

**МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ  
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)**

---

**Кафедра «Автоматизированные системы управления»**

**К.Н. МЕЗЕНЦЕВ**

**УЧЕБНОЕ ПОСОБИЕ  
«Моделирование систем в среде  
AnyLogic 6.4.1»**

**Часть 2**

**Москва 2011**

УДК 004.438

ББК 32.973.26-018.1

К.Н. Мезенцев. Учебное пособие «Моделирование систем в среде AnyLogic 6.4.1». Часть 2 /Под редакцией Заслуженного деятеля науки РФ, д.т.н., профессора А.Б.Николаева. МАДИ. — М.: 2011. 103 с.

Учебное пособие предназначено для студентов, изучающих дисциплину «Моделирование систем», а также для магистров и аспирантов всех форм обучения для создания и использования имитационных моделей с использованием специализированной системы моделирования AnyLogic. Материал учебного пособия также может помочь при работе над дипломным проектом или при проведении диссертационных исследований аспирантами.

© Московский автомобильно-дорожный  
государственный технический университет (МАДИ), 2011

## Содержание

Введение .....	5
1. Моделирование динамических систем .....	7
1.1. Исследование динамической модели Лоренца .....	7
1.2. Изучение чувствительности модели Лоренца .....	9
1.3. Типовые звенья.....	11
1.3.1. Моделирование работы интегрирующего звена .....	11
1.3.2. Исследование апериодического звена и колебательного звена .....	15
Апериодическое звено .....	15
1.4. Модель следящего гидропривода .....	19
Контрольные вопросы.....	21
2. Построение моделей систем массового обслуживания .....	21
2.1. Моделирование системы обслуживания клиентов .....	21
2.2. Анимация модели .....	23
2.3. Размещения графиков.....	25
2.4. Моделирование двухканальной СМО.....	27
2.5. Определение параметров СМО.....	32
Самостоятельные задания .....	37
Контрольные вопросы.....	37
3. Исследование систем массового обслуживания.....	38
3.1. Задача Эрланга.....	38
3.1.1. Определение расходов на обслуживание телефонных вызовов.....	42
3.1.2. Определение оптимального числа каналов .....	45
3.2. Система массового обслуживания с отказами .....	49
3.3. Задания для самостоятельной работы .....	54
3.3.1. Разработка двухканальной СМО.....	54
3.3.2. Модель трехканальной СМО .....	56
3.3.3. Модель трехканальной СМО без очередей.....	60
Контрольные вопросы.....	62
4. Моделирование сетей.....	63
4.1. Модель работы станции автосервиса .....	63
Задания для самостоятельной работы.....	76
4.2. Модель вестибюль метро.....	77
Самостоятельное задание.....	85
Контрольные вопросы.....	86
Приложение №1. ....	88
Элементы библиотеки Enterprise Library.....	88
Приложение №2. ....	99
Элементы библиотеки Pedestrian Library.....	99
Приложение №3. ....	105

Сбор статистики .....	105
Приложение №4. ....	107
Функции законов распределения .....	107
Список литературы .....	108

## **Введение**

Вторая часть практикума позволяет студентам освоить в среде AnyLogic основные приемы моделирования динамического поведения систем.

В первой части практикума рассматривается технология построения динамических моделей для линейных систем, поведение которых может быть описано линейными дифференциальными уравнениями в нормальной форме Коши. Изучается технология построения модели с экспериментом, который содержит интерфейс для вариации исследуемых параметров.

Вторая глава содержит необходимые сведения для построения моделей систем массового обслуживания (СМО) на основе библиотеки Enterprise Library, изучаются методы анимации построенной модели в Any Logic. Приводятся сведения позволяющие оценить качество работы СМО.

В третьей главе рассматривается технология проведения компьютерного оптимизационного эксперимента на примере системы массового обслуживания телефонных вызовов и изучается методика построения систем массового обслуживания с вытеснением заявок.

В четвертой главе содержатся сведения позволяющие использовать дополнительные элементы библиотеки Enterprise Library для моделирования СМО использующих статические и динамические ресурсы. В этой главе рассматриваются основные элементы библиотеки Pedestrian Library, позволяющие выполнить моделирование пешеходных СМО.

Пособие содержит четыре приложения, в которых описаны методы и свойства объектов библиотек Enterprise Library и

Pedestrian Library, необходимые для построения моделей, которые рассмотрены в пособии.

Так же в приложении приводятся сведения о технологии сбора статистики при моделировании СМО и генерации законов распределения с помощью функций AnyLogic.

В пособие включены дополнительные задания, позволяющие выполнить контроль знаний и умений, полученных студентами в процессе освоения методов моделирования в среде AnyLogic.

## 1. Моделирование динамических систем

### 1.1. Исследование динамической модели Лоренца

Динамическая модель Лоренца обычно используется для исследования неустойчивых процессов протекающих в атмосфере Земли.

В модели выделяют три переменные. Переменная  $X$  задает интенсивность конвекции, переменная  $Y$  – разность температур восходящего и нисходящего потоков, переменная  $Z$  – изменение во времени вертикальной температуры в некоторой точке пространства.

Характерной особенностью модели является наличие трех петель обратных связей. Модель задается системой дифференциальных уравнений:

$$\begin{aligned}\frac{dX}{dt} &= S \times (Y - X) \\ \frac{dY}{dt} &= (R - Z) \times X - Y \\ \frac{dZ}{dt} &= X \times Y - B \times Z\end{aligned}$$

Здесь  $S$ ,  $R$ ,  $B$  – факторы, влияющие на модель.

При некоторых начальных условиях модель описывает детерминированный хаос (аттрактор Лоренца).

Модель сильно чувствительна к начальным условиям. Такое свойство модели называется эффектом бабочки.

### Построение модели

Используйте палитру инструментов «Системная динамика» и постройте модель в соответствии с системой дифференциальных уравнений, так как это показано на рисунке 1.1

Значения параметров должны соответствовать таблице 1.1, начальные значения накопителей даны в таблице 1.2.

Далее добавьте в модель временной график для отображения изменения значения накопителя  $X(t)$ .

Параметры модели. Таблица 1.1

№	Параметр	Значение
1	R	28
2	S	10
3	B	8.0/3.0

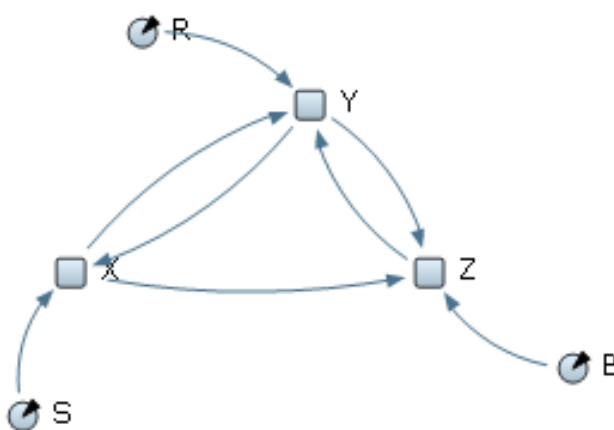


Рис.1.1. Модель Лоренца

Начальные значения накопителей. Таблица 1.2

№	Накопитель	Значение
1	X	10
2	Y	0
3	Z	10

Добавьте в модель фазовый график, показывающий изменение в накопителе  $X$  в зависимости от изменений в накопителе  $Z$ .

Настройте эксперимент модели:

- Единицы модельного времени – часы;
- Конечное время-50.



Выполните запуск модели. Вид работающей модели должен соответствовать рисунку 1.2.

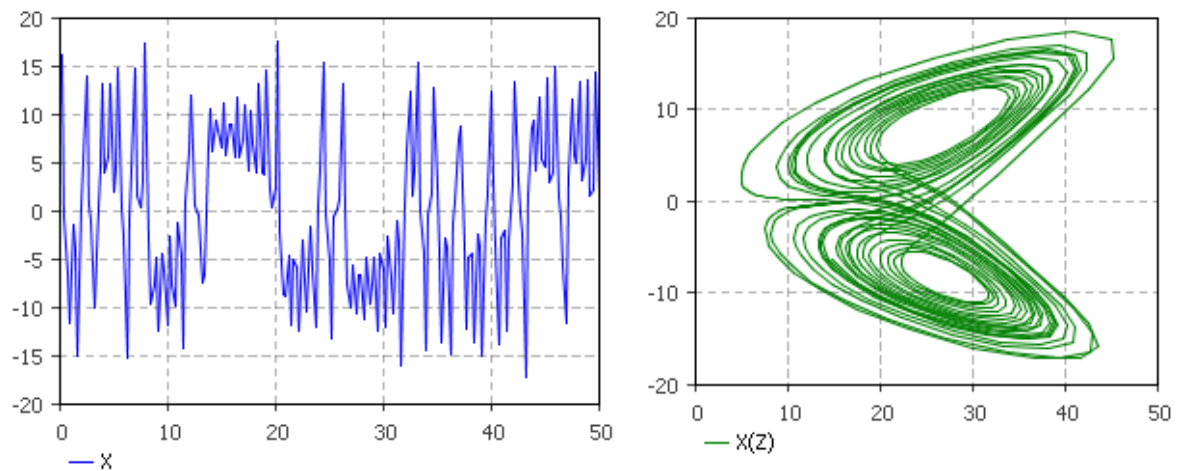


Рис.1.2. Графики процессов модели

## 1.2. Изучение чувствительности модели Лоренца

Создайте новую модель. В модель разместите динамическую модель Лоренца такую же, как в предыдущем примере. Значение параметров и исходных данных соответствуют таблицам 1.1 и 1.2. Получите копию этой модели, так как это показано на рисунке 1.3.

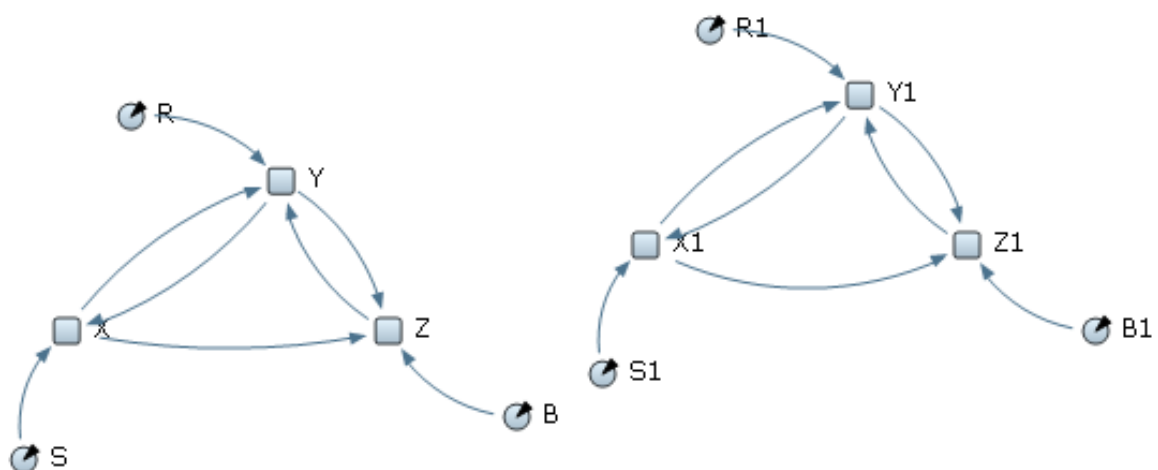


Рис.1.3. Две модели Лоренца

Во второй модели измените, начальное значение для накопителя  $z_1$ , так чтобы оно отличалось от значения накопителя  $z$  в десятом знаке: 10.0000000001.

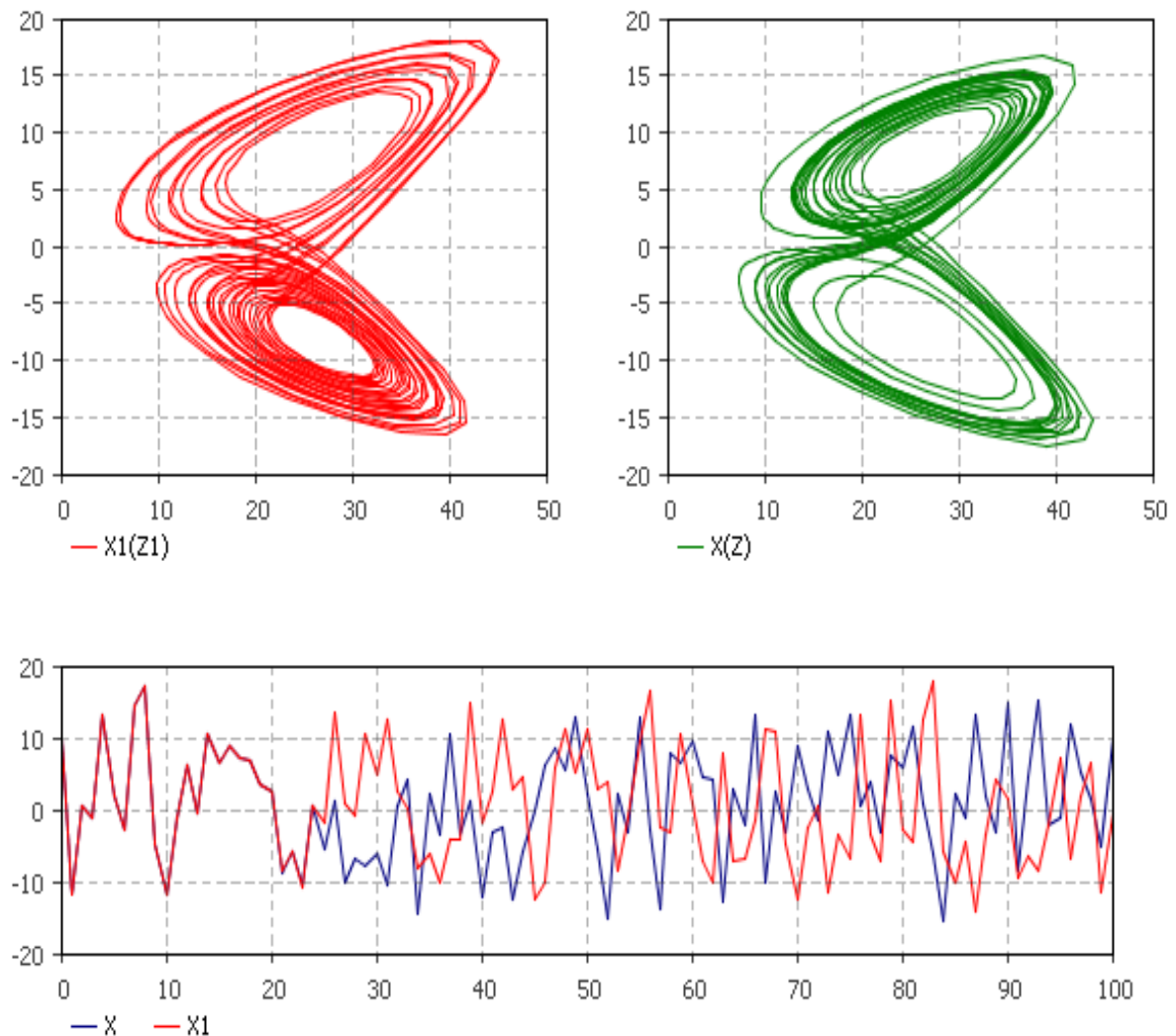


Рис.1.4. Сравнение работы двух моделей

Задайте конечное время работы модели равное 100.

Выведите фазовые графики, отражающие изменения  $x_1(z_1)$  и  $x(z)$ . Выведите временной график для показа изменения  $x_1(t)$  и  $x_2(t)$ .

Выполните тестирование модели, графики процессов модели показаны на рисунке 1.4. Для контроля установите начальное значение накопителя  $z_1 = z = 10.0000000000$ . Выполните прогон модели, при правильном ее построении графики процессов должны быть идентичны.

### 1.3. Типовые звенья

Любая техническая система может быть представлена в виде линейной динамической модели, состоящей из типовых звеньев.

#### 1.3.1. Моделирование работы интегрирующего звена

Передаточная функция звена имеет следующий вид:

$$\frac{y(s)}{u(s)} = \frac{K}{Ts}$$

Здесь:

$K$  – Коэффициент усиления звена;

$T$  – Постоянная времени звена;

$y(s)$  – выходной сигнал;

$u(s)$  – входной сигнал;

$s$  – оператор Лапласа.

Запишем уравнение, связывающее выходной сигнал с входным:

$$y(s)sT = u(s) \cdot K$$

Заменяя оператор  $s$  на производную, получим дифференциальное уравнение модели звена в виде:

$$y'T = u \cdot K$$

После подстановки  $y = x$  в окончательном виде дифференциальное уравнение модели примет вид:

$$\frac{dx}{dt} = \frac{u \cdot K}{T} \quad (1.1)$$

Создайте новую модель. Используя вкладку «Системная динамика» постройте модель колебательного звена, которая отвечает уравнению 1.1. Вид модели должен соответствовать рисунку 1.5.

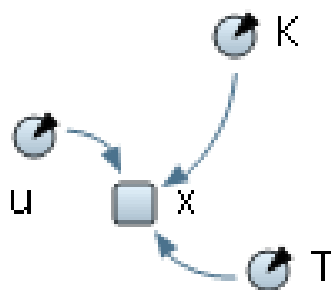


Рис.1.5. Модель колебательного звена

Присвойте значения параметрам  $K$  и  $T$  по умолчанию равное нулю, а параметру  $u$  значение равное единице.

Для отображения процесса преобразования сигнала  $u$  в выходной сигнал  $x$  постройте временной график. Задайте временной диапазон равным 20. Масштаб вертикальной шкалы задайте постоянным, предельное значение задайте равным 200.

Настройте эксперимент модели. Единицы измерения модельного времени задайте как секунды. Модель должна останавливаться при достижении времени 20 секунд.

Для исследования влияния постоянной времени  $T$  и коэффициента усиления звена  $K$  на работу модели настройте презентацию модели. Для этого на дереве проекта откройте презентацию эксперимента, так как это показано на рисунке 1.6.

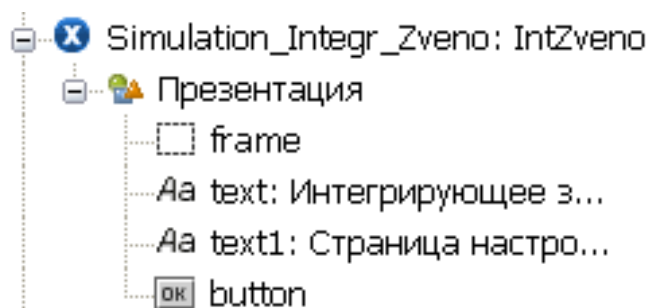


Рис.1.6. Открытие узла «Презентация»

Выполните двойной щелчок на элементе `frame`. В результате откроется окно для редактирования стартового окна запуска

модели, измените, текст на «Интегрирующее звено» так, как это показано на рисунке 1.7.

Разместите две простых переменных типа `double` и назовите их `K` и `T`, соответственно. Начальное значение `K` задайте равным 1, а `T` равным 0.95. Размещение переменных должно соответствовать рисунку 1.7.

Далее в окно презентации разместите два элемента типа "бегунок". Первый элемент служит для изменения коэффициента усиления в интервале от 1 до 10. Он должен быть связан с переменной `K`. Второй бегунок должен быть связан с переменной `T` и позволяет изменять ее в интервале от 0.01 до 1.

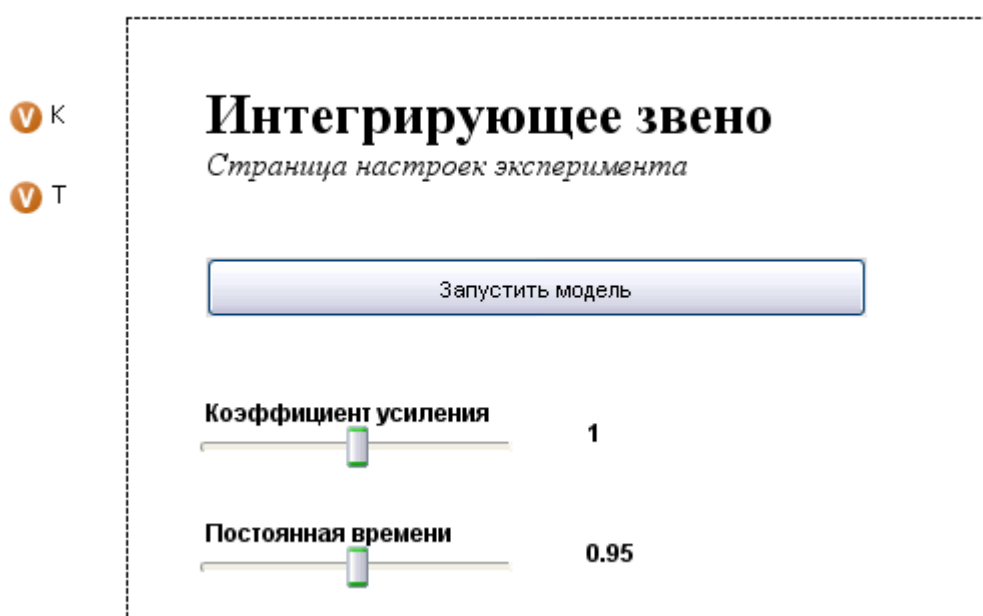


Рис.1.7. Окно презентации модели

Используя элемент «Текст» палитры «Презентация», задайте подпись к ползункам, так как это показано на рисунке 1.7.

Рядом с ползунками разместите два текстовых элемента для отображения текущего значения коэффициента усиления и постоянной времени.

Что бы отобразить значения в поле «Действие» соответствующего ползунка нужно ввести код Java записи текущего значения в элемент «Текст». Округлите выводимые значения до второго знака после запятой.

Конечный вид окна презентации должен соответствовать рисунку 1.7.

Что бы сделанный выбор значений параметров был передан в модель нужно активизировать на дереве проекта узел Simulation(эксперимент) модели и на вкладке «Основные» в текстовых полях параметров ввести имена переменных из окна презентации. В данной модели они совпадают с параметрами (см. рисунок 1.8).

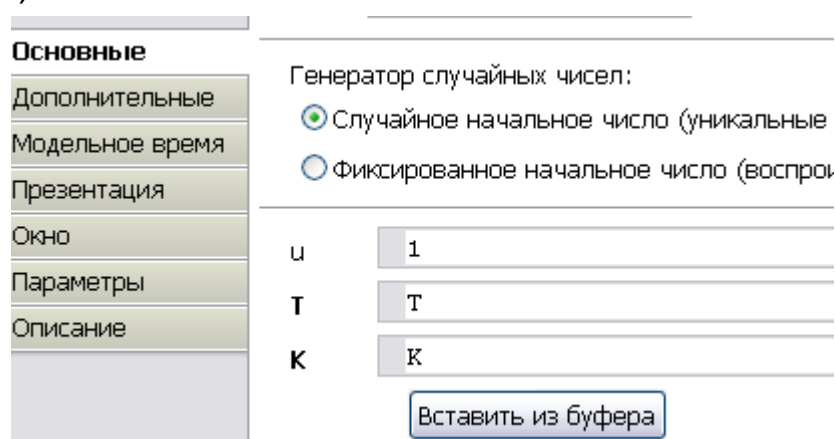


Рис.1.8. Связывание настроек с параметрами модели

Протестируем созданную модель. На рисунке 1.9 показано стартовое окно с выбранными значениями коэффициента усиления и постоянной времени.

