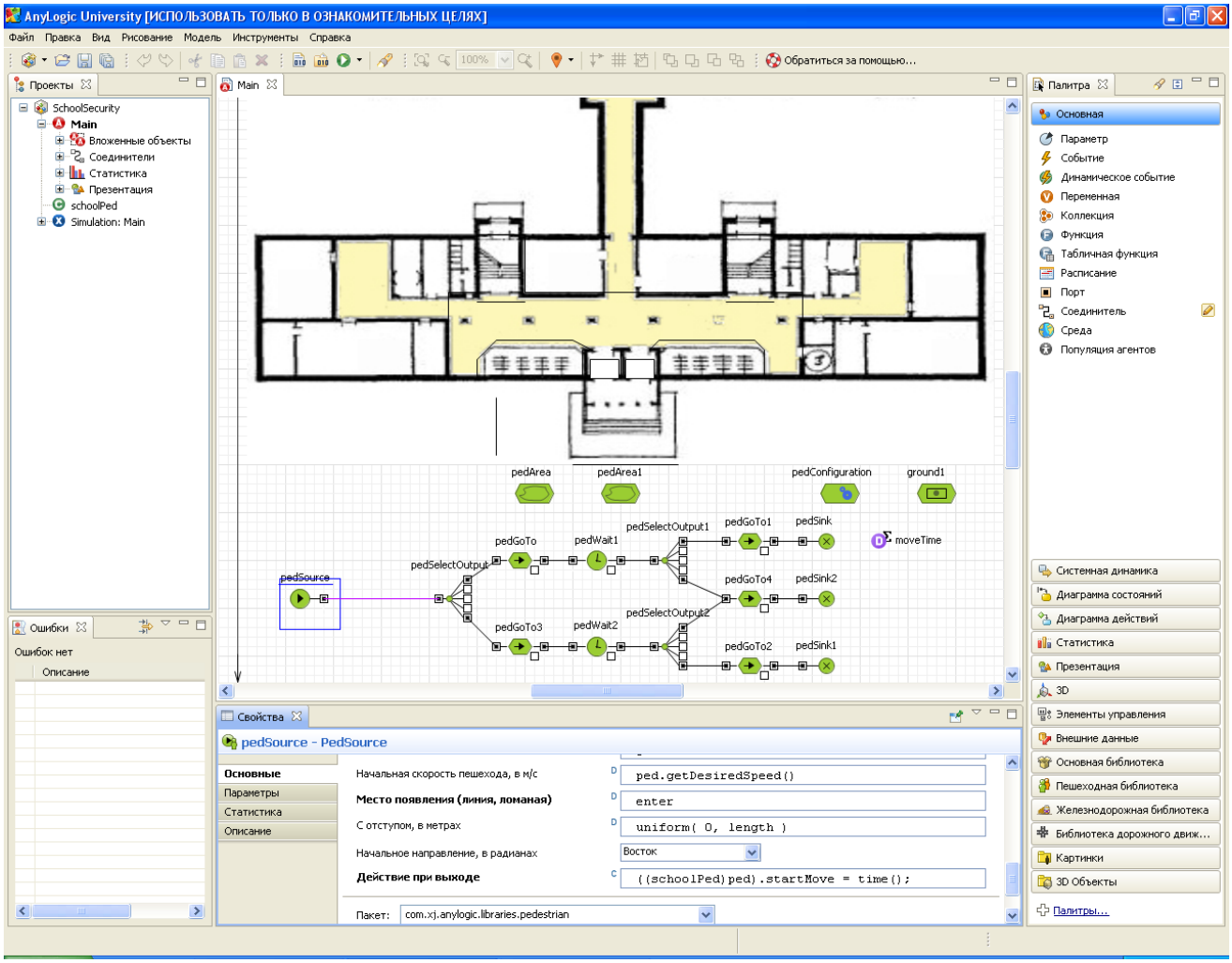


- Галочки «Создать конструктор» и «создать метод toString» уберем – мы будем записывать время сами.
- Теперь укажем на объекте источнике,
- класс пешехода – schoolPed (вместо просто Ped, который был по умолчанию). Без этого мы не сумеем обратиться к новому параметру (в базовом классе его нет)
- В пункте «Новый пешеход» вместо new Ped() укажем new schoolPed()
- В пункте «Действие при выходе» запишем в добавленный нами параметр время: ((schoolPed)ped).startMove = time();
- Поскольку в случае отказа пешеходы у нас идут в одно и тоже место, модифицируем схему – исключим лишние объекты.
- Добавим объект сбора статистики: из палитры «Статистика» перетащим объект «Статистика» и назовем его moveTime
- Сменим класс пешеход на всех объектах-выходах (также, как на входе).
- На всех объектах-выходах укажем действие при входе, добавляющее данные к статистике: moveTime.add(time()-((schoolPed)ped).startMove); Как видно, мы добавляем к статистике время – вычислив его как разницу между текущим временем и временем входа.