## Интегрирующее звено

*Страница настроек эксперимента*

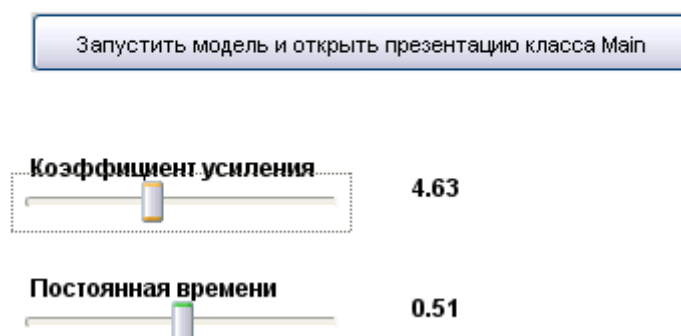


Рис.1.9. Выбор параметров

Вид работающей модели показан на рисунке 1.10. Видно, что в модель переданы выбранные настройки.

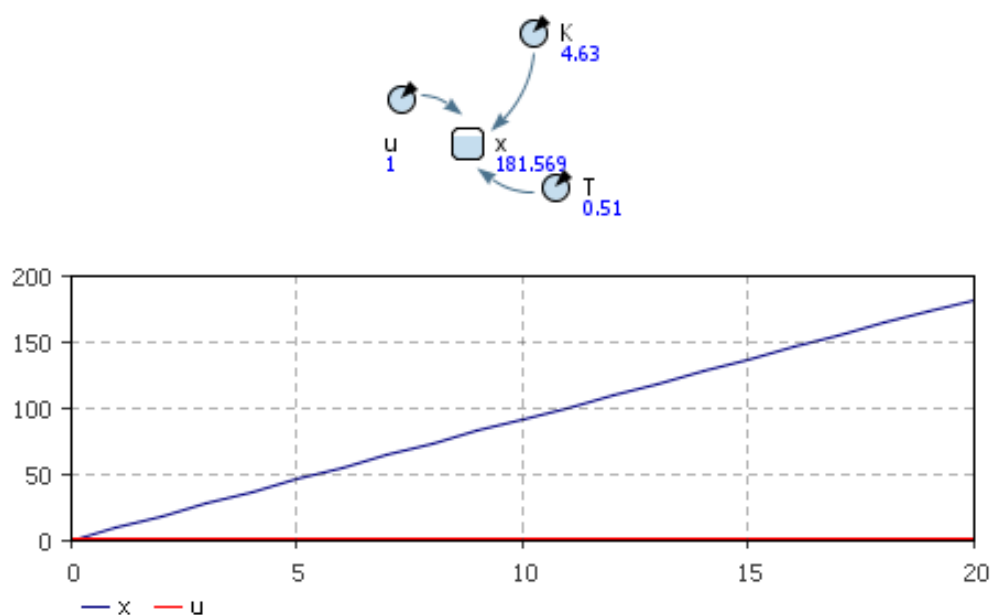


Рис.1.10. Модель интегрирующего звена.

### 1.3.2. Исследование апериодического звена и колебательного звена

#### Апериодическое звено

Апериодическое звено описывается дифференциальным уравнением вида:

$$\frac{dx}{dt} = \frac{1}{T}(u \cdot K - x)$$

Разместите в созданную ранее модель интегрирующего звена новый активный класс и постройте модель апериодического звена. Исследуйте, как влияет на работу модели изменение коэффициента усиления в диапазоне от 1 до 15 и значение постоянной времени при ее изменении в диапазоне от 0.001 до 0.85.

Характер протекания процесса в звене показан на рисунке 1.11.

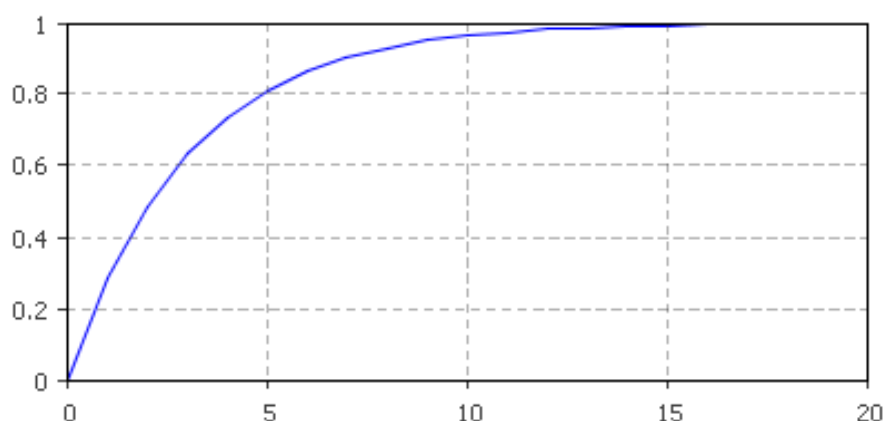


Рис.1.11. Динамический процесс в аperiodическом звене  
Примите значения по умолчанию:  $K = 2$ ;  $T = 0.5$ .

Значение задающего сигнала  $u$  задайте равным единице.

Чтобы протестировать созданную модель нужно добавить новый эксперимент в состав модели AnyLogic, используя команду контекстного меню на дереве проект, так как это показано на рисунке. 1.12.

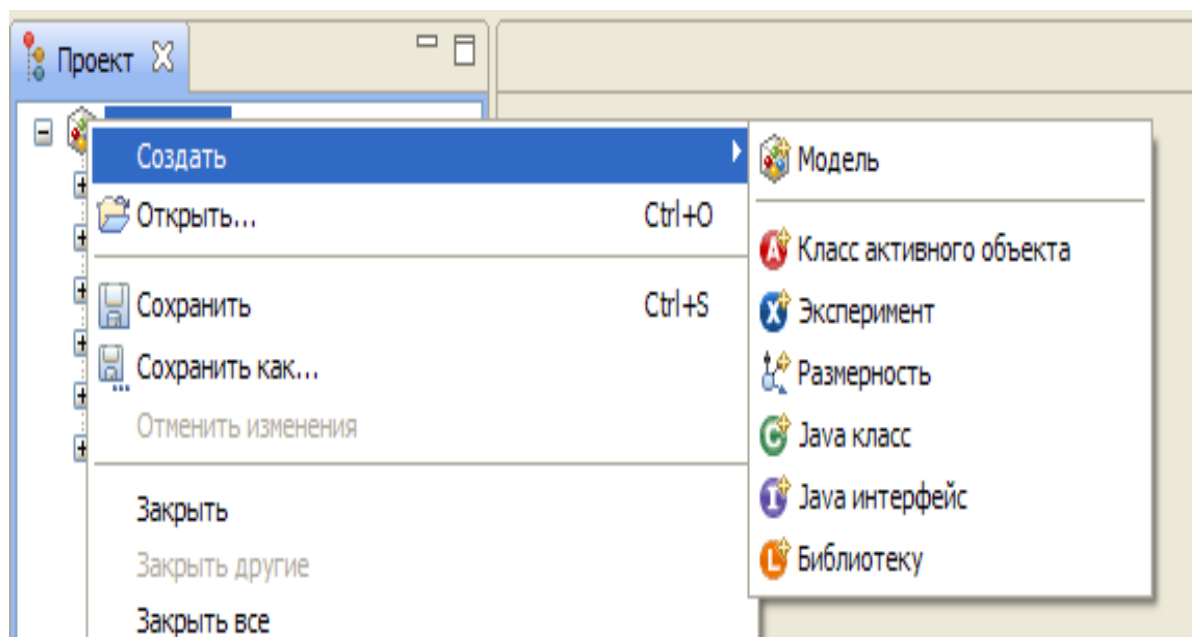


Рис. 1.12. Создание нового эксперимента.

После выбора команды Эксперимент будет открыт диалог для настройки нового эксперимента, показанный на рисунке 1.13.



## Эксперимент

Выберите тип эксперимента, задайте имя и выберите корневой класс модели.

Имя: Simulation\_ClassAperZveno

Корневой класс модели: AperZveno

Тип эксперимента:


- ☒ Простой эксперимент
- ☐ Оптимизация
- ☐ Варьирование параметров
- ☐ Сравнение "прогнозов"
- ☐ Монте-Карло
- ☐ Анализ чувствительности
- ☐ Калибровка
- ☐ Нестандартный

Запускает модель с заданными значениями виртуального и реального времени, анимации

Рис.1.13. Создание нового эксперимента

В свойстве «Имя» нужно задать новое наименование эксперименту, а в свойстве «Корневой класс модели», нужно задать имя активного - корневого класса для данной модели. Тип эксперимента выбирается как «Простой эксперимент».

Проверить правильность настройки эксперимента на корневой класс, можно после выбора его на дереве проекта. В окне свойств используется вкладка «Основные».

Для запуска нового эксперимента нужно при выборе кнопки пуска  меню AnyLogic выбрать эксперимент, который соответствует активному классу модели.

## Колебательное звено

Колебательное звено описывается системой дифференциальных уравнений:

$$\frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = \frac{u \cdot K}{T^2} - \frac{2 \cdot \zeta}{T} x_2 - \frac{x_1}{T^2}$$

Здесь  $\zeta$  - коэффициент демпфирования звена.

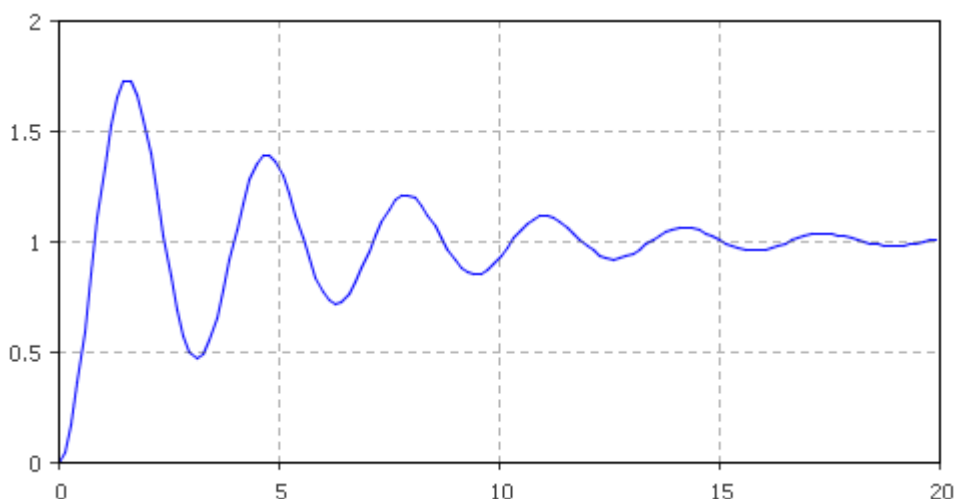


Рис.1.14. Процесс в колебательном звене

Добавьте в модель еще один активный класс и постройте модель колебательного звена. Исследуйте, как на работу звена влияет изменения таких параметров как коэффициент усиления, постоянная времени и коэффициент демпфирования при их изменении в диапазонах:

$$K \in [1..10]$$

$$T \in [0.01..1,5]$$

$$\zeta \in [0..1]$$

Примите следующие начальные значения:

$$K=2;$$

$$T=0.75;$$

$$\zeta=0.15.$$

Значение задающего сигнала  $u$  задайте равным единице.

На рисунке 1.14 показан характер протекания динамического процесса в колебательном звене.

## 1.4. Модель следящего гидропривода

Требуется построить модель следящего гидропривода и определить предельное значение коэффициента усиления регулятора по рассогласованию между задающим сигналом и выходным перемещением звена гидродвигателя.

Общий вид модели показан на рисунке 1.15.

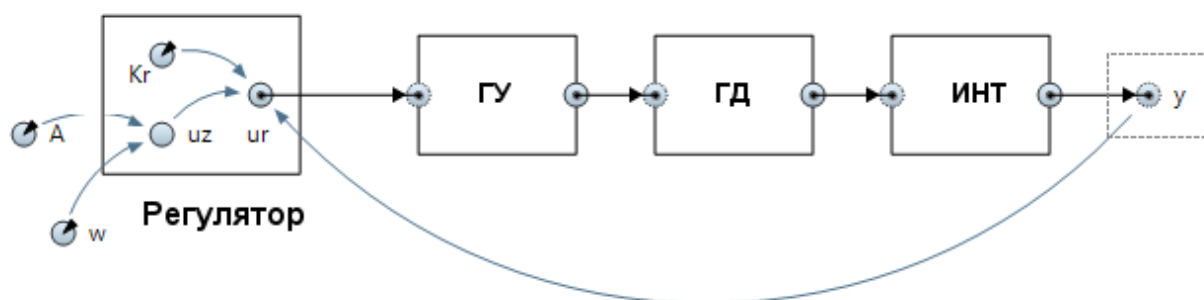


Рис.1.15. Модель следящего привода

В модели приняты следующие условные обозначения:

$ur$ - выходной сигнал регулятора;

$uz$ - задающий входной сигнал;

$y$  – перемещение выходного звена гидродвигателя.

Задающий сигнал задается по синусоидальному закону и задается по формуле  $uz = A \cdot \sin(\omega \cdot t)$  здесь  $t$  модельное время, измеряемое в секундах. Амплитуда равна 0,3, а частота 0.1.

Регулятор представляет собой пропорциональный регулятор, выходной, сигнал которого формируется по формуле:

$$ur = Kr \cdot (uz - y)$$

Здесь  $Kr$  – коэффициент усиления регулятора.

Гидропривод состоит из трех элементов:

ГУ – гидроусилитель, его работа описывается апериодическим звеном;

ГД- гидродвигатель поступательного действия, его работа описывается колебательным звеном;

ИНТ- интегратор, играет роль преобразователя для получения перемещения выходного звена привода, описывается интегрирующим звеном.

Параметры динамических звеньев приведены в таблице 1.3.

Параметры модели привода. Таблица 1.3.

Тип элемента	Постоянная времени	Коэффициент усиления	Прочие параметры
ГУ	0,5	0,75	Нет
ГД	1,5	0,25	$\zeta=0,75$
ИНТ	1	1	Нет

Расположите на модели временной график для вывода значения задающего сигнала  $u_r$  и выходного сигнала  $y$ .

Время моделирования составляет 100 секунд. Вид работающей модели и графиков динамических процессов приводится на рисунке 1.16. Коэффициент усиления регулятора выбран 1,5.

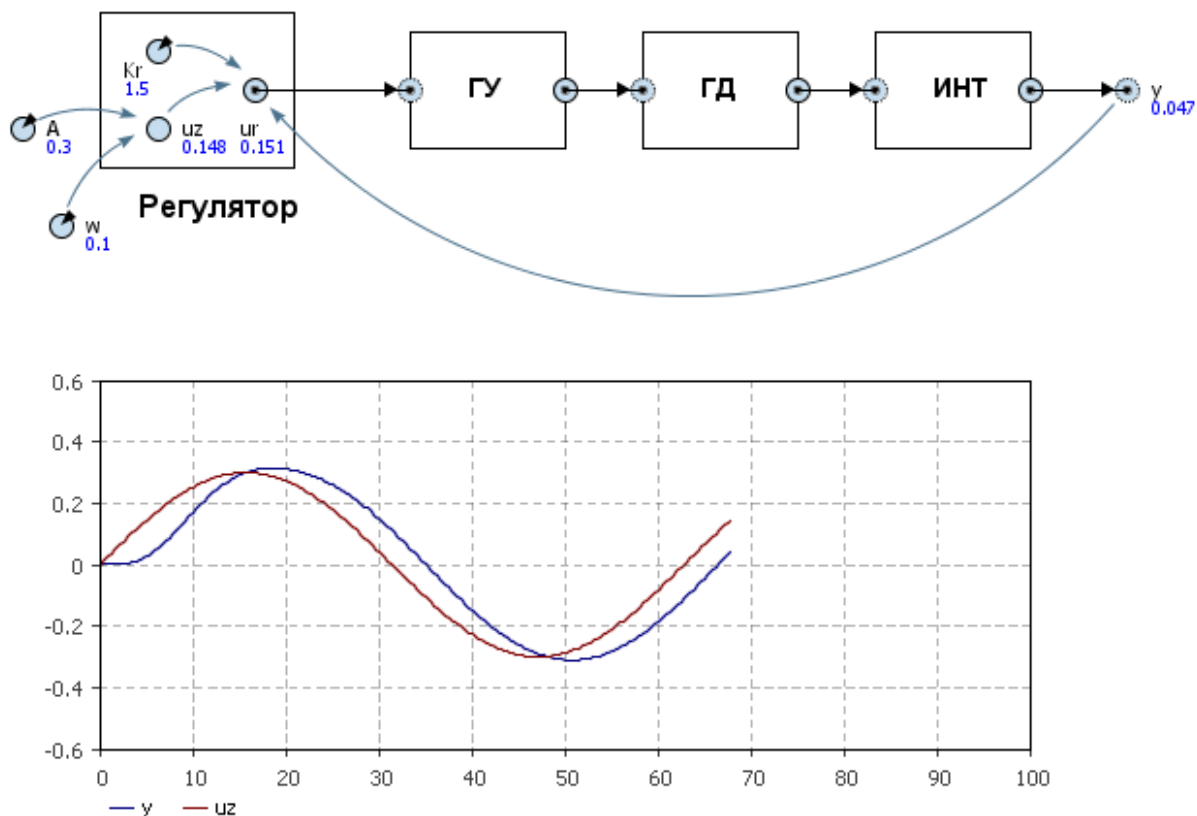


Рис.1.16. Процесс отработки приводом входного воздействия

Исследуйте, как влияет на работу привода величина коэффициента усиления регулятора  $K_r$  при изменении его от 0 до 5. Создайте эксперимент с интерфейсом для изменения коэффициента усиления с помощью ползунка.

### **Контрольные вопросы**

1. Перечислите основные этапы построения динамических моделей в среде AnyLogic.
2. Дайте характеристику модели Лоренца.
3. Для чего используются фазовые графики при представлении результатов моделирования?
4. Перечислите типовые звенья используемые при моделировании линейных динамических систем.
5. Что представляет собой переходный процесс в динамических системах?
6. Дайте характеристику типовым звеньям динамических систем с точки зрения протекания в них переходных процессов.
7. Как строится интерфейс эксперимента AnyLogic для связи с параметрами модели?
8. Как выполняется соединение подсистем при построении динамической модели в AnyLogic?
9. В чем заключается принцип работы динамической системы с отрицательной обратной связью?

## **2. Построение моделей систем массового обслуживания**

### **2.1. Моделирование системы обслуживания клиентов**

Постановка задачи. В банковский офис обращаются клиенты. Офис представляет собой автоматизированный пункт обслуживания, в котором установлен банкомат. Банкомат обслуживает одновременно одного клиента. Клиенты прибывают с интенсивностью  $\lambda=0,67$ . Одновременно в офисе может находиться не

более 15 клиентов. Интервал времени работы банкомата подчиняется треугольному закону распределения с параметрами  $x_{\min}=0.8$ ,  $x_{\max}=1.3$  предпочтительное значение 1.

### Построение модели

Модель строится с «нуля». Банковский офис представляет собой систему массового обслуживания (СМО). Построение модели такой системы выполняется с помощью элементов библиотеки Enterprise Library (см. Приложение №1). В приложении №4 приводятся сведения о генерации законов распределения при моделировании СМО. Для построения СМО используются элементы:

- Source – источник заявок.
- Queue – очередь ожидающих обслуживания заявок.
- Delay – Элемент моделирующий узел обслуживания.
- Sink – Элемент принимающий отработанные заявки.

Вид модели СМО банковского офиса показан на рисунке 2.1.

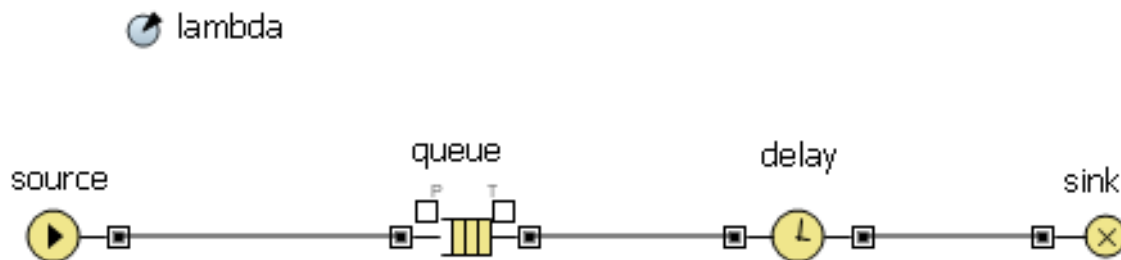


Рис. 2.1. Модель офиса

### Источник заявок

Заявки – клиенты офиса пребывают с интенсивностью  $\lambda=0.67$ .

Источник заявок обладает следующими настройками:

- Заявки пребывают согласно интенсивности.

- Интенсивность прибытия равна  $\lambda$ .
- Количество заявок пребывающих за один раз равно единице.

### Очередь

Этот элемент характеризуется параметрами:

- Вместимость очереди равна 15.
- Включить сбор статистики – да.

### Узел обслуживания

Параметры элемента:

- Задержка задается явно.
- Время задержки равно: `triangular(0.8, 1.3, 1)`.
- Вместимость узла – один клиент.
- Включить сбор статистики- да.

Элемент, принимающий заявки обладает параметрами настройки по умолчанию.

Настройте эксперимент модели:

- Модельное время – минуты.
- Время остановки модели не задано.
- Режим выполнения задайте равным 8.

На рисунке 2.2. показан вид активной модели

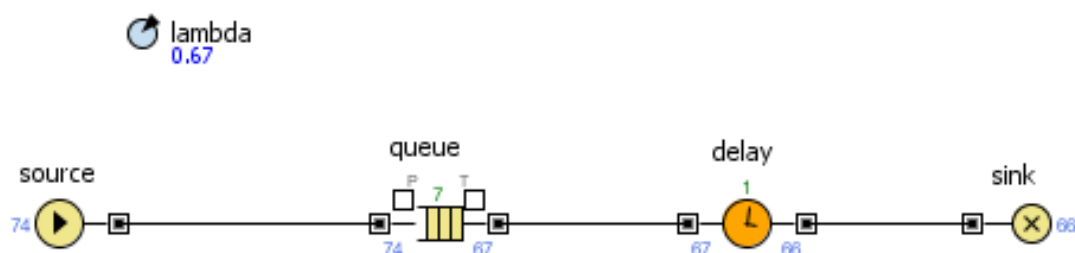


Рис.2.2. Вид работающей модели

## 2.2. Анимация модели

Покажем процесс обслуживания клиентов в виде анимации очереди, ведущей к банкомату, так как это показано на рисунке 2.3.

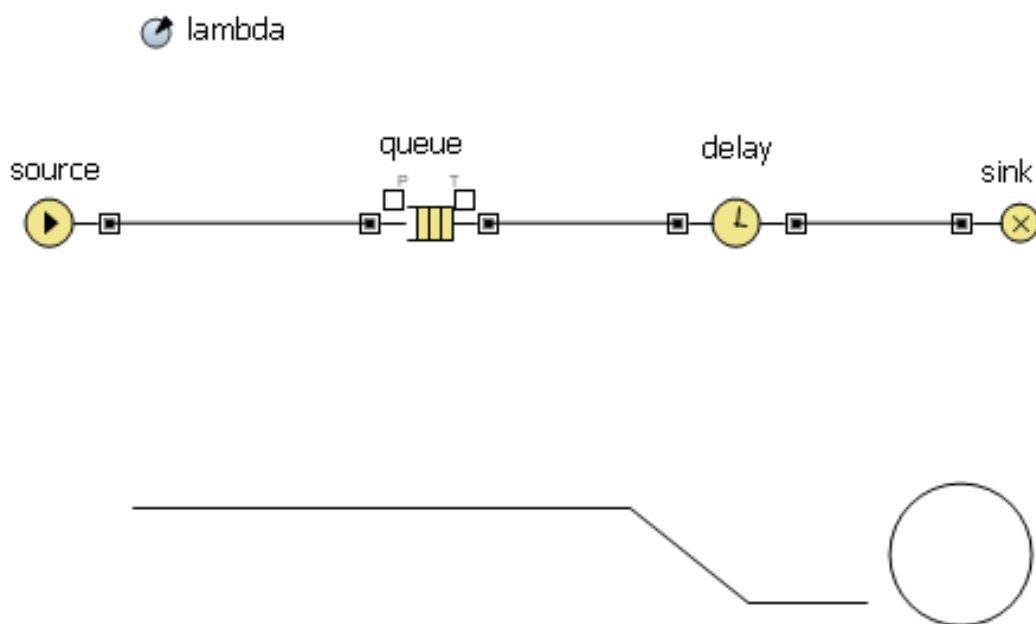



Рис.2.3. Анимация очереди

Банкомат представим в виде окружности. Когда клиент находится в банкомате, окружность будет окрашена в красный цвет, при свободном банкомате окружность закрашивается в зеленый цвет.

С помощью элемента «Овал» палитры презентация разместите окружность и присвойте ей имя `ServicePoint`. Цвет заливки должность изменяться динамически:

```
delay.size() > 0 ? Color.red: Color.green
```

Здесь `size()` – метод объекта `delay`, который возвращает количество заявок-клиентов в приборе обслуживания.

Для отображения очереди следует нарисовать ломаную линию (см. рисунок 2.3) используя элемент «Ломанная» из палитры «Презентация». Режим рисования включается после выполнения двойного щелчка по пиктограмме .

Рисование ломанной нужно выполнять по направлению движения клиентов к банкомату: слева на право. Ломанной присвойте имя `GoToService`.



После создания элементов презентации нужно выполнить ряд настроек модели.

Откройте окно свойств элемента очередь (queue) и на вкладке «Основные» задайте настройки так, как это показано на рисунке 2.4.

Фигура анимации	GoToService
Тип анимации	Путь <input type="button" value="v"/>
Направление анимации	<input checked="" type="radio"/> Вперед <input type="radio"/> Назад

Рис.2.4. Настройка очереди

Откройте прибор обслуживания – элемент delay и настройте на вкладке «Основные», свойства анимации:

Фигура анимации: ServicePoint

Тип анимации: Одиночная

Установите режим исполнения равным 4 и протестируйте модель. На рисунке 2.5. Показан вид работающей модели.

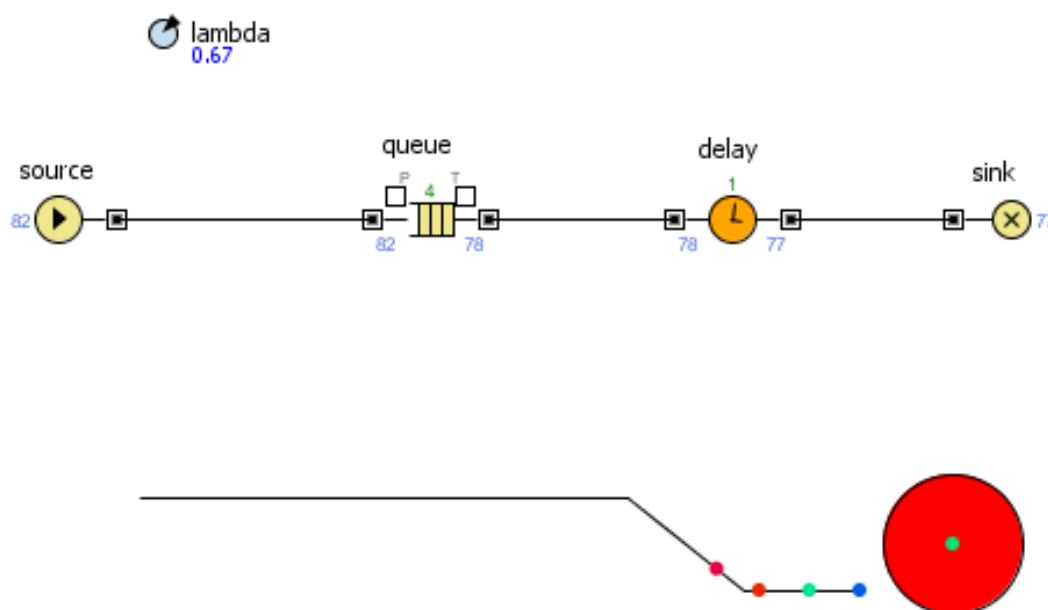


Рис.2.5. Модель с анимацией

## 2.3. Размещения графиков

Что бы представить процесс загрузки прибора обслуживания и очереди разместим два графика. Первый график отображает

среднее значение клиентов в очереди, а второй среднее значение числа обслуженных клиентов в приборе обслуживания.

Для размещения графиков нужно использовать палитру «Статистика» и элемент «Столбиковая диаграмма». Разместите две диаграммы (см. рисунок 2.6).

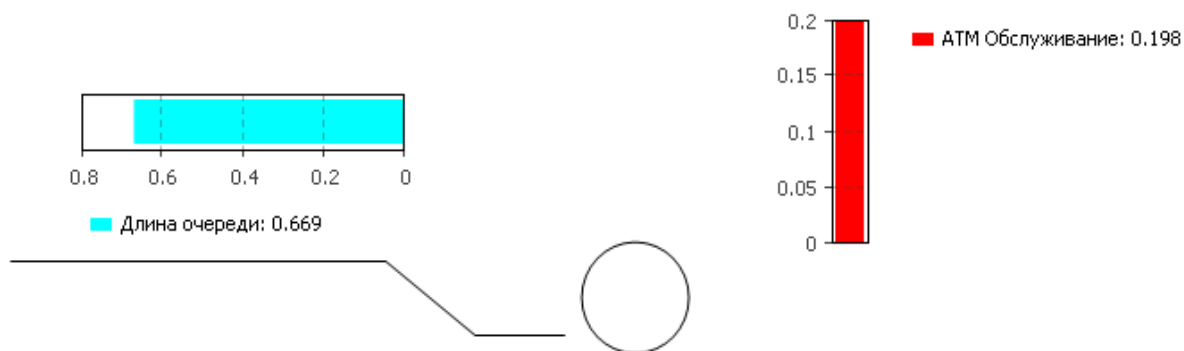


Рис.2.6. Диаграммы загрузки

Первую поместите над очередью. Добавьте элемент данных. Задайте подпись «Длина очереди». Выберите цвет, а затем в качестве значения задайте выражение:

`queue.statsSize.mean()`

Здесь метод `mean()` – возвращает среднюю длину очереди (см. Приложение №3).

При вводе выражения можно использовать помощник AnyLogic. Для этого следует нажать комбинацию клавиш CTRL + SPACE.

После ввода выражения нужно сменить ориентацию графика. Нужно открыть вкладку «Внешний вид» и изменить направление столбцов (см. рисунок 2.7).



Рис.2.7. Направление столбцов

Вторую диаграмму расположите рядом с изображением банкомата. Назовите диаграмму «АТМ Обслуживание», а в качестве значения задайте выражение:

```
delay.statsUtilization.mean()
```

Задающее среднее время обслуживания заявки в процессоре (см. Приложение №3).

Направление столбцов вертикальное.

Вид работающей модели показан на рисунке 2.8.

## 2.4. Моделирование двухканальной СМО

В банковский офис приходят клиенты. Клиент может снять деньги в банкомате, либо получить консультацию у работников банка. Первый канал – очередь клиентов к банкомату, а второй канал – очередь к консультантам. Количество консультантов равно четырем.

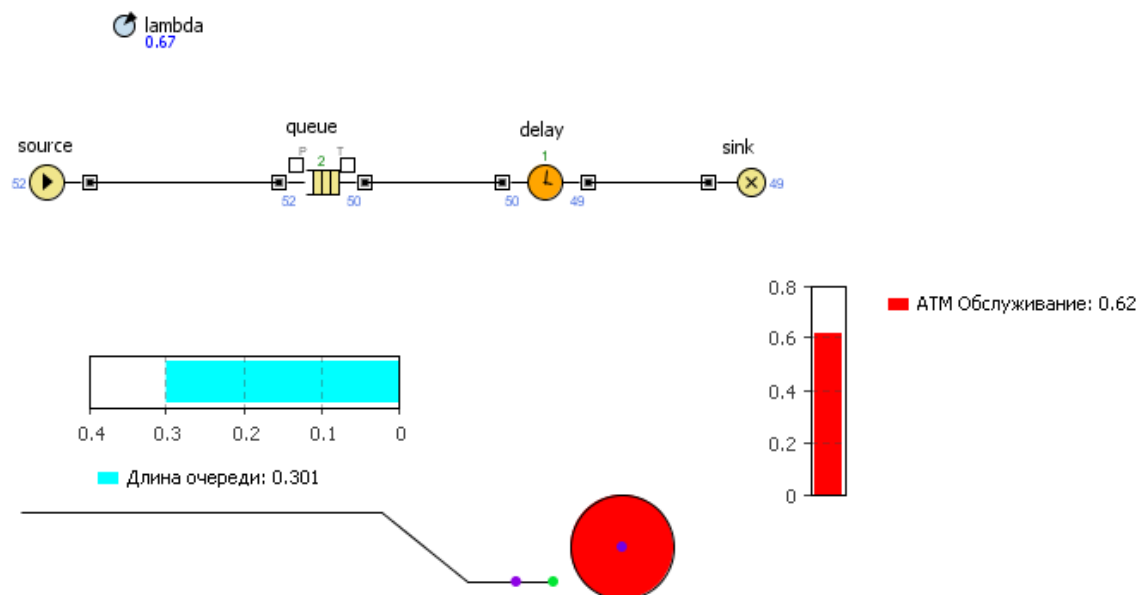


Рис.2.8. Модель с диаграммами

Для моделирования второго канала нужно использовать следующие элементы палитры Enterprise Library:

- SelectOutput – используется для моделирования процесса выбора клиентом канала обслуживания.

- Service – узел обслуживания клиентов, состоящий из очереди и прибора обслуживания delay.
- ResourcePool – источник ресурсов.

Используйте ранее созданную модель офиса с анимацией очереди к банкомату и постройте систему обслуживания в соответствии с рисунком 2.9.

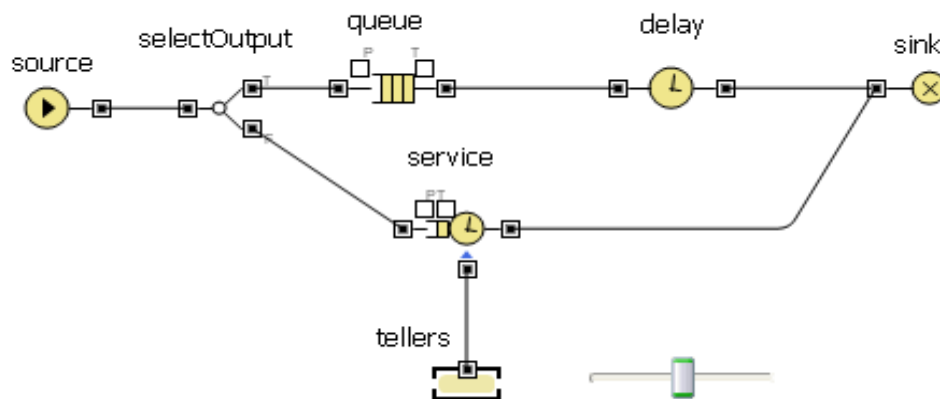


Рис.2.9. Двухканальная модель офиса.

### Элемент SelectOutput

Этот элемент должен срабатывать с заданной вероятностью. Значение вероятности равно 0.5. Равная вероятность выбора клиентом варианта обслуживания.

### Элемент Service

Время задержки соответствует треугольному закону распределения с параметрами  $x_{\min}=2,5$ ,  $x_{\max}=11$ , предпочтительное значение 6.

Вместимость очереди – 20 клиентов.

Для элемента включите сбор статистики.

### Элемент ResourcePool

Присвойте элементу имя tellers. Количество ресурсов равно 4. Это число консультантов в системе.

Включите сбор статистики.

Рядом с этим элементом поместите элемент ползунок для текущего выбора количества консультантов.

Свойства элемента:

- Минимальное значение равно нулю.
- Максимальное значение равно четырем.
- Элемент должен быть связан со свойством `capacity`(число ресурсов) элемента `tellers`.

### Анимация второго канала

Второй канал представим в виде очереди к консультантам. Расположите ее под очередью, ведущей к банкомату. Разместите рабочие места консультантов в виде ломанной линии, состоящей из трех сегментов. К рабочим местам проведите ломанную, моделирующую очередь (см. рисунок 2.10).

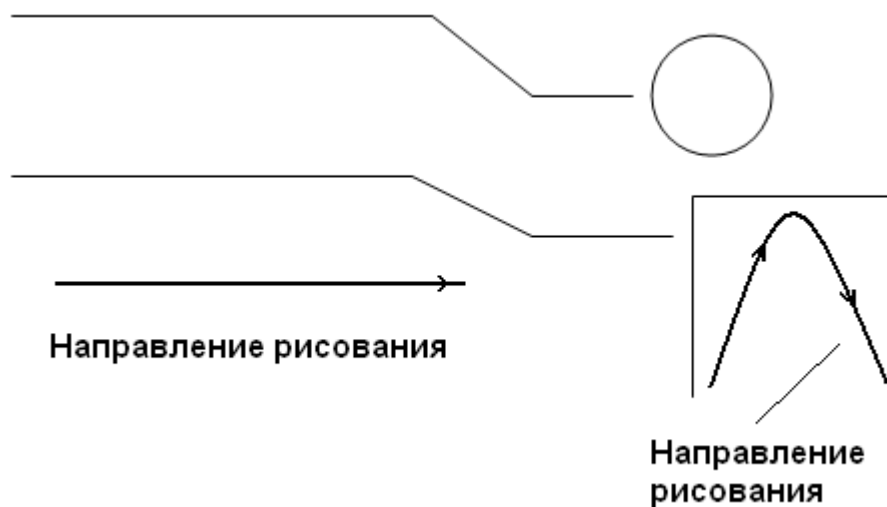



Рис.2.10. Очереди каналов СМО

Ломаную линию, для очереди назовите `qBeforeTellers`, а ломанную для рабочих мест консультантов `tellerPlaces`.

Для анимации очередей применим фигуру, отличную от стандартной. Клиентов в очереди к банкомату и консультантам

будем отображать фигурой «Человек»  из палитры «Картинки». Разместим эту фигуру за границами видимости модели и назовем

person. Рядом с этой фигурой поместите две картинки, изображающие консультантов.

Для размещения картинок нужно использовать палитру «Презентация» и два элемента «Изображение». Откройте вкладку свойств «Основные» первого изображения и нажмите кнопку «Добавить» и выберите файл busy.png (занятый консультант). Задайте имя картинке busyTeller. Установите флажок «Исходный размер» (см. рисунок 2.11).

Аналогично создайте новое изображения для свободного консультанта, графический файл idle.png (свободный консультант), имя – idleTeller.

### Настройка анимации очередей

По умолчанию очередь к банкомату изображается точками. Для использования пиктограммы person откройте элемент «Источник заявок» (Source) и на вкладке свойств «Основные» в свойстве «Фигура анимации заявки» укажите имя person.

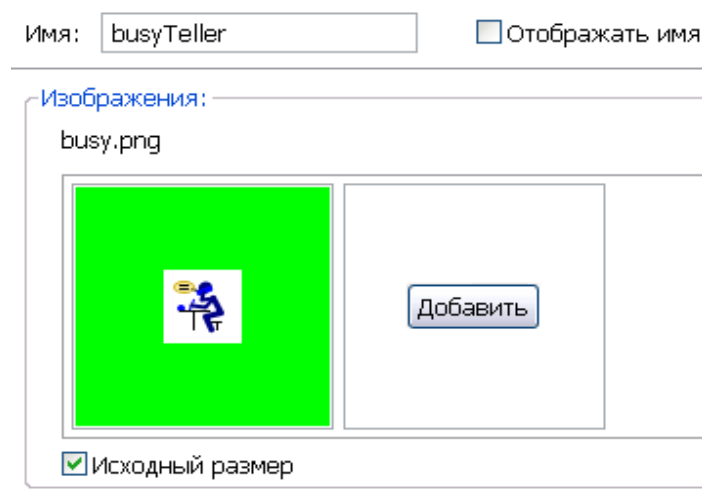


Рис.2.11. Размещение изображения

Откройте элемент Service и настройте его свойства в соответствии с рисунком 2.12.

<b>Фигура анимации (queue)</b>	qBeforeTellers
Тип анимации (queue)	Путь <input type="button" value="v"/>
Направление анимации (queue)	<input checked="" type="radio"/> Вперед <input type="radio"/> Назад
Фигура анимации (delay)	
Тип анимации (delay)	Путь <input type="button" value="v"/>
Направление анимации (delay)	<input checked="" type="radio"/> Вперед <input type="radio"/> Назад

Рис.2.12. Настройка очереди к консультантам

### Анимация рабочих мест клиентов

Откройте источник ресурсов ResourcePool и настройте его анимацию в соответствии с рисунком 2.13.

<b>Фигура анимации свободного ресурса<sup>D</sup></b>	idleTeller
<b>Фигура анимации занятого ресурса<sup>D</sup></b>	busyTeller
Уникальная фигура для каждого ресурса	<input type="checkbox"/>
Разрешить вращение	<input type="checkbox"/>
<b>Фигура анимации</b>	tellerPlaces
<b>Тип анимации</b>	Набор <input type="button" value="v"/>

Рис.2.13. Настройка анимации ресурсов

Вид презентации работающей модели показан на рисунке 2.14.

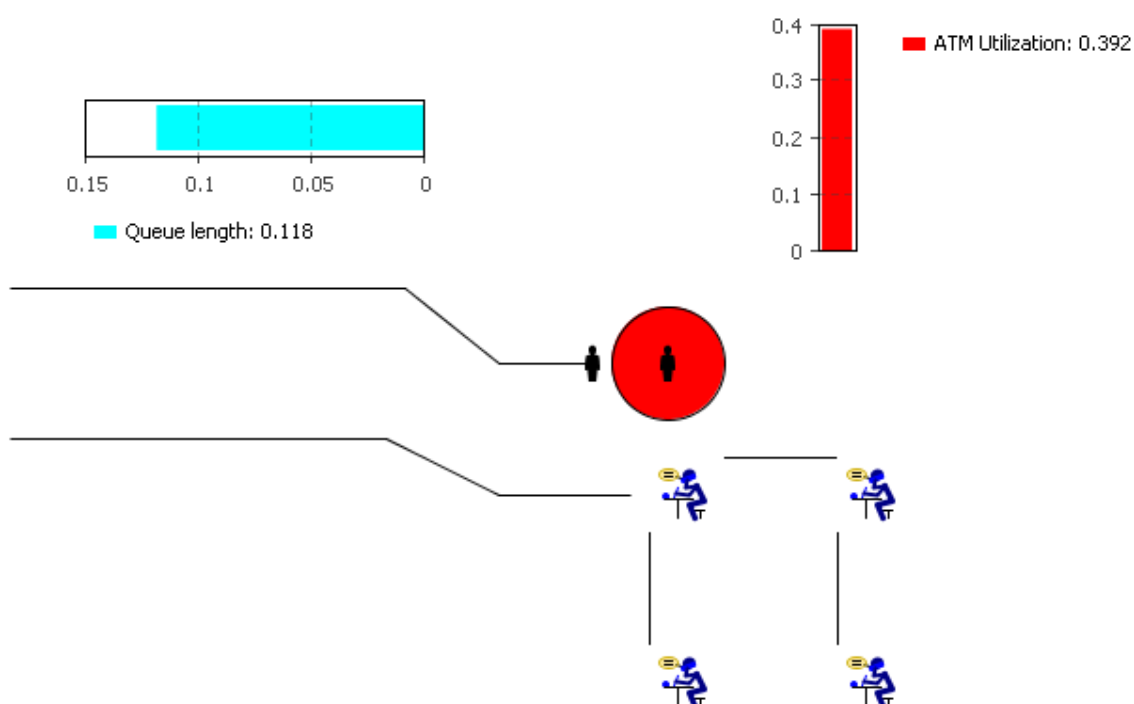


Рис.2.14. Презентация двухканальной СМО

Изменяя количество консультантов с помощью ползунка, протестируйте работу офиса.

## 2.5. Определение параметров СМО

Возьмем модель двухканальной СМО, созданной в предыдущем разделе, и определим следующие ее параметры:

1. Распределение времени ожидания обслуживания клиентом в системе.
2. Распределение времени, проведенного клиентом в системе.

### Создание класса заявки

Для определения этих параметров в модель следует разместить класс, спецификация которого показана на рисунке 2.15.

Класс содержит два глобальных атрибута:

`enteredSystem` – время появления клиента в системе.

`startWaiting` – время начала ожидания обслуживания.

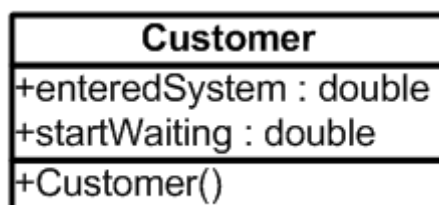


Рис.2.15. Класс для учета времени.

Размещение класса выполняется путем вызова диалога, показанного на рисунке. Для этого следует получить контекстное меню для корневого класса на дереве проекта и выполнить команду «Создать > Java класс», см. рисунок 2.16.



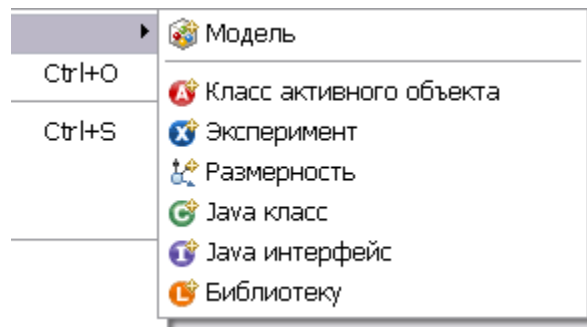


Рис.2.16. Создание класса

В результате открывается диалог создания класса. На первом шаге нужно задать имя класса и имя базового класса, который будет расширять новый класс. При работе с заявками в качестве такого класса выступает системный класс `com.xj.anylogic.libraries.enterprise.Entity`. Он представляет собой абстрактную заявку (см. рисунок 2.17).

#### Java класс

Создание нового Java класса

Рис.2.17. Создание класса, первая фаза

Затем следует перейти ко второму шагу и задать поля класса – атрибуты, так как это показано на рисунке 2.18.

#### Поля класса

Добавьте поля Java класса

Имя	Тип	Доступ	Начальное зна...
enteredSystem	double	default	
startWaiting	double	default	

☒ Создать конструктор  
☒ Создать метод toString()

Рис.2.18. Создание класса, завершение

В результате будет создан класс Java, код которого можно открыть и отредактировать при необходимости.

### **Сбор параметров системы**

Разместите в модель из палитры «Статистика» два элемента «Гистограмма» и два элемента «Данные гистограммы» (см Приложение №3). Гистограммы будут отображать распределение времен в системе на основе данных, собранных в элементах «Данные гистограммы». Первый такой элемент назовите `waitTimeDistr` (распределение времени ожидания), а второй `timeInSystemDistr` (распределение времени пребывания в системе), см. рисунок 2.19.