Теперь мы можем при выполнении прогонов не наблюдать за счетчиком, а просто отслеживать накопленные данные: минимальное, максимальное и среднее время нахождения прохода школьников.

Увидеть результаты можно просто щелкнув мышью на статистике:


SchoolSecurity : Simulation - AnyLogic University [ИСПОЛЬЗОВАТЬ ТОЛЬКО В ИССЛЕДОВАТЕЛЬСКИХ ЦЕЛЯХ] [ИСПОЛЬЗОВАТЬ ТОЛЬКО В ОЗ...
корневой.Main
AnyLogic
moveTime
Кол-во 129
Среднее 1.224
Мин 0.856
Макс 2.027
Среднеквадр. отклонение 0.256
Доверит. интервал для среднего 0.044
Сумма 157.916
pedArea
pedArea1
pedConfiguration
pedSink
pedGoTo1
pedSelectOutput1
pedGoTo
pedWait1
pedSelectOutput
pedSource
pedGoTo4
pedSink2
pedSelectOutput2
pedGoTo3
pedWait2
moveTime
Прогон: 0 Выполняется
Время: 9.55
Прогон: Время остановки не задано
Память:


AnyLogic University [ИСПОЛЬЗОВАТЬ ТОЛЬКО В ОЗНАКОМИТЕЛЬНЫХ ЦЕЛЯХ]
Файл Правка Вид Рисование Модель Инструменты Справка
Обратиться за помощью...
Проекты
SchoolSecurity
Main
Вложенные объекты
Соединители
Статистика
Презентация
schoolPed
Simulation: Main
Ошибки
Ошибок нет
Описание
Main
pedArea
pedArea1
pedConfiguration
ground1
pedSelectOutput1
pedGoTo1
pedSink
pedGoTo
pedWait1
moveTime
pedSelectOutput
pedSource
pedGoTo4
pedSink2
pedSelectOutput2
pedGoTo3
pedWait2
pedGoTo2
pedSink1
Палитра
Основная
Параметр
Событие
Динамическое событие
Переменная
Коллекция
Функция
Табличная функция
Расписание
Порт
Соединитель
Среда
Популяция агентов
Системная динамика
Диаграмма состояний
Диаграмма действий
Статистика
Презентация
3D
Элементы управления
Внешние данные
Основная библиотека
Пешеходная библиотека
Железнодорожная библиотека
Библиотека дорожного движ...
Картинки
3D Объекты
Палитры...
Свойства
pedSource - PedSource
Основные
Параметры
Статистика
Описание
Начальная скорость пешехода, в м/с
ped.getDesiredSpeed()
Место появления (линия, ломаная)
enter
С отступом, в метрах
uniform(0, length)
Начальное направление, в радианах
Восток
Действие при выходе
((schoolPed)ped).startMove = time();
Пакет:
com.xj.anylogic.libraries.pedestrian

Задания для самостоятельной работы:

1. Как изменится время, если карта будет обрабатываться от 1 до 2 секунд?
2. Сколько нужно будет времени на проход, если детям придется еще и переодеваться (задержка от 2 до 5 минут в раздевалке)?
3. Модель предполагает, что люди приходят в течении всех 30 минут равномерно. Тем не менее, очевидно что на последних 10 минутах плотность будет значительно выше. Исследуйте поведение модели с повышенной частотой появления людей.

Задача 2. Дискретно-событийная модель работы учреждения

Ежегодно работникам многих предприятий полагается проходить так называемый профессиональный медицинский осмотр. Процедура его построена примерно так: в назначенный день работник является в поликлинику, получает в регистратуре документы, получает от врача-терапевта направление и должен пройти осмотр у нескольких врачей-специалистов.

Люди выбирают специалиста сами, исходя из того, у кого очередь короче.