Откройте первую гистограмму, нажмите кнопку «Добавить данные». Задайте заголовок «Распределение времени ожидания», в поле «Данные» задайте `waitTimeDistr`, выберите цвет гистограммы. Затем откройте вторую гистограмму, добавьте данные и задайте в качестве данных `timeInSystemDistr`, а подпись «Время в системе».

При настройке гистограмм должен быть активен флажок «Отображать плотность вероятности».

Откройте элемент `Source` и измените его настройки. На вкладке «Основные» установите свойства.

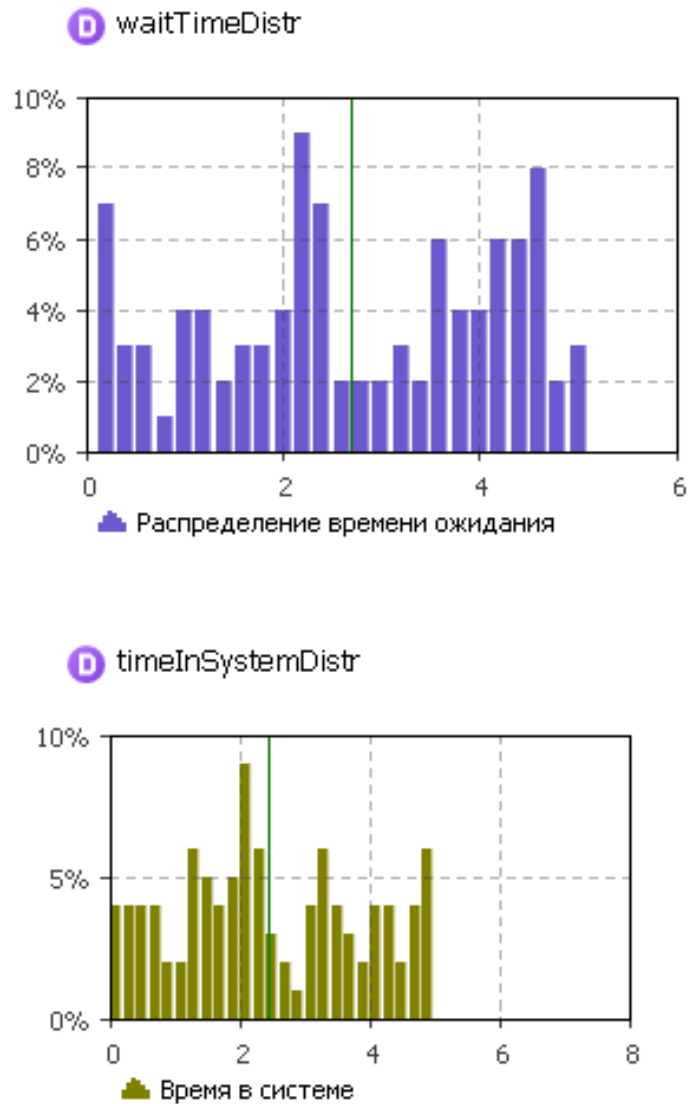


Рис.2.19. Гистограммы

Класс заявки: `Customer`

Новая заявка: `new Customer()`

Действие при выходе: `entity.enteredSystem=time()` .

С помощью оператора `new` получают новый экземпляр заявки. Для обращения к атрибутам экземпляра служит указатель `entity`. При покидании заявки источника фиксируется время выхода.

Откройте элемент очередь – `queue`. Задайте параметры

Класс заявки: `Customer`

Действие при входе: `entity.startWaiting=time()`

Действие при выходе:

```
waitTimeDistr.add(time()-entity.startWaiting)
```

Эти настройки позволяют получить время начала ожидания и заполнить данные для гистограммы. Элемент «Данные гистограммы» содержит метод `add`, с помощью которого добавляют данные. В данном случае это разность между текущим временем и временем начала ожидания.

Откройте элемент `Sink` и установите параметры.

Класс заявки: `Customer`

Действие при входе: `timeInSystemDistr.add(time()-entity.enteredSystem)`

Данные настройки позволяют зафиксировать время пребывания заявки в системе от входа до ее обработки.



Рис.2.20. Гистограммы параметров системы

Запустите модель. Гистограммы распределения времен показаны на рисунке 2.20, вертикальная линия отображает среднее значение распределения. Для отображения среднего значения времени откройте последовательно гистограммы и на вкладке «Основные» установите флажок «Отображать среднее», выберите цвет линии.

### **Самостоятельные задания**

Задание №1. Постройте гистограмму распределения времени в точке обслуживания клиентов банкоматом.

Задание №2. Постройте гистограмму распределения времени обслуживания клиентов консультантами банка.

Задание №3. Увеличьте в модели число консультантов до пяти, протестируйте созданную модель.

Задание №4. Измените правила распределения клиентов между каналами обслуживания элементом `SelectOutput`. Выберите на вкладке «Основные» режим переключения «При выполнении условия». В качестве условия задайте `queue.size()<15`, где 15 максимальная длина очереди, а `size()` метод, который возвращает текущий размер очереди.

Поставьте эксперимент для определения значения интенсивности входного потока заявок, при котором система сохраняет работоспособность. Для изменения интенсивности  $\lambda$  в диапазоне от  $[0.1...3]$  создайте интерфейс эксперимента с помощью ползунка. Обеспечьте предварительный просмотр выбранного значения  $\lambda$ .

### **Контрольные вопросы**

1. Перечислите особенности систем массового обслуживания (СМО).

2. Перечислите базовые элементы AnyLogic необходимые для построения канала СМО и дайте им краткую характеристику.
3. Как строится анимация процесса обслуживания заявок в AnyLogic?
4. Как определить среднее число заявок в очереди в модели AnyLogic?
5. Как определить среднее число заявок обслуженных процессором в модели AnyLogic?
6. Какие элементы AnyLogic нужно использовать для моделирования канала обслуживания с учетом ресурсов?
7. Как строится модель AnyLogic в которой выполняется выбор канала обслуживания?
8. Перечислите особенности построения анимации канала обслуживания с ресурсами?
9. Как создать в модели СМО класс заявки отличной от принятого по умолчанию?
10. Как в модели СМО использовать класс заявки созданный разработчиком?
11. Перечислите основные этапы создания гистограмм в модели СМО.

### **3. Исследование систем массового обслуживания**

#### **3.1. Задача Эрланга**

Задача Эрланга связана с разработкой системы массового обслуживания для телефонной сети.

#### **Постановка задачи**

В систему поступают заявки – телефонные вызовы с интервалом времени между вызовами, который соответствует экспоненциальному закону распределения с интенсивностью  $\lambda=1.5$ .

Заявки обслуживаются в процессоре. Обслуживание заявки выполняется с интенсивностью  $\mu=0.5$ . При обслуживании вызова используется  $N$  каналов телефонной сети. Число каналов телефонной станции изменяется от 1 до 100.

Постройте модель обслуживания телефонных вызовов в соответствии с описанием. Вид модели показан на рисунке 3.1.

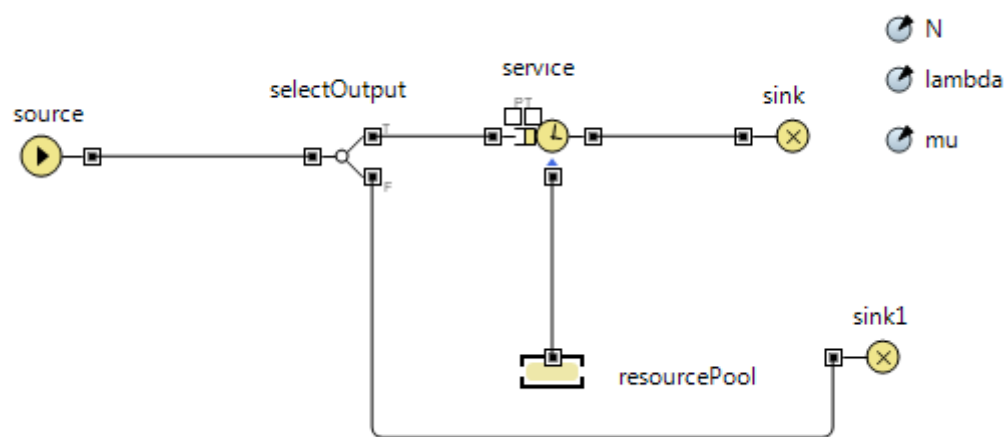


Рис.3.1. Система обслуживания вызовов

Параметры модели:  $N$ - число каналов – линий связи,  $\lambda$  (lambda) – интенсивность прихода заявок,  $\mu$  (mu) – интенсивность обслуживания заявок в процессоре – `service`.

Параметр  $N$  должен быть целого типа `int`.

### Настройки элементов модели

Элемент `source` – источник заявок. Его настройки показаны на рисунке 3.2.

Заявки прибывают согласно ☐ Интенсивности ☒ Времени между прибытиями

Время между прибытиями<sup>D</sup>

Количество заявок, прибывающих за один раз<sup>D</sup>

Рис.3.2. Настройка источника заявок

Для моделирования интервала времени используется встроенная функция AnyLogic `exponential`.

Процессор обслуживания использует ресурсы – каналы телефонной сети, которые моделируются элементом `resourcePool`. Настройки элемента должны соответствовать рисунку 3.3.

Ресурсы моделируются ☐ Как индивидуальные объекты ☒ Просто как их количество

Количество задано ☒ Напрямую ☐ Табличной функцией

Количество ресурсов

Рис.3.3. Источник ресурсов

Процессор обслуживания моделируется элементом `service`. Его настройки показаны на рисунке 3.4. Время обслуживания заявки подчиняется экспоненциальному закону. Система обслуживания телефонных вызовов представляет собой двух канальную систему. Первый основной канал, в котором обслуживаются вызовы, второй канал служит для приема отвергнутых системой вызовов. Распределение заявок-вызовов выполняет элемент `selectOutput`.



Количество ресурсов <sup>D</sup>	1
Время задержки <sup>D</sup>	exponential(mu)
Объект ResourcePool <sup>D</sup>	resourcePool
Действие при входе <sup>D</sup>	
Действие при начале задержки <sup>D</sup>	
Действие при выходе <sup>D</sup>	
Вместимость очереди	100

Рис.3.4. Настройка процессора

Его настройки показаны на рисунке 3.5.

Выход true выбирается	<input checked="" type="radio"/> При выполнении условия
Условие <sup>D</sup>	<code>service.delaySize() &lt; N</code>

Рис.3.5. Настройка элемента распределения заявок

Распределение заявок выполняется по условию. Условие проверяет наличие свободных каналов в процессоре. Для этого вызывается метод `delaySize`, который возвращается число точек обслуживания компонента `service`, используемых в текущий момент.

Выполните настройку эксперимента модели:

- Моделирование останавливается в заданное время.
- Модельное время измеряется в минутах.
- Конечное время моделирования равно 500 минут.
- Презентация прорисовывается со скоростью 8 единиц.

Примите в качестве доступного числа каналов `N` значение равное трем.

Выполните запуск модели.

### 3.1.1. Определение расходов на обслуживание телефонных вызовов

Требуется определить такое количество каналов связи  $N$ , чтобы при заданной интенсивности поступления звонков  $\lambda$  и интенсивности их обслуживания  $\mu$  прибыль от использования системы была бы максимальной.

Для вычисления прибыли введем формулу:

$$BenefitMean = \frac{Income - Penalties}{t} - equipmentPrice(N) \quad (3.1)$$

Здесь:  $t$  – текущее время,  $Income$  – стоимость обслуженных звонков,  $Penalties$  – начисленный штраф за не обслуженный вызов,  $equipmentPrice$  – функция, которая определяет стоимость обслуживания задействованных каналов.

Введем в модель системы еще два параметра:

$minPrice=0.12$  у.е. (стоимость обслуживания звонка клиента)

$penaltyPerCall=1$  у.е. (стоимость штрафа, за не обслуженный вызов)

Используя палитру «Основная» введите в модель функцию  $callPrice$  с помощью инструмента «Функция». Эта функция позволит вычислить стоимость обслуженного звонка.

Аргумент функции  $t$  текущее время. Тип аргумента `double`.

Функция возвращает результат так же типа `double`.

Код функции:

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

Если текущее время меньше 1 модельного времени, то значение функции – минимальная стоимость обслуживания вызова, в противном случае стоимость звонка равна произведению минимальной стоимости обслуживания на текущее время разговора.

Для получения стоимости обслуживания каналов связи нужно в модель ввести табличную функцию, с помощью элемента палитры «Основная». Это функция `equipmentPrice`. Для ее создания нужно задать ее свойства в соответствии с рисунком 3.6.

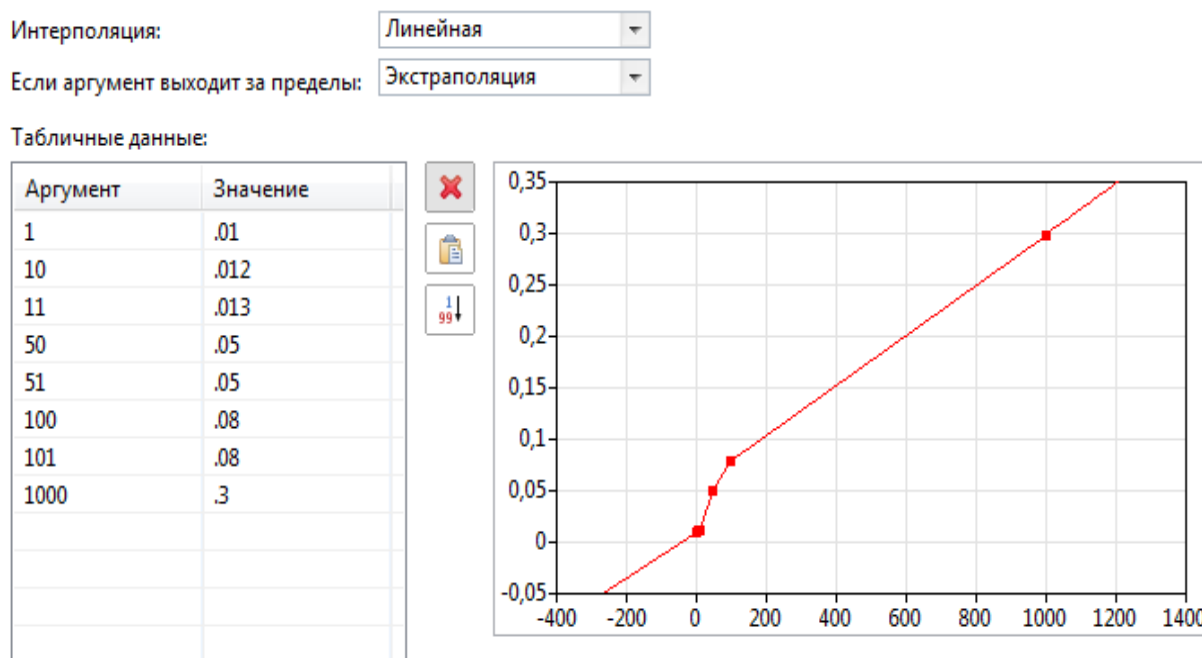


Рис.3.6. Функция вычисления стоимости каналов

Для использования функции `callPrice` нужно знать время, затраченное на обслуживание заявки в процессоре `service`. Что бы вычислить это время введите в модель класс заявки `Call`, спецификация класса показана на рисунке 3.7.

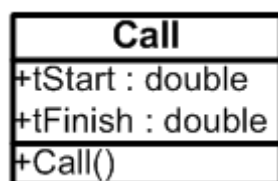


Рис.3.7. Класс Call

Класс содержит два атрибута с доступом `public`. Первый атрибут `tStart` используется для фиксирования времени начала обслуживания вызова в процессоре `service`, а второй атрибут

Настройте модель таким образом, что бы она учитывала прохождение по каналам системы заявки класса `Call`. Определите, используя атрибуты классы, время начала обслуживания вызова и время завершения обслуживания вызова в процессоре `service`.

```
Income+=callPrice(entity.tFinish-entity.tStart);.
```

```
Penalties+=penaltyPerCall;.
```

В результате модель примет вид, показанный на рисунке 3.8.



44

Проведите несколько экспериментов с разными значениями  $N$ . Проследите, как меняется прибыль от эксплуатации системы с разным числом каналов. При уменьшении  $N$  начинается увеличиваться число отклоненных вызовов. Увеличение каналов приводит к отрицательному доходу, так неоправданно возрастают расходы на их обслуживание.

### 3.1.2 Определение оптимального числа каналов

Из результатов моделирования с разным числом каналов  $N$  следует, что существует оптимальное число каналов, при котором прибыль от использования системы будет максимальная.

Что бы определить это число нужно использовать оптимизационную подсистему AnyLogic.

Введем в модель новый – оптимизационный эксперимент. Для этого нужно вызвать контекстное меню активного класса модели Main, с помощью правой кнопки мыши и выполнить команду «Создать», так как это показано на рисунке 3.9.

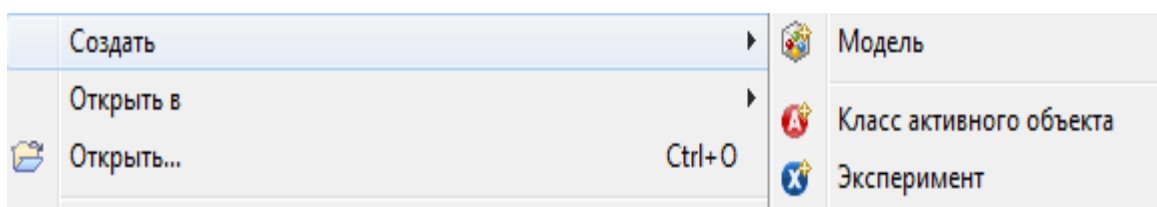


Рис. 3.9. Контекстное меню

Затем следует выполнить команду «Эксперимент».

В открывшемся окне нужно выбрать тип эксперимента «Оптимизация», генератор случайных должен быть настроен, так как это показано на рисунке 3.10. Затем следует нажать кнопку «Готово».

Созданный оптимизационный эксперимент требует дальнейшей настройки.

Тип эксперимента:

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

Ищет значения параметров, при которых достигается оптимальное значение заданной целевой функции. Может быть задан ряд ограничений на значения параметров и переменных модели.  
Оптимизация в условиях неопределенности производится с помощью репликаций. Отображается график прогресса выполнения оптимизации.

Генератор случайных чисел:

☐ Случайное начальное число (уникальные "прогоны")

☒ Фиксированное начальное число (воспроизводимые "прогоны")

Начальное значение:

Рис.3.10. Выбор типа эксперимента

Откройте в окне свойств эксперимента вкладку «Основные» и выполните настройку эксперимента, так как это показано на рисунке 3.11.

Целевая функция: ☐ минимизировать ☒ максимизировать

Условия остановки оптимизации

☐ Количество итераций:

☒ Автоматическая остановка

Параметры:

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


Рис.3.11. Настройка эксперимента

Оптимизационный эксперимент требует задания целевой функции. В нашей модели это значение динамической переменной BenefitMean. Так как она расположена в корневом активном классе модели, то обращение к ней выполняется с помощью

указателя `root`. При этом нужно указать цель оптимизации: поиск минимума или максимума.

Затем нужно определить режим остановки оптимизации. Для большинства моделей используется режим автоматической остановки. Далее следует указать, какие параметры будут меняться в процессе оптимизации. Для нашей модели это параметр `N`. Его нужно указать как дискретный и указать минимальное значение, максимальное значение, шаг изменения. Можно дополнительно задать начальное значение.

После задания всех настроек, нужно сформировать интерфейс эксперимента. На вкладке «Основные» имеется кнопка генерации интерфейса «Создать интерфейс». Нужно нажать эту кнопку, интерфейс создается автоматически и не требует настроек.

Выполните оптимизационный эксперимент. Для этого в верхней части окна программы AnyLogic выберите инструмент  и нажмите стрелку. В качестве текущего эксперимента задайте имя созданного оптимизационного эксперимента. В результате откроется окно с интерфейсом эксперимента, показанное на рисунке 3.12.

Нажмите кнопку «Запустить оптимизацию». После завершения эксперимента в колонке «Лучшее» будет показано полученное значение функционала и оптимальное значение числа каналов `N`. Справа будет отражен процесс оптимизации в виде графика.

Выполните контроль полученного значения числа каналов.

Для этого выполните созданный ранее простой эксперимент модели, установив с помощью бегунка оптимальное значение каналов `N`. Совпадение результатов экспериментов возможно

только в том случае, если простой эксперимент имеет установку для генератора случайных чисел «фиксированное начальное число» !

## ModelOptim : Optimization

Оптимизационный эксперимент

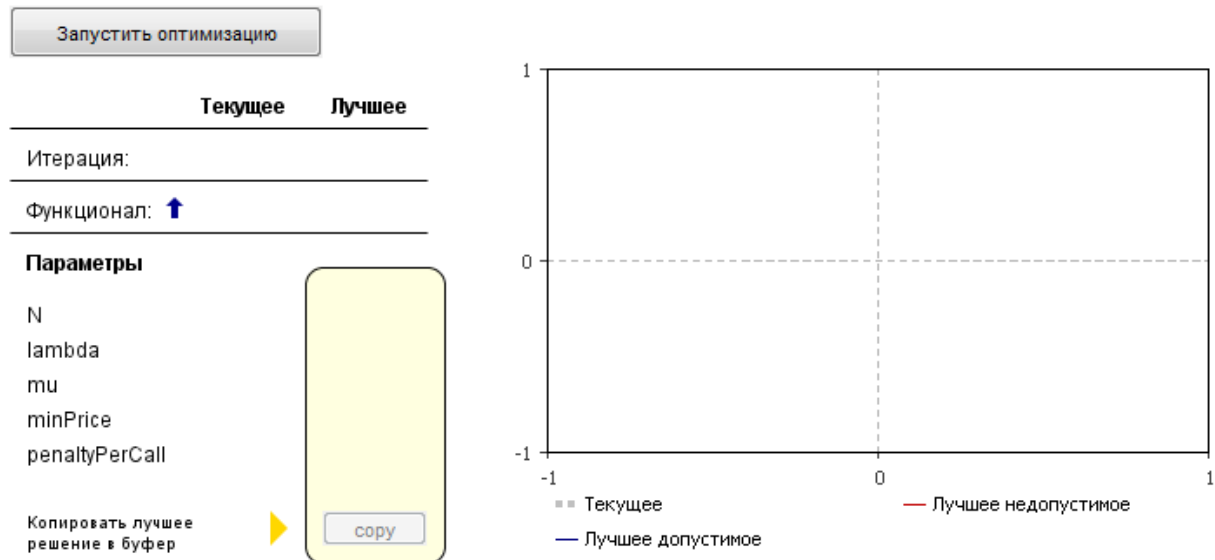


Рис.3.12. Интерфейс оптимизационного эксперимента

Настройку генератора можно проверить на вкладке «Основные».

Разместите в модель временной график, отображающий изменение прибыли.

Запустите эксперимент. С помощью бегунка установите значение, полученное в результате оптимизации.

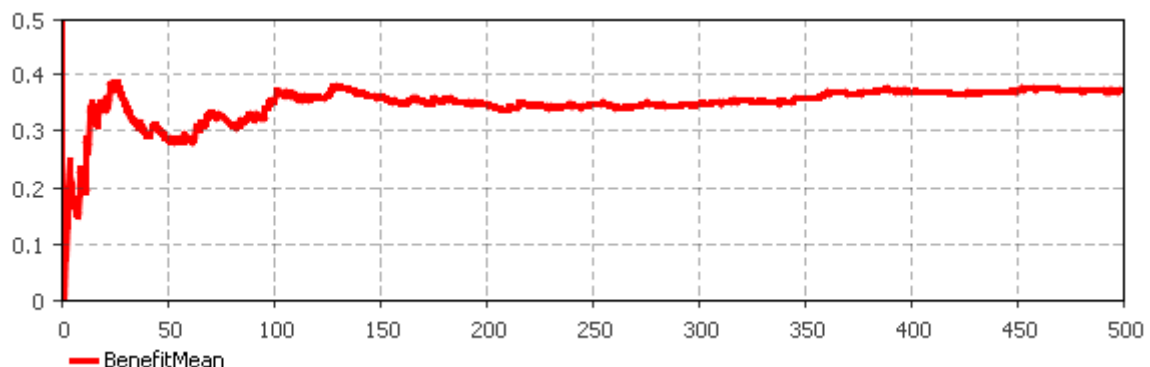


Рис.3.13. Изменение прибыли при моделировании



Сравните результаты с данными оптимизации. При правильном построении модели и оптимизационного эксперимента оптимальное число каналов  $N \cong 11$ , а значение прибыли устанавливается равным  $\cong 0.37$  у.е, так как это показано на рисунке 3.13.

### 3.2. Система массового обслуживания с отказами

Построим модель автозаправочной станции. При построении модели будем считать, что автозаправочная станция представляет собой одноканальную систему массового обслуживания (СМО). Постройте СМО в соответствии с рисунком 3.14. Элемент `sink2` свяжите с портом `P`, а элемент `sink1` с портом `T` очереди.

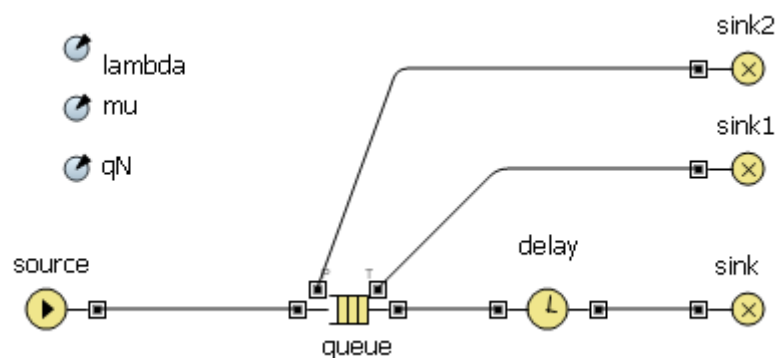


Рис.3.14. Модель СМО

Заявки – автомобили поступают в систему по времени между прибытиями. Изменение времени соответствует экспоненциальному закону распределения с интенсивностью  $\lambda$  (`lambda`), равной 0.5. Для задания времени используйте встроенную функцию AnyLogic `exponential(lambda)`.

Вместимость очереди `qN` равна 10 автомобилям. Заявки могут покидать очередь по истечении таймаута через порт `T`. Значение таймаута равно 20. Заявки могут быть вытеснены из очереди, если в очередь приходит заявка с большим приоритетом через порт `P`.

Приоритет заявок целое число, которое выбирается в модели случайно из диапазона от 1 до 5. Для получения кода приоритета используйте функцию AnyLogic `uniform`, которая возвращает равномерно распределенное число в заданном диапазоне:  
`(int)uniform(1,5).`

Так как функция возвращает результат типа `double`, его нужно преобразовать к типу `int` с помощью операцию приведения типов Java (`type`), здесь `type` – требуемой результирующий тип преобразования.

Процессор обслуживания (бензоколонка) `delay` выполняет обработку заявки с явно заданным временным интервалом, который подчиняется экспоненциальному закону распределения. Интенсивность работы процессора  $\mu$  (`mu`) равна 0.25.

Настройте эксперимент модели:  
Модельное время измеряется в минутах.  
Конечное время моделирования 500 минут.  
Режим выполнения презентации равен 32 единицам.  
Создайте интерфейс для эксперимента, так как это показано на рисунке 3.15.

## ISSLSMO1

*Страница настроек эксперимента*

Запустить модель и открыть презентацию класса Main

Начальное значение очереди:

Рис.3.15. Интерфейс эксперимента

Интерфейс позволяет изменять с помощью текстового поля емкость очереди СМО -  $q_N$ . В качестве начального значения примите длину равную 10 заявкам.

Выполните запуск модели.

Внесите изменения в модель.

Определите в процентах:

- долю вытесненных заявок - автомобилей через порт Р очереди;
- долю заявок - автомобилей покинувших СМО не дождавшихся обслуживания (вытесненных через порт Т очереди);
- долю обработанных заявок - автомобилей.

Постройте круговую диаграмму, показывающую распределение долей заявок в СМО.

Для вычисления долей заявок в процентах введите в модель динамические переменные, значение которых вычисляется с помощью функции: `fProcent(source, exit)`

Функция возвращает значение `procent`, которое определяется по формуле:

$$procent = \begin{cases} 0, & \text{при } source = 0 \\ exit/source \cdot 100, & \text{в противном случае} \end{cases}$$

Параметры:

`source` – количество заявок, от которого вычисляется процент;

`exit` – полученное количество заявок (отработанных, вытесненных, отклоненных)

Все параметры функции должны быть типа `double`. Возвращаемое значение функции, также должно быть типа `double`.

Для использования функции нужно определить количество соответствующих заявок. Это можно сделать с помощью

соответствующих счетчиков, введенных в модель как простые переменные типа `double`.

Создайте анимацию для процесса обслуживания заявок. В качестве пиктограммы анимации используйте «Грузовик2» из палитры «Картинки». Вид анимации в конструкторе модели показан на рисунке 3.16. Если процессор занят, то фигура анимации «Бензоколонка» окрашивается в красный цвет, иначе в зеленый.

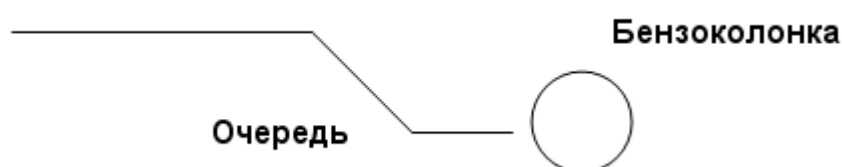


Рис.3.16. Анимация заявок

Ломаную линию – траекторию движения заявок настройте таким образом, чтобы она не отображалась в процессе работы модели. Используйте свойство «Цвет линии=Нет линии» из вкладки «Основные».

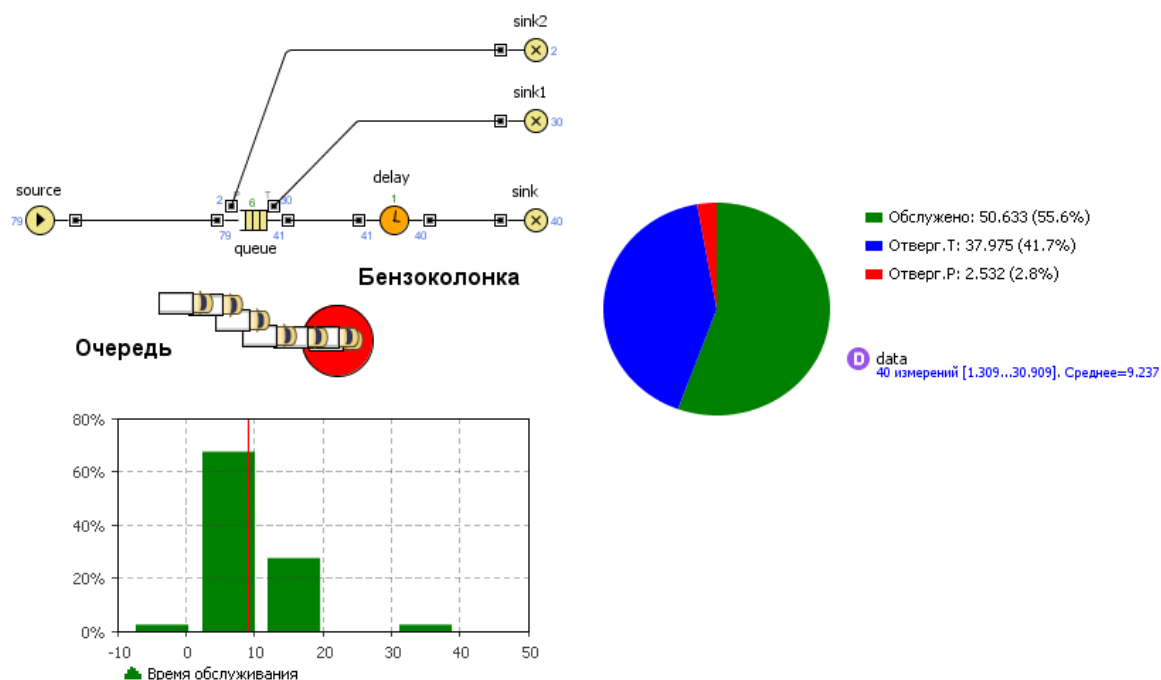


Рис.3.17. Тестирование модели одноканальной СМО

Постройте гистограмму, отражающую затраты времени на обслуживание заявки в СМО, выведите среднее значение.

Протестируйте созданную модель. Вид работающей модели показан на рисунке 3.17. Выполните запуск модели с разными значениями для емкости очереди.

Что бы определить, приемлемое значение для емкости очереди используем формулу Литтла:

$$N = \lambda \cdot t \quad (3.1)$$

Здесь:  $t$  - время нахождения заявки в СМО от момента поступления до ее обслуживания.

Для вычисления значения  $t$  разместите в модель элемент «Данные гистограммы». Создайте набор значений времен нахождения заявок в СМО от момента входа до выхода, после обслуживания.

Дополните созданную модель, добавив в нее динамическую переменную для вычисления значения  $N$  по формуле 3.1 беря в качестве времени среднее значение, возвращаемое элементом «Данные гистограммы» с помощью метода `mean()`. Выведите вычисленное значение.

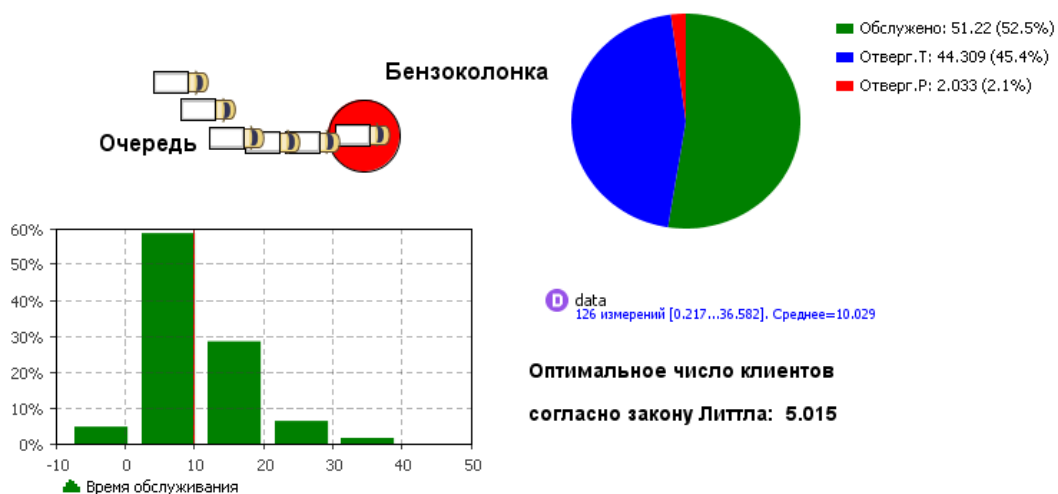


Рис.3.18. Определение емкости очереди по формуле Литтла

Запустите модель со значением  $q_N$  по умолчанию, равным 10, вид работающей модели показан на рисунке 3.18. Как видно из результатов моделирования приемлемое значение для емкости очереди составляет значение равное 5 заявкам.

Повторите запуск эксперимента модели, введя с помощью интерфейса, определенное значение  $N$  по формуле Литтла. Вид работающей модели показан на рисунке 3.19.

Как видно из рисунка значение отвергнутых заявок по времени ожидания уменьшилось, а возросла доля вытесненных заявок по приоритету.

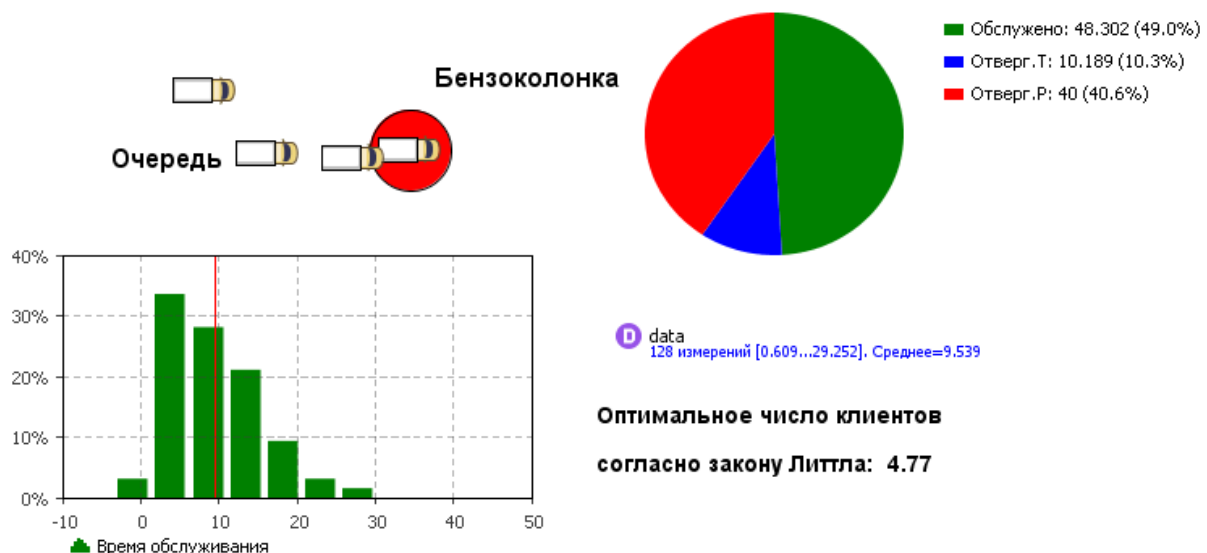


Рис.3.19. Моделирование СМО с оптимальным значением емкости очереди

### 3.3. Задания для самостоятельной работы

#### 3.3.1. Разработка двухканальной СМО

Дополните модель СМО, созданную в разделе 3.2 вторым каналом, для обслуживания заявок, вытесненных из очереди

согласно приоритету. Принципиальная схема СМО показана на рисунке 3.20

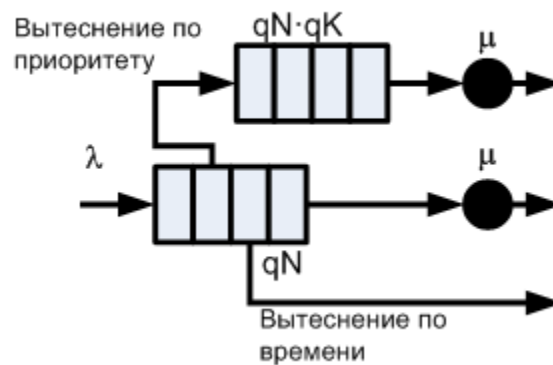


Рис.3.20. Двухканальная СМО

Емкость очереди второго канала вычисляется по формуле:

$$N = qN \cdot qK$$

Здесь  $qN$  – емкость очереди главного канала СМО, а  $qK$  – коэффициент, изменяющийся в пределах от 1 до 10.

## ISSLSMO1

*Страница настроек эксперимента*

Запустить модель и открыть презентацию класса Main

Начальное значение очереди:

Масштабный коэффициент очереди 2го канала: 1

Рис.3.21. Эксперимент двухканальной СМО

Создайте эксперимент с интерфейсом, для задания значения  $qN$  очереди и  $qK$ . Для задания  $qN$  используйте текстовое поле, а для задания  $qK$  элемент «бегунок» с начальным значением равным 1.

Проведите ряд экспериментов для определения оптимального значения  $q_K$  при значении  $q_N$ , определенном по формуле Литтла.

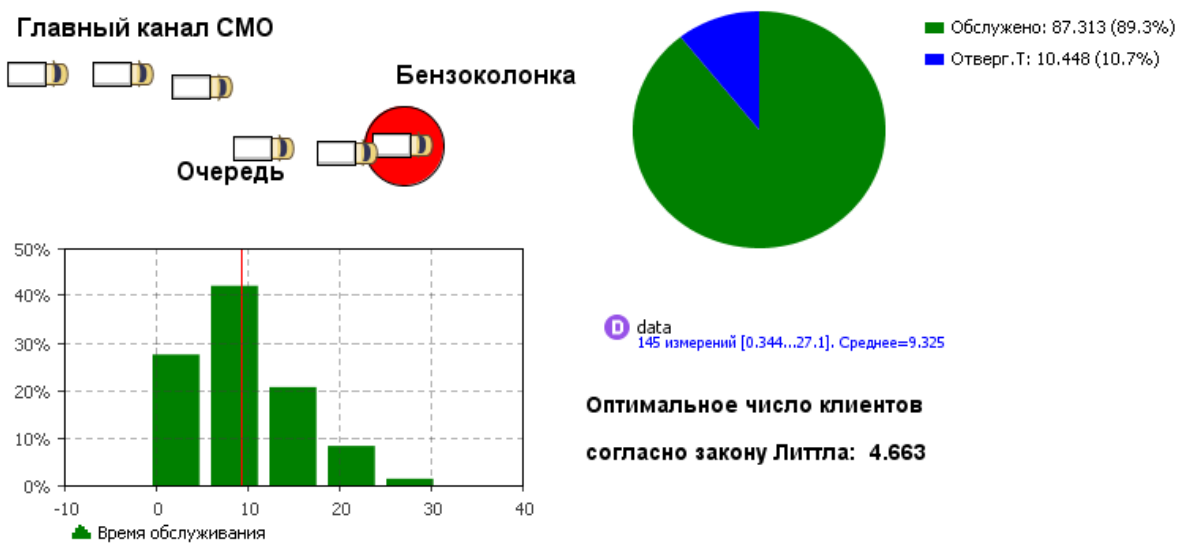


Рис.3.22. Модель двухканальной СМО

Для выбора оптимального значения коэффициента  $q_K$  выведите круговую диаграмму, показывающую долю вытесненных заявок по времени и суммарную долю обслуженных заявок в двух каналах СМО.

Результат моделирования показан на рисунке 3.22.

Дополните презентацию модели. Добавьте анимацию для показа процесса движения заявок в канале обслуживания вытесненных заявок, схема презентации та же, что и для главного канала.

### 3.3.2. Модель трехканальной СМО

Разработайте модель трехканальной СМО заправочной станции. На вход системы поступает поток заявок-автомобилей с интенсивностью  $\lambda=0.67$  и интервалом времени, который описывается экспоненциальным законом распределения. Коммутатор распределяет заявки по каналам обслуживания - бензоколонкам. Канал обслуживания состоит из накопителя



емкостью  $q_K=20$  и процессора обслуживания (колонки), работающего с интенсивностью  $\mu=0.25$ . Емкость каждого накопителя и интенсивность работы процессоров одинаковые. Распределение времени обработки заявки экспоненциальное. Заявки могут вытесняться из накопителей при истечении таймаута равного 20

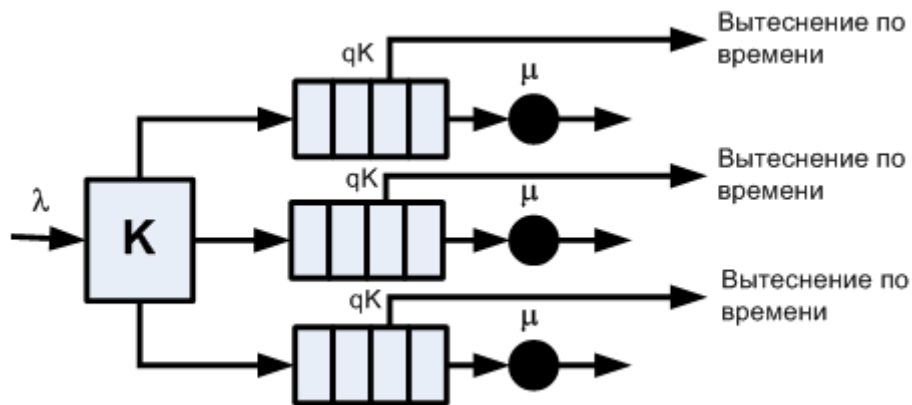


Рис.3.23. Схема трехканальной СМО

Алгоритм работы коммутатора:

1. Предполагается, что в каком либо накопителе всегда есть, хотя бы одно свободное место.
2. Сначала определяется загрузка накопителя первого канала, если в накопителе есть свободное место, то заявка помещается в накопитель.
3. Если в накопителе нет свободных, то проверяется наличие свободного места в накопителе второго канала
4. Если он занят, то заявка поступает в накопитель третьего канала.

### Методические указания

Для построения коммутатора используйте два элемента `selecOutput` из палитры Enterprise Library.

Элементы должны срабатывать по условию. Условие должно проверять наличие свободных мест в очереди канала. Для

определения текущего размера очереди используйте ее метод `size()`, который возвращает результат типа `double`.

Постройте круговые диаграммы, показывающие доли отработанных и отклоненных заявок по каждому каналу. Для каждого канала выведите количество поступивших заявок, отклоненных и отработанных.

Выведите круговую диаграмму, показывающую общую долю отработанных и отклоненных заявок в системе. Выведите общее количество поступивших в систему заявок, отклоненных, отработанных.

Определите по формуле Литтла оптимальный размер очереди накопителя.

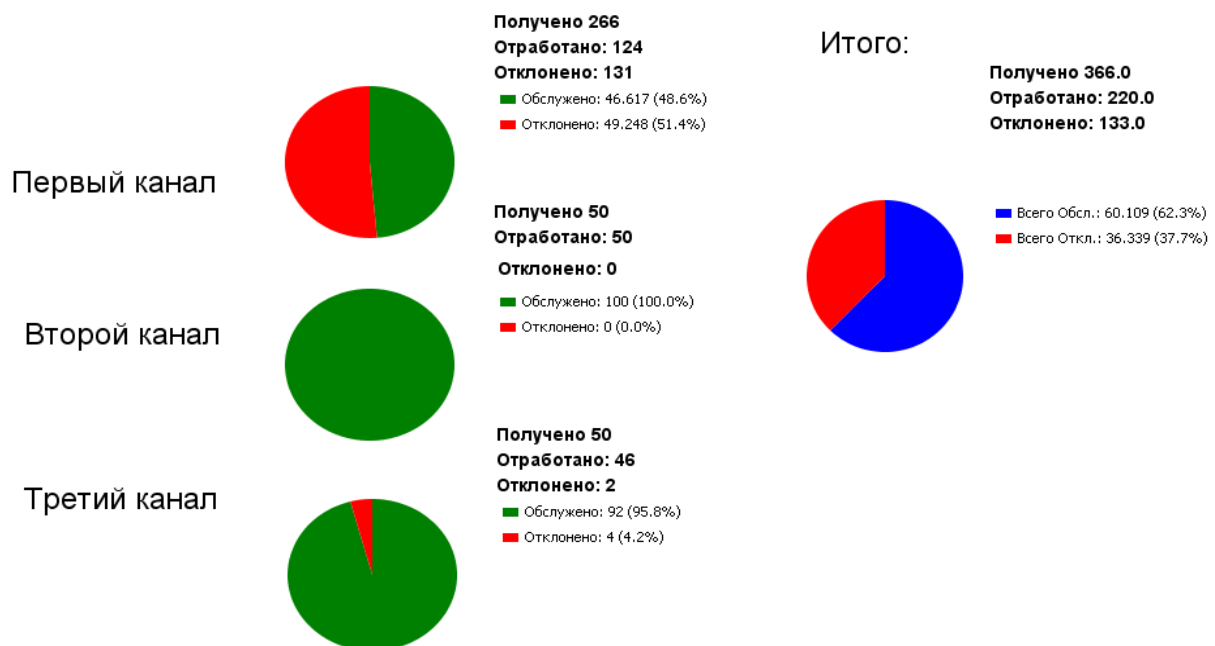


Рис.3.24. Модель трехканальной СМО

Используйте формулу в виде:

$$N \approx \lambda \cdot \sum_{i=1}^n t_i \quad (3.2)$$



В формуле  $t_i$  – время, затраченное на обработку заявки в  $i$  том канале системы.