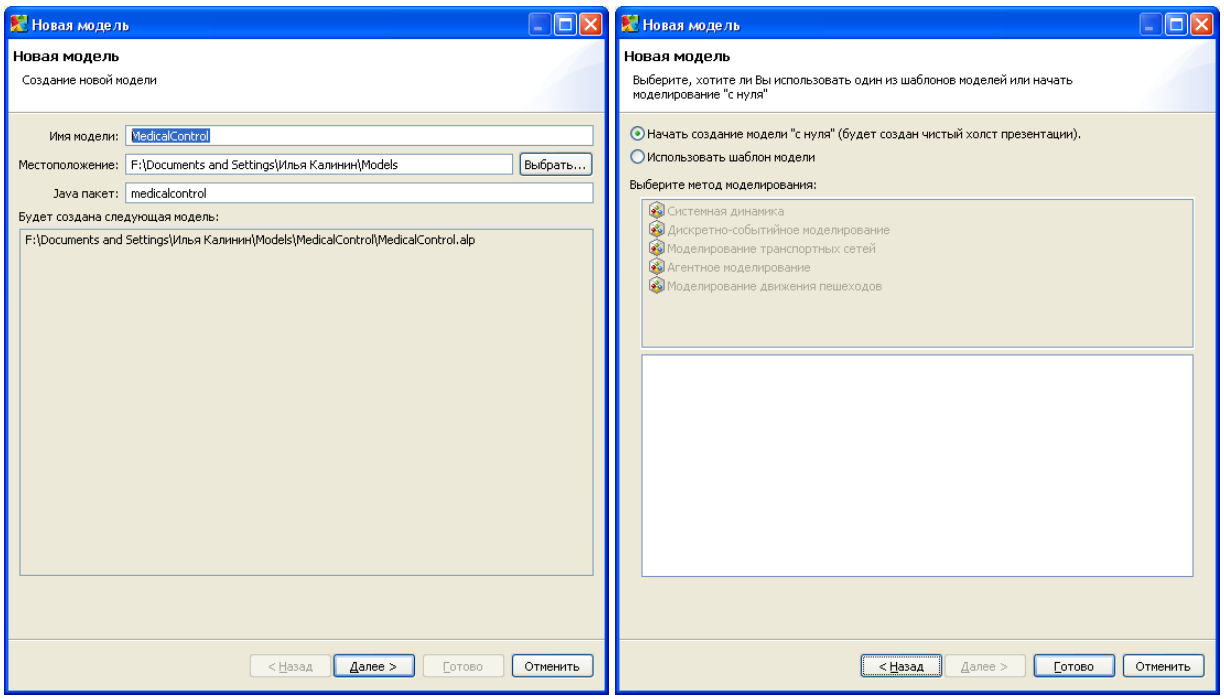
Общее время на проведение осмотра, выделенное лечебно-профилактическим учреждением – 6 часов. Нам известно сколько примерно тратят на осмотр и выдачу документов.

Требуется понять – сколько людей успеют пройти осмотр, не будут ли они где-то скапливаться при имеющихся нормах времени? Сколько в среднем проведет человек на осмотре, сколько максимально?

В модели, с помощью которой мы можем оценить эти параметры, нам не важна траектория движения людей, их взаимодействие между собой и т т.д. Нам важно только время на каждом этапе обслуживания. При этом обслуживание для нас дискретно: нельзя же одновременно осматривать полтора человека.

Воспользуемся для моделирования специальным видом моделей, который называется дискретно-событийными моделями. Основная единица в нем – заявка на обслуживание, которая будет некоторое время обрабатываться на каждом этапе.

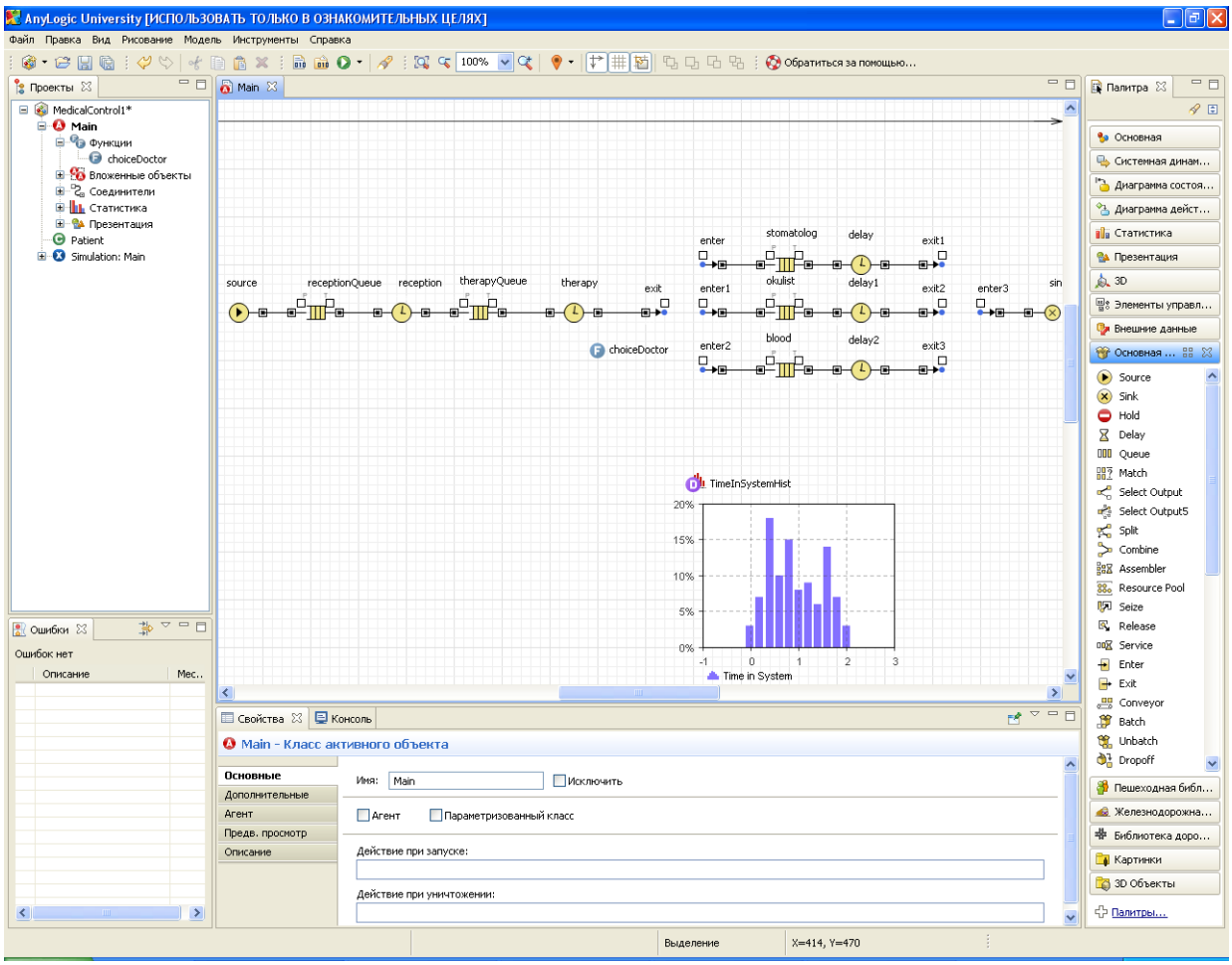
Создадим чистую модель (Файл/Создать/Модель), на втором этапе выберем «Чистый холст»



Для создания модели нам понадобится, как и в предыдущих случаях, разместить необходимые объекты-этапы на рабочем поле и связать их между собой. Для дискретно-событийной модели мы воспользуемся объектами с закладки «Основная библиотека».

Обработка начнется с создания заявки. Для этого мы перенесем из основной библиотеки объект «Source».
Каждый этап будет состоять из двух объектов: «Очередь» (Queue) и «Ожидание» (Delay). Чтобы учесть выбор очереди к специалисту, нам придется вывести заявку из потока (с помощью объекта «Выход» exit, и потом для каждого специалиста принять заявку в объекте Enter). После обработки заявки уничтожаются в объекте Sink.
Там, где обработка идет линейно, свяжем объекты как и в предыдущих моделях.

Для 3 специалистов получится примерно такая схема:



Как и в предыдущем примере, нам понадобится объект для сбора статистики и гистограмма для отображения данных. Основным нашим параметром, статистику которого мы будем собирать специально, будет время , проведенное заявкой в системе – то есть собственно время осмотра. Количество людей мы сможем узнать из чисел на объектах.