## Методические указания

Презентацию модели в виде диаграмм и результатов моделирования постройте в редакторе модели в той части, которая отображается при запуске модели.

Саму модель СМО, переменные и константы поместите в те области редактора модели, которые не видны при ее запуске. Модель системы отделите от переменных и констант.

Используя палитру «Презентация» и элемент «Область просмотра» задайте области просмотра для видимой части модели с графиками и той области редактора, где находится модель СМО. При этом присвойте областям соответствующие заголовки. Например «СМО» для модели системы и «Диаграммы» для видимой области модели.

В процессе моделирования можно выбрать нужную область просмотра с помощью инструмента  «Показать область» панели инструментов окна моделирования. Этот инструмент входит в панель «Вид». Если этой панели нет в окне, то ее нужно вывести, используя инструмент  «Настройка панели управления»

Параметры эксперимента модели:

Модельное время измеряется в минутах.

Конечное время моделирования 500 минут.

Режим выполнения презентации равен 32 единицам.

На рисунке 3.24 показана презентация модели с оптимальным значением  $N$ . Сравните полученный результат с работой модели с заданным значением  $N$ .

### 3.3.3. Модель трехканальной СМО без очередей

Требуется разработать модель автозаправки с общей очередью, которая соответствует принципиальной схеме, показанной на рисунке 3.25

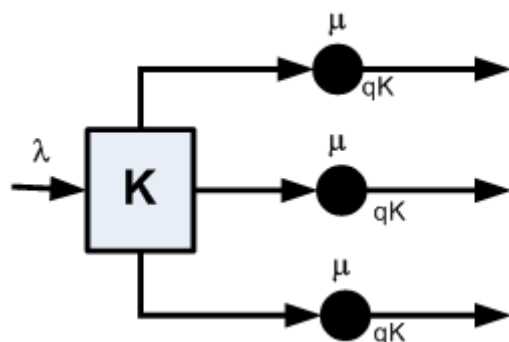


Рис.3.25. Схема трехканальной СМО без накопителя

В такой системе заявки - автомобили сразу поступают в процессор обслуживания - бензоколонку. Процессор обслуживания может принять одновременно  $q_K$  заявок. Интенсивность обслуживания заявок процессором равна  $\mu$ , распределение времени обслуживания соответствует экспоненциальному закону.

Алгоритм работы коммутатора заявок  $K$  соответствует алгоритму работы коммутатора модели задания 3.3.2.

Постройте модель для  $\lambda=0.05$ ,  $\mu=0.02$ . Заявки пребывают в систему в течении интервала времени, изменение которого соответствует экспоненциальному закону распределения. Значение  $q_K$  примите равным 10.

Определите загрузку в процентах каждого процессора системы. Результат моделирования выведите в виде презентации, показанной на рисунке 3.26. Загрузку процессоров выведите с помощью столбиковых диаграмм. Порядок прохождения заявок через процессор покажите в виде траектории их движения.

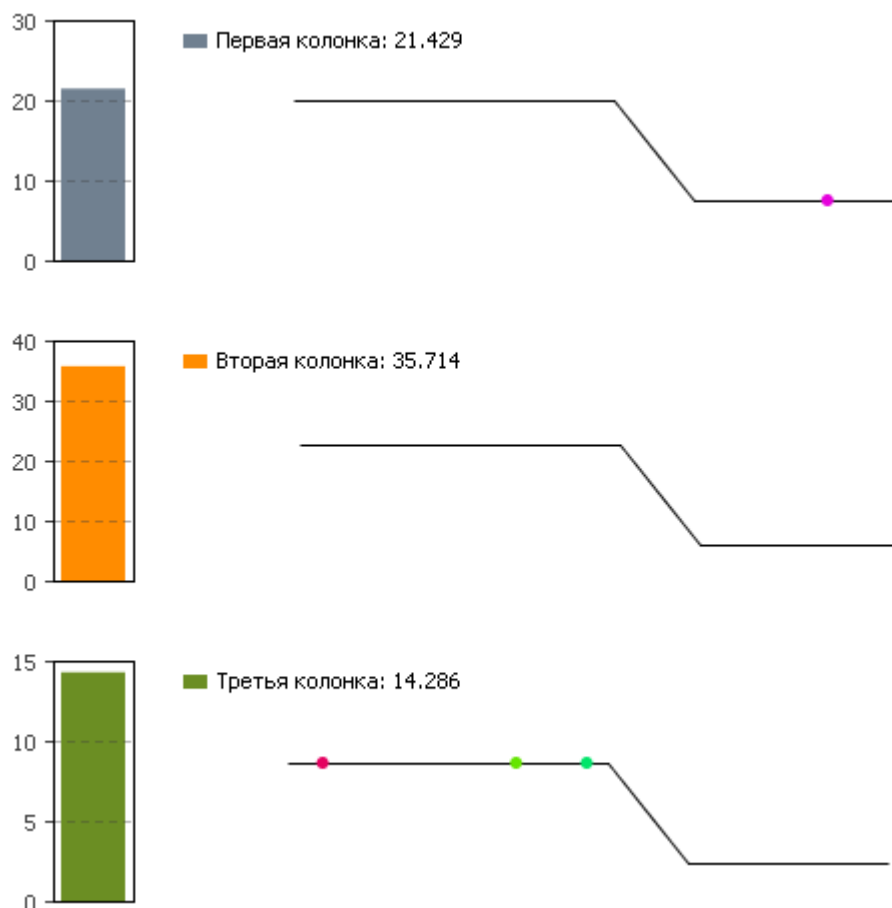


Рис.3.26. Презентация модели СМО без очередей

### Методические указания

При разработке коммутатора в качестве условия срабатывания выберите отсутствие заявок в обработке в процессоре канала. Количество удерживаемых заявок процессором (элемент `delay`) можно определить, вызвав его метод `size()`.

Определить общее число заявок поступивших в систему можно с помощью метода `count()` элемента `source`.

Для показа презентации – движения заявок разместите их траектории в виде ломанных. Настройка анимации выполняется с помощью двух свойств вкладки «Основные» элемента `delay` (процессор системы):

Фигура анимации = Идентификатор ломанной

Тип анимации = Путь

Создайте отдельные области просмотра для презентации модели и модели СМО.

Параметры эксперимента модели:

Модельное время измеряется в минутах.

Конечное время моделирования 500 минут.

Режим выполнения презентации равен 32 единицам.

### **Контрольные вопросы**

1. Что является объектом исследования при решении задачи Эрланга?
2. В каких случаях используется табличная функция в моделях AnyLogic?
3. Какие основные параметры нужно задать в модели AnyLogic при размещении табличной функции?
4. В чем отличие динамической переменной от обычной переменной в модели AnyLogic?
5. В чем смысл оптимизационного эксперимента в AnyLogic?
6. Перечислите основные этапы создания и особенности проведения оптимизационного эксперимента в AnyLogic.
7. Как построить модель канала обслуживания СМО в AnyLogic с учетом вытеснения заявок из очереди по времени?
8. Как построить модель канала обслуживания СМО в AnyLogic с учетом вытеснения заявок по приоритету?
9. Для чего используется формула Литтла при исследовании СМО?
10. Как создать анимацию процесса прохождения заявок в процессоре обслуживания в AnyLogic?

## **4. Моделирование сетей**

### **4.1. Модель работы станции автосервиса Создание рабочего поля сети**

Требуется спроектировать модель обслуживания автомобилей в авторемонтном предприятии. Такая модель представляет собой модель СМО, которую можно построить, используя сетевые элементы палитры Enterprise Library (см. Приложение №1).

Для построения модели требуется создать новую модель с нуля.

Предварительно в поле активного класса модели нужно разместить графическое изображение, которое будет представлять собой план территории авторемонтного предприятия.

Что бы разместить план нужно использовать элемент «Изображение» из палитры «Презентация». В качестве изображения выбирается файл layout.png. После загрузки изображения нужно активизировать флажок «Исходный размер».

Затем следует разместить на плане территории области, которые будут использоваться при моделировании работы СМО.

Эти области показаны на рисунке 4.1.

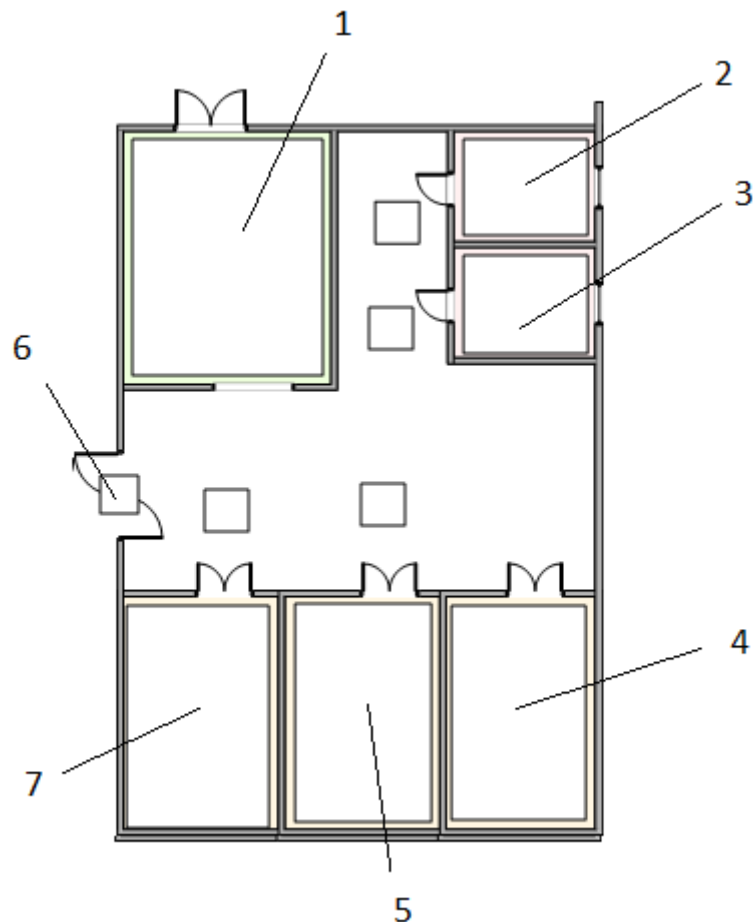



Рис.4.1. План автосервиса

Что бы создать эти области используйте элемент «Прямоугольник» из палитры «Презентация». Разместите области в соответствии с рисунком 4.1. Для удобства работы отключите автоматическую привязку фигур к линиям сетки поля графического редактора модели с помощью инструмента  (Включить/Отключить сетку), панели инструментов. Присвойте областям идентификаторы с помощью вкладки «Основные» и свойства «Имя»:

- 1- waitingHall (помещение ожидания для прибывших автомобилей)
- 2- staffRoom (комната отдыха для ремонтников, проводящих осмотр)
- 3- storageRoom (комната для хранения инструментов ремонтников)
- 4- procRoom3 (комната осмотра автомобилей)
- 5- procRoom2 (комната осмотра автомобилей)



#### 7- procRoom1(комната осмотра автомобилей)

Автомобили пребывают в помещении ожидания свободного ремонтника, затем свободный ремонтник сопровождает автомобиль для осмотра и берет необходимые ему инструменты.

Что бы смоделировать завершение осмотра автомобиля, нужно разместить точку выхода из автомастерской. Это точка отмечается небольшим прямоугольником – позиция 6. Разместите прямоугольник и присвойте ему идентификатор `exit`.

Далее следует создать траекторию движения автомобилей и ремонтников.

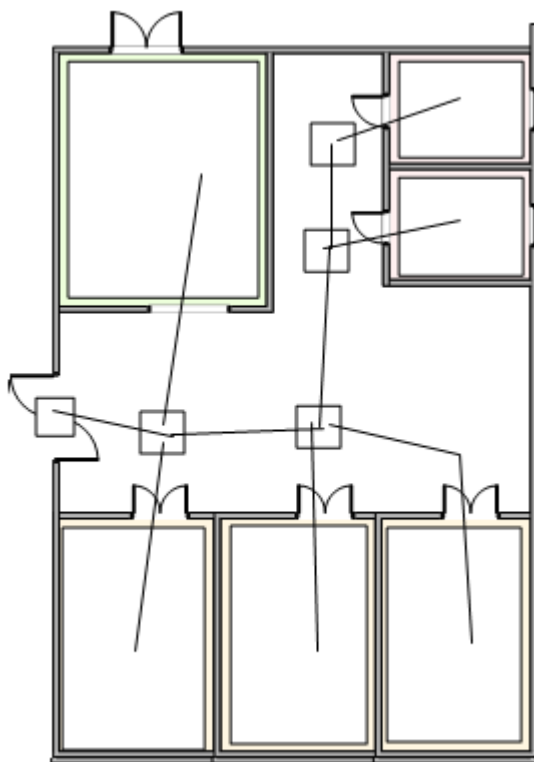


Рис.4.2. Траектория движения

Траектория формируется путем задания ее узлов с помощью прямоугольников, так как это показано на рисунке 4.1. Разместите узлы, оставив идентификаторы прямоугольников по умолчанию. Затем следует соединить узлы между собой и с помещениями с помощью элемента «Ломанная» палитры презентация, так как это

показано на рисунке 4.2. Важно: соединение выполняется парно ! Узел с узлом, помещение с узлом.

После создания траектории нужно сформировать группу, состоящую из плана помещения, прямоугольников и траектории. Для этого выделите все элементы, так как это показано на рисунке 4.3.

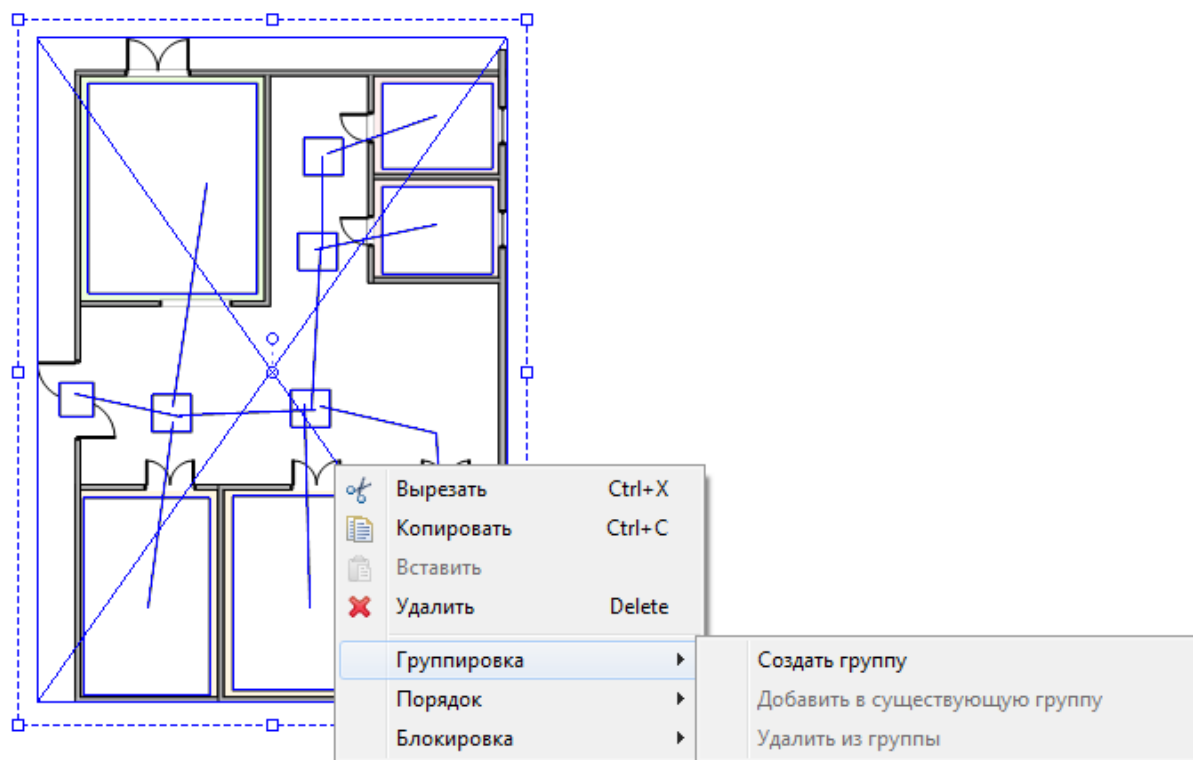


Рис.4.3. Создание группы

Выполните команду контекстного меню «Создать группу». Если, при выделении пропадают элементы, можно использовать команду «Порядок» для правильного размещения элементов, затем повторно выполняется группировка. При ошибке, группу можно расформировать с помощью команды «Разгруппировать» контекстного меню группы.

Созданная группа должна обладать идентификатором `group`.

Проверить правильность создания группы можно с помощью дерева проекта. Правильно созданная группа будет иметь вид, показанный на рисунке 4.4.

Выделяя нужный элемент на дереве можно его удалить, или переименовать с помощью соответствующих команд контекстного меню элемента.

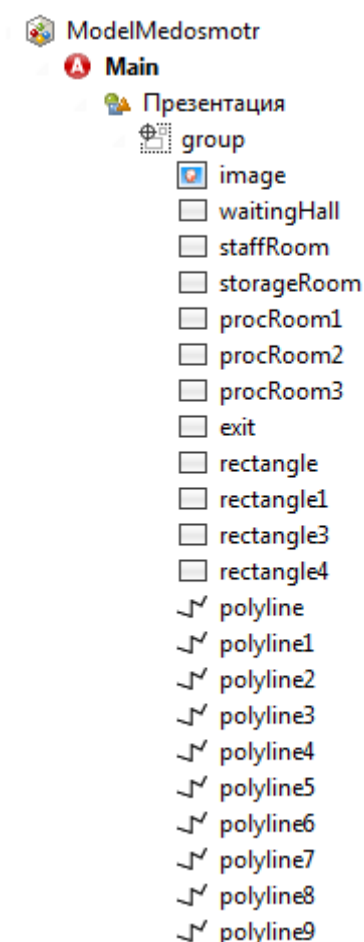


Рис.4.4. Группа на дереве проекта

Если при создании группы возникают проблемы, то создать группу можно поэлементно. Например, элемент не попал в группу. Тогда его нужно выделить щелчком мыши в поле графического редактора модели и используя контекстное меню выполнить команду: Группировка > Добавить в существующую группу. В результате графический редактор переходит в режим указания группы для помещения в нее элемента. Все доступные группы отмечаются символом – маркером «Перекрестие» (см. рисунок 4.5). Нужно выполнить щелчок на требуемом маркере и элемент попадает в группу.

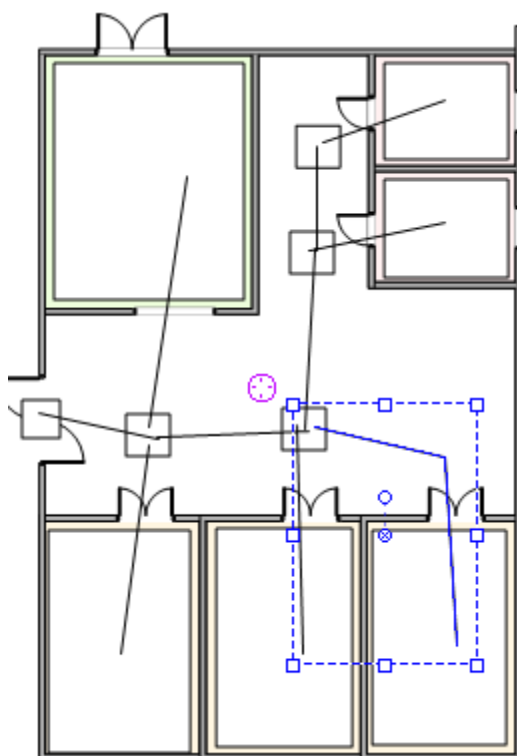


Рис.4.5. Помещение элемента в группу

Можно создать всю группу поэлементно. Для первого элемента группы выполняется команда контекстного меню «Группировка > Создать группу», а затем в группу последовательно помещают элементы, так как это было описано выше. Для ускорения процесса размещения элементов в группу их можно предварительно выделить в поле редактора при нажатой клавише CTRL, делая щелчки на нужных элементах левой кнопкой мыши.

### **Создание сети**

Следующим шагом построения модели является создание сети с помощью элементов библиотеки Enterprise Library.

Для этого используются элементы:

- Network
- NetworkResourcePool

Предварительно на рабочее поле поместите ломанную линию, так как это показано на рисунке 4.6. Используя вкладку «Основные» свойств ломанной задайте ей идентификатор `roomsLocation` и выберите красный цвет для линий ломанной. Ломанная линия используется для указания помещений при осмотре автомобилей ремонтниками.