Обратите внимание мы не связали очереди специалистов с основным потоком – заявки у нас будут попадать туда с помощью специальной функции.

Для работы модели нам понадобится создать новый класс заявок, как в прошлом примере для пешеходов. Назовем его Patient. Класс создаем на основе класса Entity (заявка) и описываем четыре дополнительных переменных:

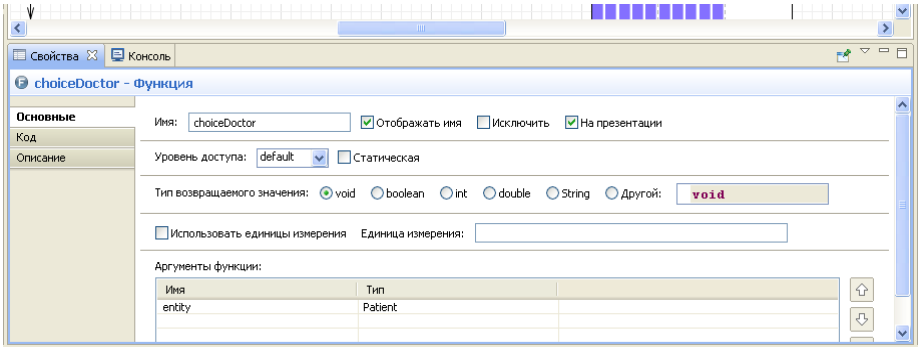
```
boolean stomatolog;
boolean okulist;
boolean blood;
double ArrivalTime;
```

В них мы учтем специалистов, которых человек прошел, и время входа в здание. В конструкторе класса мы напишем начальные значения:

```
public Patient(double ArrivalTime){
        this.ArrivalTime = ArrivalTime;
        this.blood = false;
        this.okulist = false;
        this.stomatolog = false;
  }
```

Этот класс - Patient - мы должны будем указать на всех объектах, через которые проходит заявка.

Кроме этого нам необходима будет специальная функция, которая будет выполнять выбор: в какую из очередей поставить заявку. Для этого из палитры «Основная» перетащим в рабочее окно компонент «Функция», дадим ей название choiceDoctor и создадим один входной параметр: entity, то есть заявку из уже созданного нами класса Patient. Вот как примерно будет выглядеть страница основных свойств функции.



Конечно, функции необходим и программный код, который и будет реализацией. Код запишем в разделе «Код». Реализовать его можно по-разному, автор предлагает такой вариант:

```
if ((entity.blood)&&(entity.stomatolog)&&((entity.okulist)))
        enter3.take(entity);
else {
        int bloodSize = ((!entity.blood)?( blood.size()):(255));
        int okulistSize = ((!entity.okulist)?( okulist.size()):(255));
        int stomatologSize = ((!entity.stomatolog)?( stomatolog.size()):(255));

        if (( bloodSize <= okulistSize)&&(bloodSize <= stomatologSize))
        {
                entity.blood = true;
                enter2.take(entity);
        } else
                if ( okulistSize <= stomatologSize )
                {
                        entity.okulist = true;
                        enter1.take(entity);
                } else
                {
                        entity.stomatolog = true;
                        enter.take(entity);
                }
}
```

Логика работы этой функции такова:

1. Если человек прошел всех специалистов – он свободен.
2. Если нет, то мы выясняем размер очереди для каждого из них. Для тех специалистов, которые уже обработали заявку-обращение, делаем этот размер заведомо неприемлемым.
3. Выбираем самую короткую очередь ставим туда заявку в нее и делаем отметку, что человек этого специалиста прошел – все равно, пока заявка не выйдет из очереди эти данные не используются.

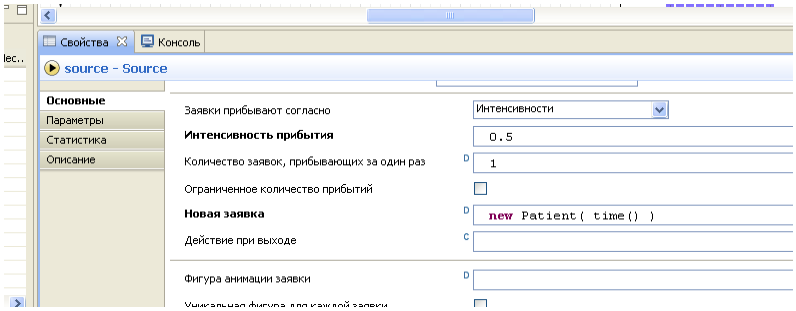
Этой функцией мы будем пользоваться всякий раз, когда нам нужно будет выяснить куда теперь направить заявку. Для этого во всех объектах exit укажем на странице свойств, что при выходе нужно вызвать действие: choiceDoctor(entity);