Затем разместите один элемент `Network` и три элемента `NetworkResourcePool` и соедините их между собой (см. рисунок 4.6)

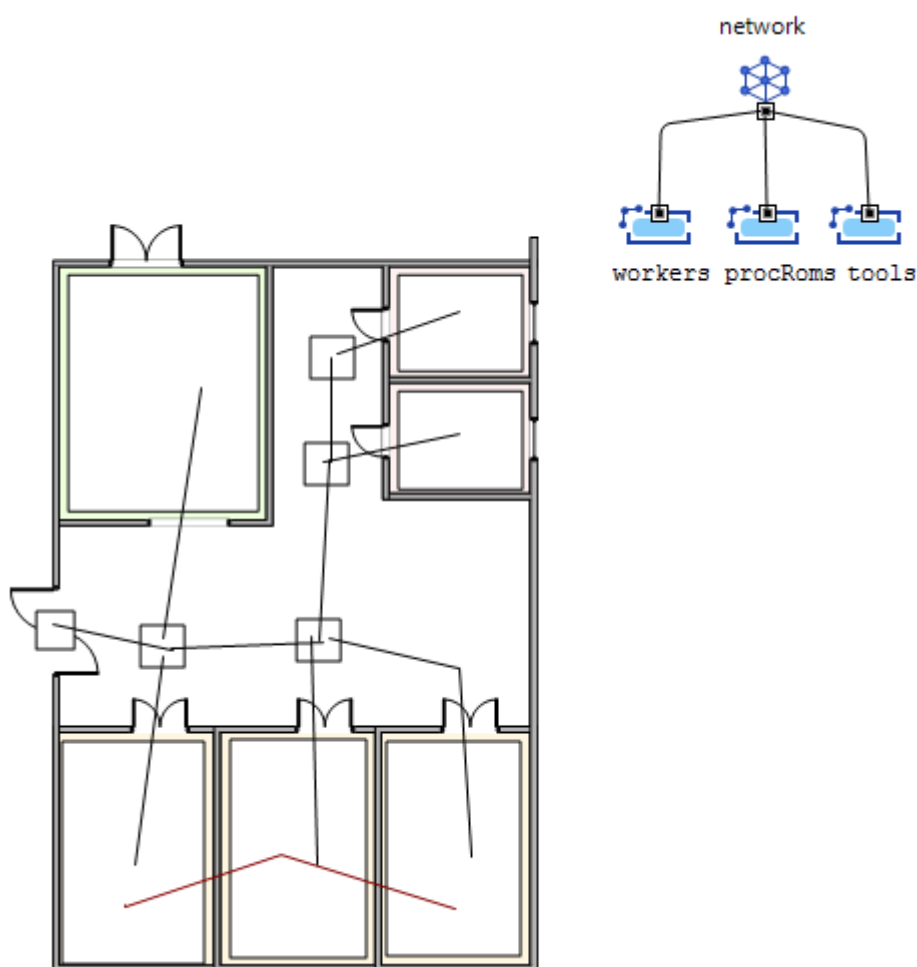


Рис.4.6. Создание сети

Используя палитру «Рисунки» разместите в поле модели два изображения: «Человек», «Грузовик2». Первому изображению присвойте идентификатор `worker` (ремонтник), оставьте идентификатор второго изображения по умолчанию. Элементы

разместите в любой не отображаемой зоне графического редактора активного класса модели. Используя палитру «Презентация» и элемент «Прямоугольник» создайте закрашенный в синий цвет прямоугольник шириной в 10 пикселей и высотой 3 пиксела. Задайте прямоугольнику идентификатор `tool`(инструмент) и расположите его рядом с фигурами ремонтника и автомобиля.

Выполните настройку элементов:

Элемент Network

Идентификатор: `network`. Остальные свойства смотри рисунок 4.7.

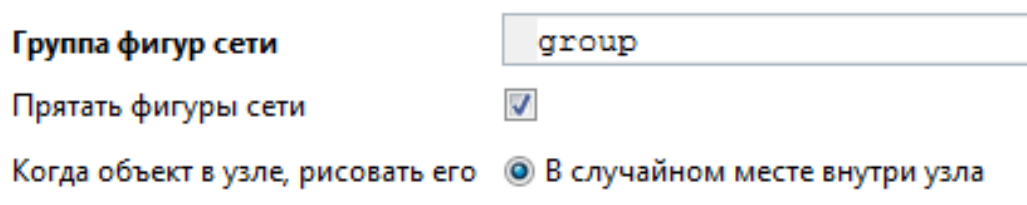


Рис.4.7. Свойства объекта Network

Настройка элементов `NetworkResourcePool`, показана в таблицах 4.1, 4.2, 4.3. Выбор элементов для настройки с лева на право (см. рисунок 4.6).

Первый элемент. Таблица 4.1.

Свойство	Значение
Имя	<code>workers</code>
Тип ресурса	Движущийся
Количество задано	Напрямую
Количество ресурсов	5
Фигура анимации свободного ресурса	<code>worker</code>
Фигура анимации занятого ресурса	<code>worker</code>
Базовое место расположения задается как	Один узел
Базовый узел	<code>staffRoom</code>

Второй элемент. Таблица 4.2.

Свойство	Значение
Имя	procRooms
Тип ресурса	Статический
Количество задано	Фигурой базового местоположения
Базовое место расположение задается как	Путь через узлы
Путь через узлы	roomsLocation

Третий элемент. Таблица 4.3

Свойство	Значение
Имя	tools
Тип ресурса	Переносной
Количество задано	Напрямую
Количество ресурсов	5
Фигура анимации свободного ресурса	tool
Фигура анимации занятого ресурса	tool
Базовое место расположение задается как	Один узел
Базовый узел	storageRoom

### Создание канала обслуживания

Что бы показать динамику обслуживания автомобилей нужно построить канал СМО, используя элементы библиотеки Enterprise Library (см. Приложение №1).

Создайте канал обслуживания, так как это показано на рисунке 4.8, используя соответствующие элементы. Затем следует настроить элементы СМО.

Элемент Source моделирует прибытие транспортных средств на технический осмотр с интенсивностью равной 0,05. За один раз пребывает один автомобиль. Для заявки – автомобиля задайте имя

фигуры анимации lorry2. Установите флажок «Разрешить вращение» в активное состояние.

Элемент NetworkEnter используется для связи с созданной сетью обслуживания. Для этого элемента нужно задать два свойства:

Сеть=network

Узел входа=waitingHall

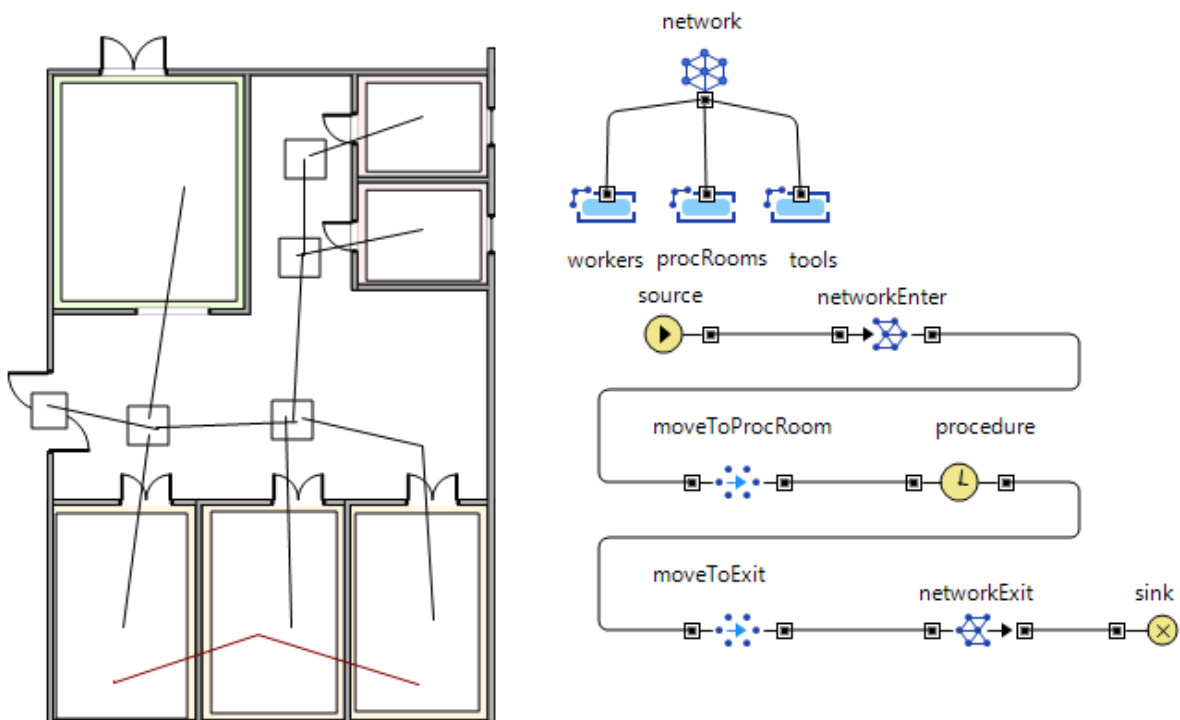


Рис.4.8. Канал обслуживания

Элемент NetworkMoveTo. Используется для моделирования перемещения в сети. Задайте ему свойства:

Имя=moveToProcRoom

Заявка перемещается=В заданный узел

Узел=procRoom1

Элемент Delay. В данной СМО моделирует процесс осмотра автомобиля ремонтником. Настройка свойств:



Имя = `procedure`

Задержка задается = Явно

Время задержки = `uniform(10)`

Вместимость=5

Элемент `NetworkMoveTo1`, моделирует убытие транспортного средства после технического осмотра. Задайте ему следующие свойства:

Имя = `moveToExit`

Заявка перемещается = В заданный узел

Узел = `exit`

Элемент `NetworkExit` используется для вывода заявки из сети.

Элемент `Sink` уничтожает заявку в СМО.

Протестируйте созданную модель. Модельное время – минуты. Время остановки не задано. Скорость прорисовки презентации задайте равной 16 единиц. Вид работающей модели показан на рисунке 4.9.

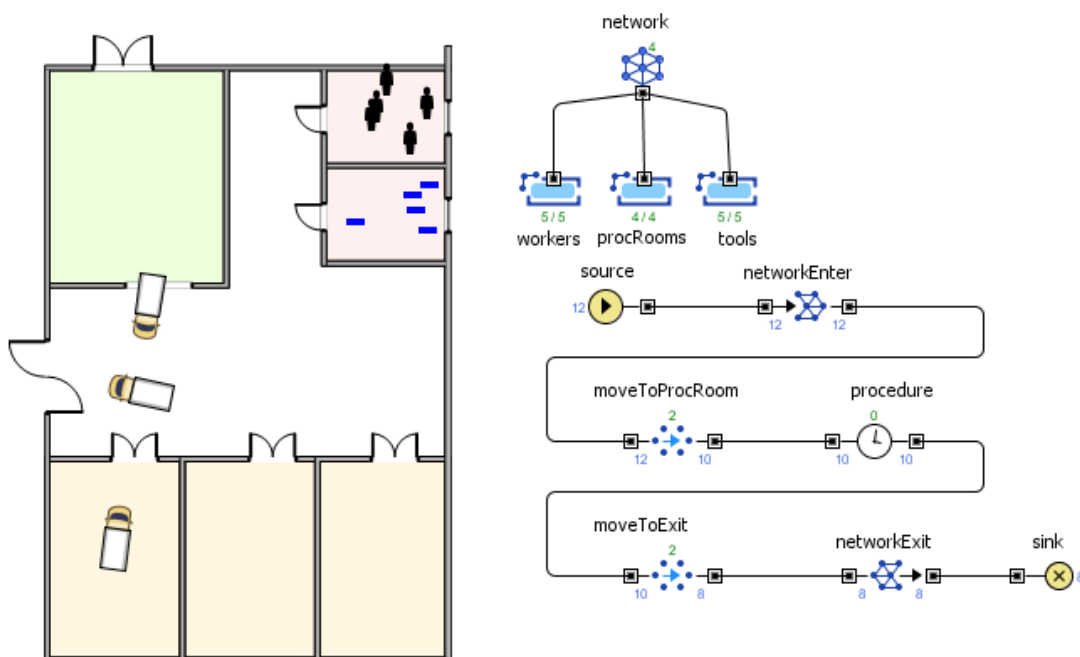


Рис.4.9. Движение транспортных средств

Данная модель позволяет проследить движение автомобилей из помещения ожидания в левое помещение осмотра, из которого транспортное средство убывает на выход.

### Моделирование обслуживания автомобилей

Чтобы получить полностью функционирующую модель нужно ввести в СМО ряд новых элементов из палитры Enterprise Library.

Измените СМО в соответствии с рисунком 4.10.

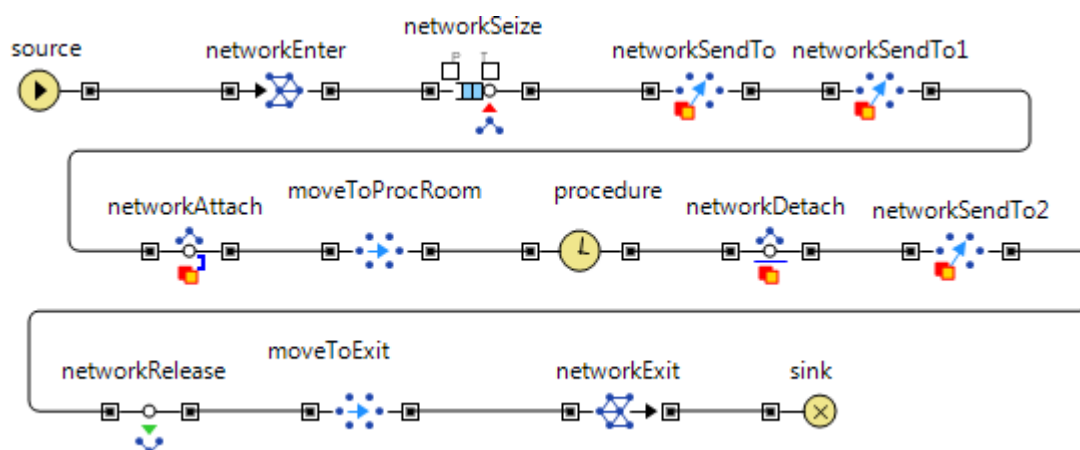


Рис.4.10. Модифицированная СМО обслуживания автомобилей

Введем новые элементы по ходу движения заявки от элемента Source.

Элемент NetworkSeize. Задает перечень ресурсов сети для обслуживания автомобилей. Этот перечень задается в свойстве

Список ресурсов = {procRooms, workers, tools}

Элемент NetworkSendTo. Используется для соединения ресурса с заявкой. Таких элементов добавлено два подряд (NetworkSendTo, NetworkSendTo1). Первый элемент связывает ремонтника с набором инструментов. Его настройки:

Имя = SendToStorage

Список ресурсов = {workers}

Отсылать ресурсы = К захваченному ресурсу

Ресурс = `tools`.

Второй элемент связывает ремонтника, взявшего инструменты с заявкой – автомобилем. Его свойства имеют значения:

Имя = `networkSendToCar`

Список ресурсов = `{workers, tools}`

Отсылать ресурсы = К заявке

Элемент `NetworkAttach`. Присоединяет ресурсы к заявке. Здесь нужно настроить свойство:

Присоединить = Все захваченные ресурсы в месте нахождения заявки

Элемент `NetworkDetach`. Отсоединяет ресурсы от заявки. Для этого элемента нужно указать свойство:

Отсоединить = Все присоединенные ресурсы

После осмотра автомобиля ремонтник должен вернуть на место взятые инструменты. Для этого он должен войти в помещение их хранения. Моделируется это действие элементом `NetworkSendTo2`. Значения свойств элемента:

Имя = `returnTool`

Список ресурсов = `{workers, tools}`

Отсылать ресурсы = В заданный узел

Узел = `storageRoom`

Элемент `NetworkRelease`. Используется для освобождения всех захваченных ресурсов. Его свойства:

Освободить = Все захваченные ресурсы

Движущиеся ресурсы = Возвращаются в базовое местоположение.

Чтобы осмотр происходил не в одном помещении, а в трех нужно изменить настройки элемента `moveToProcRoom`:

Заявка перемещается = К захваченному ресурсу

Ресурс = `procRooms`

Протестируйте модель. Вид работающей модели показан на рисунке 4.11.

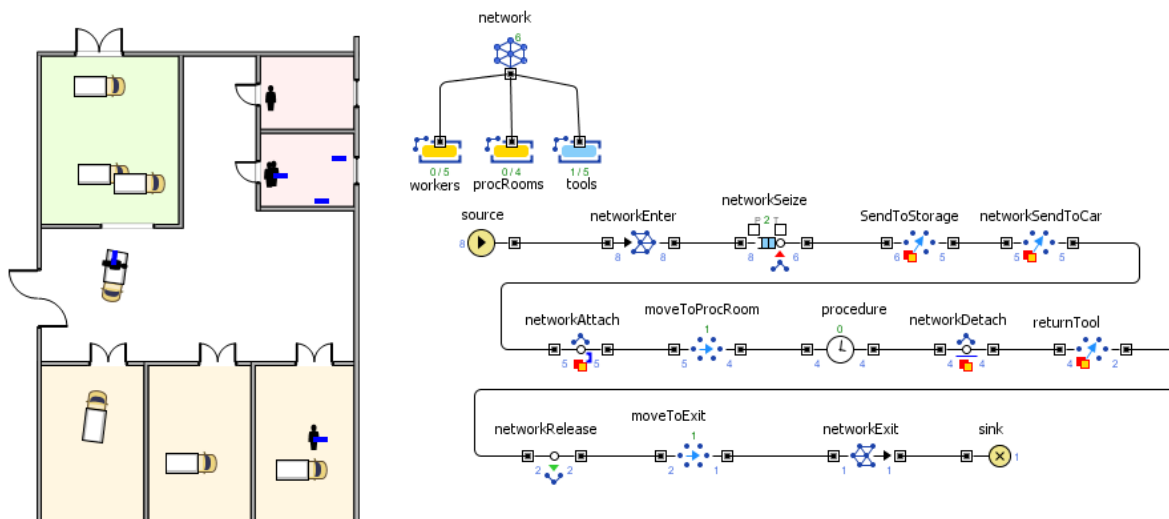


Рис.4.11. Модель обслуживания транспортных средств

### Задания для самостоятельной работы

Настройте эксперимент модели таким образом, что бы она работала фиксированный период времени, равный 700 единицам модельного времени.

1. Определите долю обслуженных транспортных средств, долю не обслуженных транспортных средств. Покажите распределение долей в виде круговой диаграммы.
2. Постройте гистограмму распределения времени нахождения автомобиля в автосервисе от момента вхождения до момента выхода после осмотра ремонтником. Определите среднее время.
3. Постройте гистограмму распределения времени, затрачиваемого ремонтником на осмотр автомобиля. Определите среднее время.
4. Определите по формуле Литтла оптимальное число транспортных средств, которые могут находиться в СМО.

5. Постройте новую модель СМО, которая показывает движение транспортных средств в сервисном пункте. Интенсивность прибытия автомобилей равна 0,07. Осматривает машины бригада из трех автомехаников. Время осмотра изменяется случайным образом от 0 до 10 минут (см. Рис. 4.12).

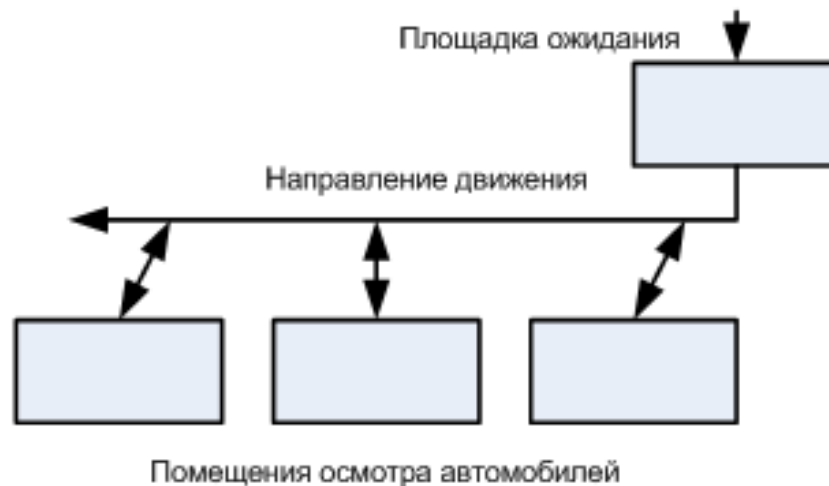


Рис.4.12. Схема движения

## 4.2. Модель вестибюль метро

В состав AnyLogic входит палитра Pedestrian Library, которая содержит элементы позволяющие моделировать СМО пешеходного - пассажирского движения (см. Приложение №2)

Требуется построить модель перемещения пассажиров в вестибюле метрополитена.

### Разметка вестибюля

Создайте новую модель с нуля. Загрузите в поле графического редактора корневого класса модели графическое изображение entrance\_layout.png, которое будет представлять собой план вестибюля. Изображение должно выводиться в исходном размере, так как это показано на рисунке 4.13.

Затем следует обозначить границу вестибюля, обведя его внутренние стены, так как это показано на рисунке 4.14. Для этого воспользуйтесь инструментом «Ломанная». Предварительно отключите режим привязки изображений к графической сетке редактора.

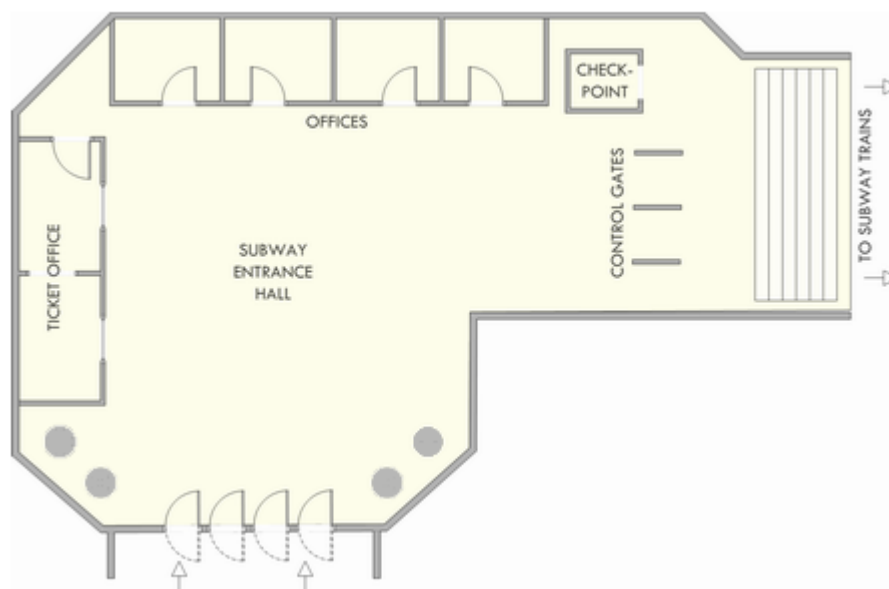


Рис.4.13. Вестибюль метро

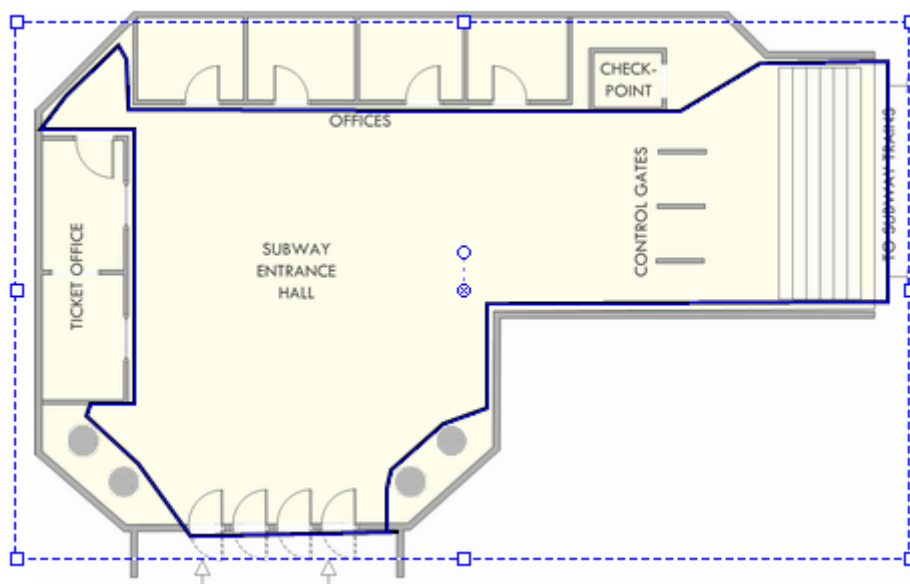


Рис.4.14. Стены вестибюля

После завершения рисования ломанной ей нужно задать идентификатор `walls` и сделать активным значение «Замкнутая». Важно: Ломанная должна быть на переднем плане !