Основная часть модели готова, но нужно задать еще несколько важных параметров: время задержки. Очевидно, что время задержки на разных этапах будет разным для разных заявок. Это может быть обусловлено массой причин, но сами причины нам не важны – важно как распределится время обработки каждой заявки на каждом этапе. У нас нет полных статистических сведений, но есть средние оценки. Мы будем считать время обработки заявки случайной величиной. Форма ее распределения нам неизвестна, воспользуемся простейшим подходом – предположим, что это «треугольник», с вершиной в области среднего значения.

Предположим, что:
1. Время поиска и выдачи документов в регистратуре – от 3 до 10 минут, в среднем – 5. Зададим время задержки так: triangular(3, 5, 10). Для начала у нас только одно окно выдачи.
2. Терапевт работает быстрее, время осмотра одного человека – от 2 до 5 минут, в среднем – 3.
3. Стоматолог: от 5 до 15, в среднем – 10 минут.
4. Окулист – от 5 до 10, в среднем – 7.
5. Забор крови от 1 до 5 минут, в среднем – 3.

Время на переходы мы игнорируем – предположим, что все происходит на одном этаже и человек видит все очереди.
Нетрудно посчитать, что формально максимум времени на полный осмотр: 45 минут. Теперь посмотрим, как это будет происходить в более точной модели.

Будем считать, что у нас прибывает 1 пациент в 2 минуты, то есть в параметрах источника укажем скорость прибытия 0.5 единицы. Там же проверим, что не забыли указать класс заявки при создании:

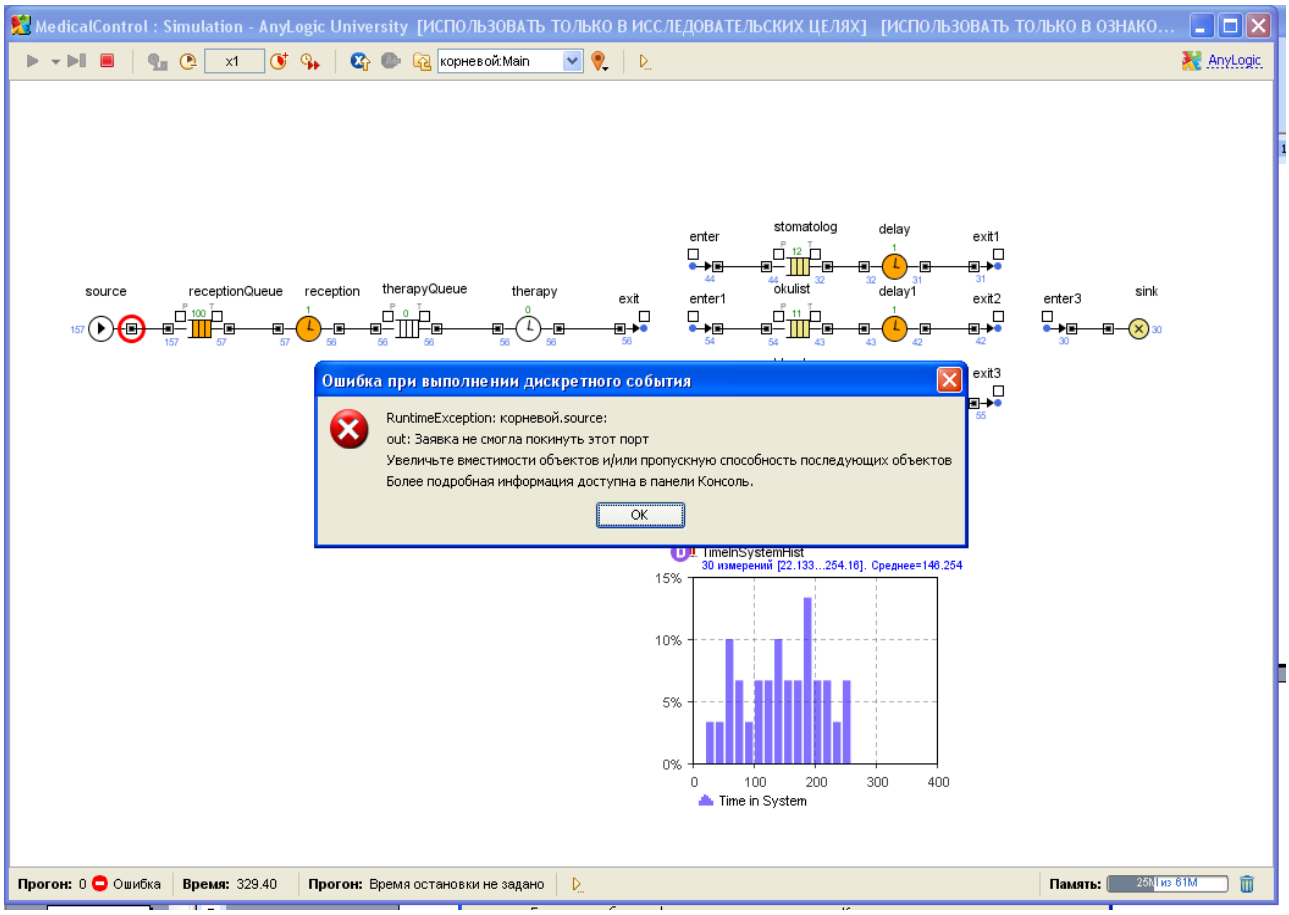


Ограничим время работы модели: выберем ветку Simulation, и в ее свойствах на закладке «Модельное время» укажем конечное время работы – 480 минут.

Теперь можем запустить модель. Сразу можно отметить несколько особенностей:

1. У нас появляется очередь в регистратуру, причем она все время растет.
2. К окулисту и стоматологу всегда стоит очередь, а на анализ крови ее нет.

Результаты исполнения таковы: примерно через 5,5 часа очередь в регистратуру переполняется. За ВСЕ это время осмотр смогли пройти только 30 человек, причем солидная часть из них потратила на это более 3 часов.



Очевидно, что без снижения времени на осмотр и улучшения работы регистратуры такая система не сможет фактически исполнить свои задачи.

Задания для самостоятельной работы:

1. Почему на анализ крови практически не было очереди?
2. Как отразить в параметрах модели тот факт, что коридор где ждут своей очереди на осмотр – не безразмерный?
3. Что произойдет, если в очереди будут появляться заявки с длительным временем обслуживания – например, любители поговорить о здоровье? Какие параметры нужно будет изменить, чтобы это выяснить?
4. Что произойдет в системе, если за счет внедрения электронной карточки врачи смогут уменьшить время на заполнение документов на 2 минуты каждый?
5. Как повлияет на такую систему появление второго сотрудника в регистратуре? Вы можете реализовать это в модели с помощью объектов Service и ResourcePool (они заменят очередь и обслуживание в регистратуре) основной библиотеки. Пример использования – модель Bank из обучающих моделей AnyLogic.

Как видно даже из очень небольшого материала, метод имитационного моделирования позволяет решать огромный спектр практических задач. С методической точки зрения ценность его еще и в том, что он ярко демонстрирует саму суть понятия "моделирование" не требуя высокой математической и технической подготовки.
Применение современной среды моделирования позволяет не только говорить о методе на занятиях, но и провести массу эффектных и полезных проектных работ, темы для которых можно найти в самых разных школьных предметах, собственном окружении, личной практике.

Литература:

1. Качала В.В. Основы теории систем и системного анализа. – М.: Горячая линия-Телеком, 2007 г. – 216 с.
2. Форрестер Дж. Основы кибернетики предприятия (индустриальная динамика), М.: Прогресс, 1971 г. – 339 с.
3. Шеннон Р. Имитационное моделирование систем – искусство и наука, М.: Мир, 1978 г. – 418 с.
4. Медоуз Д.Х. Азбука системного мышления – М.: Бином. Лаборатория знаний, 2010 г – 343 с.
5. Арнольд В.И. Теория катастроф.—3-е изд., доп.—М.: Наука. Гл. ред. физ.-мат. лит., 1990.—С. 128.
6. Калинин И.А., Самылкина Н.Н. Информатика. 10 класс. Углубленный уровень. Учебник., М.:Бином, 2013