Поместите ломанную в группу. Группа должна обладать идентификатором `group`. Затем используя элемент «Линия», определите зоны входа и выхода пассажиров. Так как это показано на рисунке 4.15. Нижняя линия определяет входную зону. Задайте линии идентификатор `entry`. Линия справа – выходная зона. Задайте ей идентификатор `exit`.

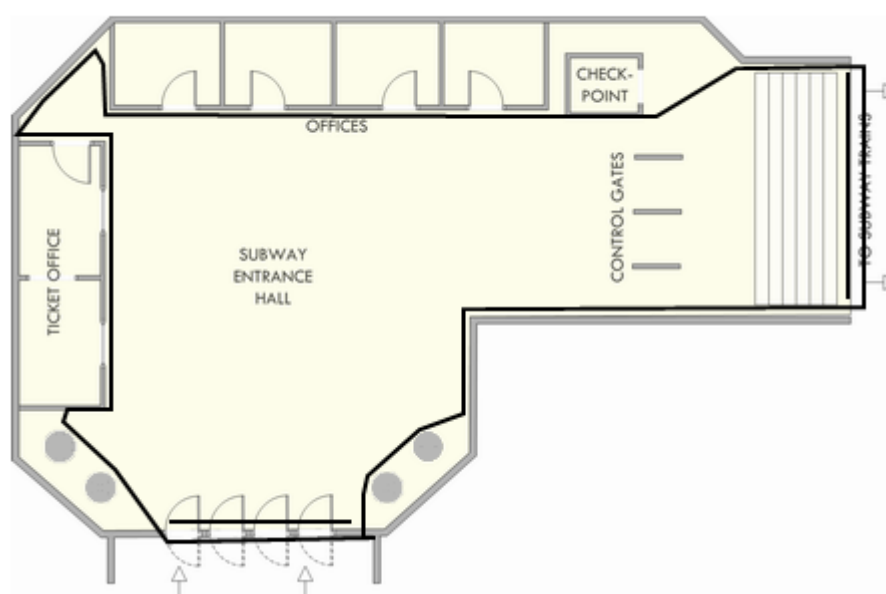


Рис. 4.15. Входная и выходная зона пассажиров

### Разметка турникетов

Что бы смоделировать прохождение пассажиров через турникеты нужно определить их местоположение и задать траектории очередей.

Эти элементы модели задаются в области Contol Gates (см. рисунок 4.16).

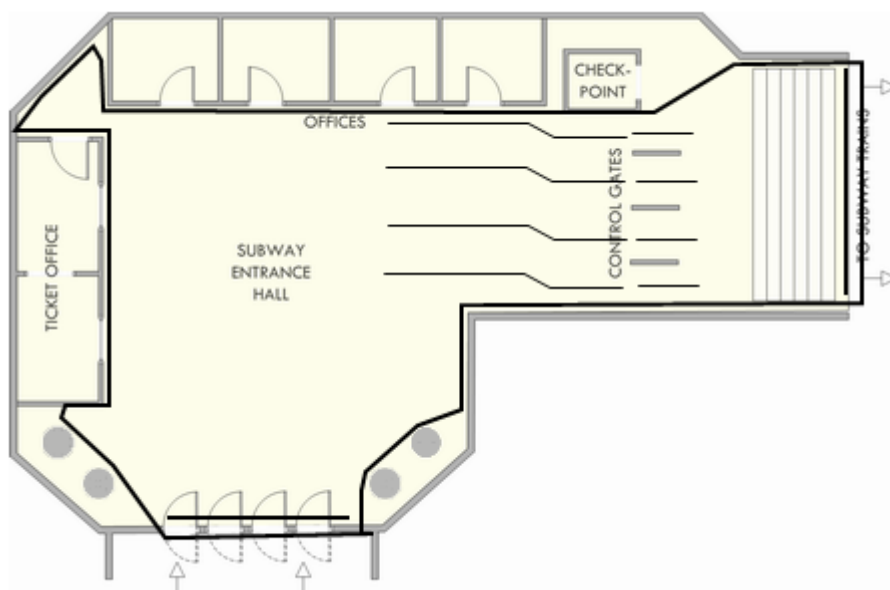


Рис.4.16. Турникеты и очереди

Для задания турникетов разместите четыре линии с помощью элемента «Линия». Сформируйте группу из линий. Группе присвойте идентификатор `gatesGroup`.

Затем задайте траектории очередей к турникетам, используя элемент «Ломанная». Важно: Ломанные нужно рисовать от турникетов ! Соберите ломанные в группу с названием `gatesQGroup`.

После разметки местоположения турникетов и очередей к ним следует указать местоположение окон касс продажи билетов пассажирам и указать траектории очередей. Разметка производится в области TICKET OFFICE, в левой части плана вестибюля (см. рисунок 4.17).

С помощью элемента «Линия» укажите два кассовых окна (вертикальные линии). Объедините их в группу `winGroup`. О кассовых окон проведите две траектории очередей, так как это показано на рисунке 4.17. Объедините их в группу `winQGroup`.



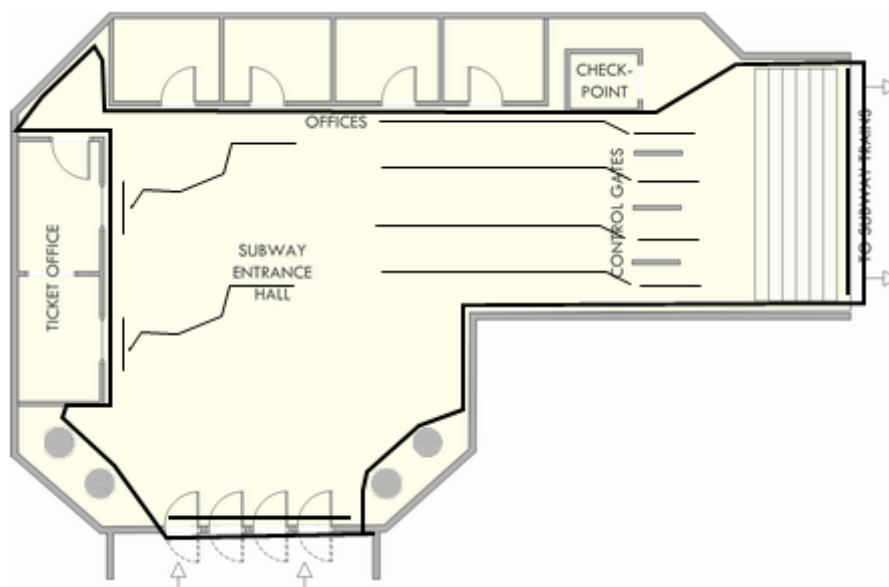


Рис. 4.17. Разметка касс и очередей к ним

### Построение канала СМО

Для моделирования движения пассажиров в вестибюле нужно создать модель СМО, используя элементы палитры Pedestrian Library (см. рис.4.18).

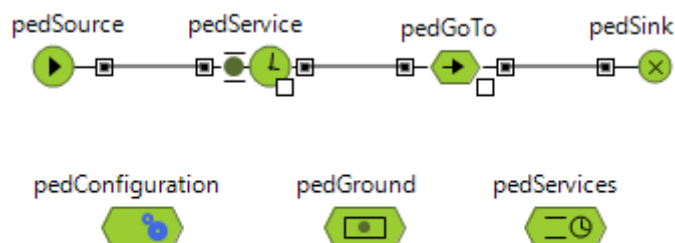


Рис.4.18. СМО Вестибюль метро

Создайте модель СМО в соответствии с рисунком 4.18. Модель состоит из двух частей: канала СМО (верхняя часть) и сервисов (нижняя часть).

Сначала требуется создать сервисы. Сервисы слева на право: PedConfiguration. Задаёт параметры пешеходной модели:

Скрыть фигуры среды = активен

PedGground. Конфигурация здания – вестибюля:

Стены (группа, необязательный) = `group`

`PedServices`. Сервис обслуживания прохождения пассажиров в вестибюле:

Имя = `gates`

Сервисы (группа линий) = `gatesGroup`

Тип сервиса = Протяженный

Время задержки = `uniform(2.0*second(), 3.0*second())`

Очереди(группа линий, ломанных) = `gatesQGroup`

Правило выбора очереди = Самая короткая

Обслуживается = Самая длинная очередь

Далее следует настроить канал СМО.

`PedSource`. Элемент выполняет генерацию заявок – пассажиров:

Объект создает = Пешеходов

Пешеходы пребывают согласно = Интенсивности

Расписание пешеходов в ед. времени = `1000/hour()` (В час пребывает 1000 пешеходов)

Этаж (`PedGround`) = `pedGround`

Место появления (линия,ломанная) = `entry`

`PedService`. Определяет используемый сервис модели:

Сервис (`PedServices`) = `gates`

`PedGoTo`. Элемент используется для задания направления движения. Поток пассажиров должен двигаться к выходу из вестибюля:

Цель (точка,линия) = `exit`

`PedSink`. Элемент, уничтожающий заявки. В данной модели все параметры элемента остаются по умолчанию.

Настройте эксперимент модели:

Модельное время – минуты, конечное время не задано. Скорость прорисовки презентации равна 1.

Вид работающей модели показан на рисунке 4.19.

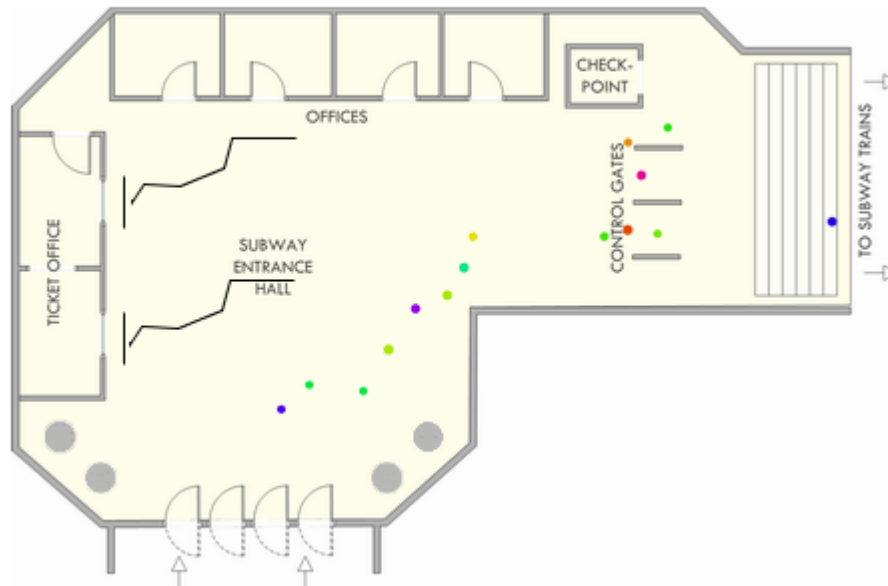


Рис.4.19. Тестирование модели движения пассажиров

Из результатов моделирования видно, что пассажиры проходят в вестибюль, минуя кассы направляясь к турникетам.

### Моделирование покупки билетов

Для моделирования процесса покупки билетов требуется изменить модель в соответствии с рисунком 4.20.

В модель нужно добавить новый сервис PedServices:

Имя = windows

Сервисы (группа линий) = winGroup

Тип сервиса = Точечный

Время задержки = `triangular(15*second(), 25*second(), 35*second())`

Очереди (группа линий, ломаны)= `winQGroup`

Правило выбора очереди = Самая короткая очередь

Обслуживается = Самая длинная очередь

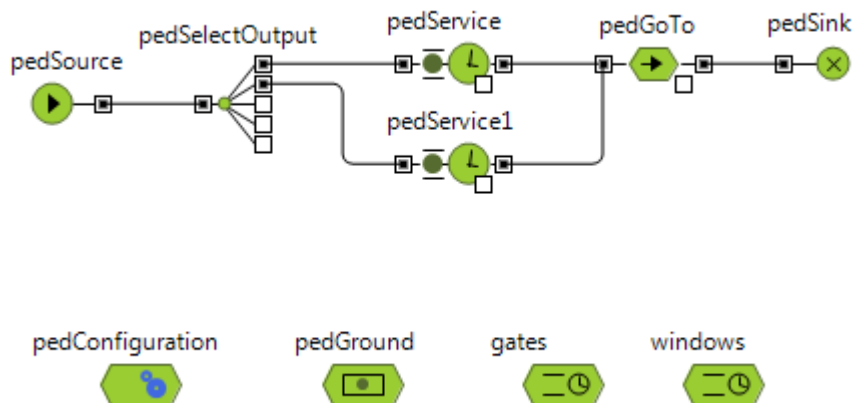


Рис. 4.20. СМО с двумя каналами

Далее следует в СМО добавить второй канал для подключения нового сервиса. Для этого в СМО вводится элемент PedSelectOutput, для переключения на нужный канал. Переключение выполняется в соответствии со значением коэффициента предпочтения каналов:

Коэфф. Предпочтения 1 = 85

Коэфф. Предпочтения 2 = 15

Во втором канале покупки билетов нужно настроить элемент PedService:

Сервис (PedServices) = windows

Действие при выходе = `ped.setColor(Color.red)`

В результате заявка – пассажир после покупки билета в кассе будет окрашена в красный цвет.

Выполните тестирование модели. Вид работающей модели показан на рисунке 4.21.

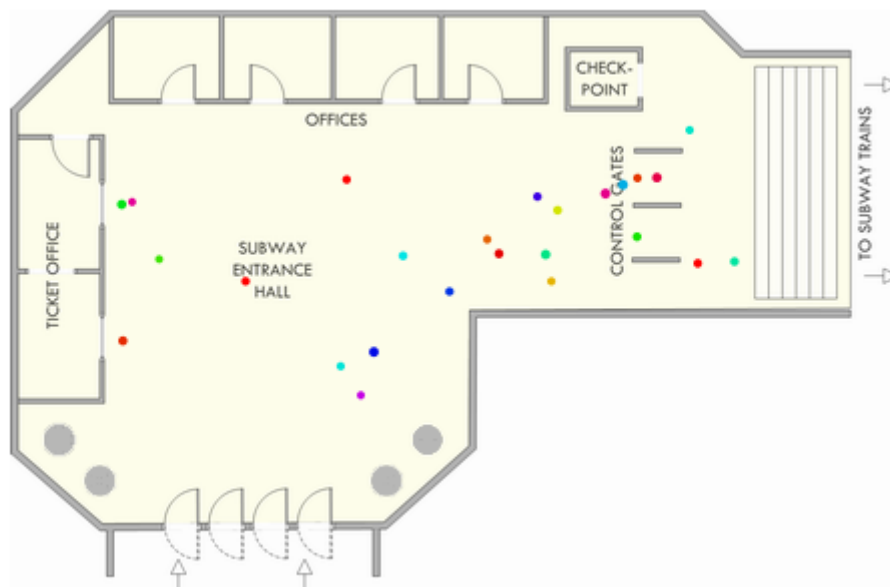


Рис.4.21. Моделирование покупки билетов

### Самостоятельное задание

1. Создайте эксперимент для изучения влияния интенсивности входа пассажиров в вестибюль метро. Создайте интерфейс эксперимента с ползунком для изменения интенсивности потока пассажиров от нуля 5000 пассажиров в час. Для отображения выбранного значения используйте элемент «Текст».

Методические указания:

Интенсивность пассажиропотока определяется значением свойства `rate` элемента `PedSource`.

2. Постройте модель отображающую движение работников фирмы в вестибюле. Пример схемы движения приведен на рис. 4.21.

Известно, что 75% входящих в вестибюль являются сотрудниками фирмы, а 25% это посетители, которые должны получить пропуск. Интенсивность входного людского потока – 1000 человек в час.

Посетители после получения пропуска сразу проходят через турникеты.

3. Внесите изменения в модель задания №2 таким образом, что бы посетители взявшие пропуск, становились в очередь вместе с сотрудниками.

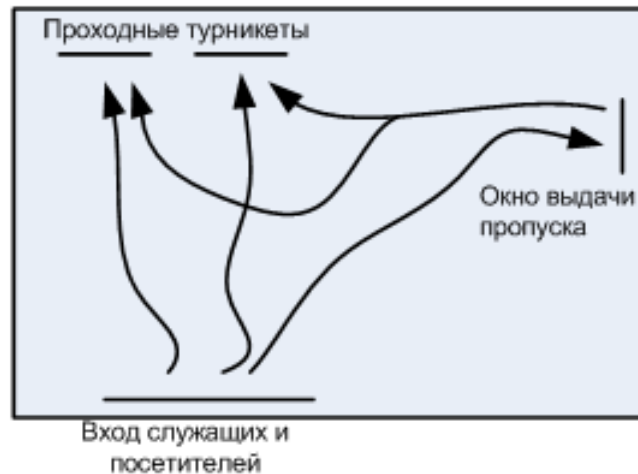


Рис.4.21. Схема движения

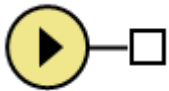
### Контрольные вопросы

1. Перечислите основные особенности создания сетевой модели в AnyLogic.
2. Дайте классификацию ресурсов сети AnyLogic.
3. Перечислите основные элементы AnyLogic необходимые для построения канала СМО при сетевом моделировании и дайте их краткую характеристику.
4. Какие элементы AnyLogic и для чего нужно ввести в модель канала СМО для моделирования процесса захвата и возврата ресурсов?
5. Перечислите основные этапы создания пешеходной СМО в AnyLogic.
6. Перечислите особенности процесса анимации движения пешеходов в пешеходной СМО.

7. Перечислите основные элементы AnyLogic, с краткой характеристикой, необходимые для создания пешеходной СМО.
8. Как строится модель пешеходной СМО в которой имеется несколько каналов обслуживания заявок – пешеходов?

## Элементы библиотеки Enterprise Library

### Source



#### Назначение:

Генерация заявок на вход СМО. Класс заявки: по умолчанию Entity.

#### Методы:

`void inject(int n)`. Создает `n` заявок.

`int count()`. Возвращает количество заявок, созданных объектом.

#### Свойства:

`int arrivalType` (Способ генерации заявок). Заявки могут пребывать следующими способами:

С заданной интенсивностью. Интенсивность эквивалентна экспоненциальному закону распределения времени между прибытиями заявок со средним значением, равным  $1/\text{интенсивность}$ .

Путем задания времени между прибытиями заявок. Время между двумя последовательными прибытиями определяется заданным выражением. Этот режим рекомендуется использовать для периодической генерации заявок или для генерации заявок с интервалами времени, не подчиняющихся экспоненциальному закону распределения.

`int entitiesPerArrival`. Количество заявок, прибывающих за один раз. Свойство доступно, если заявки прибывают согласно интенсивности или по времени между прибытиями. Значение по умолчанию равно одной заявке.

Фигура анимации заявки. Фигура анимации, с помощью которой заявки, создаваемые этим объектом, будут отображаться на презентации.

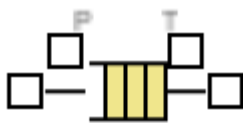


`double rate` (Интенсивность прибытия заявок). Свойство доступно, если заявки прибывают согласно интенсивности. Значение по умолчанию равно 1.

`Entity newEntity` (Новая заявка). Значение по умолчанию равно `new Entity()`

`void onExit` (Действие при выходе). Код, выполняемый при генерации заявки в канал СМО.

## Queue



Назначение:

Моделирование очереди заявок.

Методы:

`int size()`. Возвращает количество заявок, находящихся в данный момент в очереди.

`boolean canEnter()` . Возвращает `true`, если в очередь может быть добавлена еще одна заявка.

Свойства:

`int capacity` (Емкость очереди).

`boolean enableTimeout` (Разрешить уход по таймауту).

`double timeout` (Таймаут) .

`void onExitTimeout` (Действие при уходе по таймауту).

`boolean enablePreemption` (Разрешить вытеснение).

`double priority` (Приоритет заявки).

`void onExitPreempted` (Действие при вытеснении).

`void onEnter` (Действие при входе). Код, выполняемый, когда заявка поступает в объект.

`void onExit` (Действие при выходе). Код, выполняемый, когда заявка покидает объект.

## Delay



Назначение:

## Моделирование процессора обслуживания.

### Методы:

`int size()`. Возвращает количество заявок, задерживаемых объектом в текущий момент времени.

`boolean canEnter()` . Возвращает `true`, если объект может принять еще одну заявку.

### Свойства:

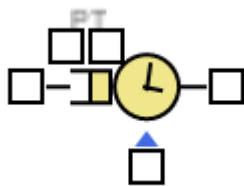
`int capacity`. Вместимость объекта Delay. Задаёт максимальное количество заявок, которое может одновременно находиться в объекте. Значение по умолчанию равно одной заявке.

`void onEnter` (Действие при входе). Код, выполняемый, когда заявка поступает в объект.

`void onExit` (Действие при выходе). Код, выполняемый, когда заявка покидает объект.

`double delayTime` (Время задержки заявки).

## Service



### Назначение:

Моделирование очереди с процессором обслуживания с учетом использованных ресурсов.

### Методы:

`int queueSize()` . Возвращает количество заявок во вложенном объекте Queue.

`int delaySize()`. Возвращает количество заявок во вложенном объекте Delay.

### Свойства:

`int quantity` (Количество ресурсов). Выражение, вычисляющее количество ресурсов, необходимое текущей заявке. Значение по умолчанию равно одному.

`void onEnterDelay` (Действие при начале задержки). Код, выполняемый, когда заявка поступает во вложенный объект Delay.

`ResourcePool resourcePool` (Объект `ResourcePool`). Выражение, возвращающее имя объекта `ResourcePool`, задающего ресурсы, которые требуются заявке. Если поле пусто, или если выражение будет возвращать `null`, то будут использоваться ресурсы объекта `ResourcePool`, соединенного с портом `access` этого объекта `Service`. Значение по умолчанию `null`.

`int quantity` (Количество ресурсов, захваченных заявкой).

`double delayTime` (Время задержки, вычисленное для заявки)

Объект поддерживает свойства «Действие при входе», «Действие при выходе».

## ResourcePool



Назначение:

Моделирование ресурсов.

Методы:

`int idle()`. Возвращает количество свободных ресурсов.

`int busy()`. Возвращает количество занятых ресурсов.

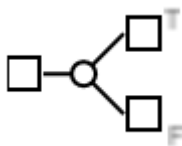
Свойства:

`boolean unitsAreObjects` (Ресурсы моделируются).

Определяет, как моделируются ресурсы: как индивидуальные объекты или просто как их количество. Значение по умолчанию: «Как индивидуальные объекты (`true`)». Если ресурсы выделяются в виде количества их нельзя отобразить на презентации.

`int capacity` (Количество ресурсов). Свойство доступно если количество ресурсов задано напрямую. Значение по умолчанию равно одному.

## SelectOutput



Назначение:

Используется для выбора канала СМО при моделировании.

Свойства:

`boolean conditionIsProbabilistic` (Выход true выбирается ).  
Определяет, как будет производиться маршрутизация заявок: будут ли заявки направляться на выход true (верхний порт `outT`) при выполнении условия, заданного в поле «Условие» или же случайно с заданной вероятностью, определенной в поле «Вероятность [0..1]».

`boolean condition` (Условие). Свойство доступно, если выход true выбирается при выполнении условия. Условие, вычисляемое для входящей заявки. Если оно выполняется (равно true), то заявка покидает объект через порт `outT`, если нет - через порт `outF`. Значение по умолчанию `randomTrue( 0.5 )` просто разделяет поток заявок на две равные части.

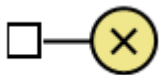
`code onEnter` (Действие при входе ). Код, выполняемый, когда заявка поступает в объект. Синтаксис:

`code onExitTrue` (Действие при выходе true). Код, выполняемый, когда заявка покидает объект через порт `outT`.

`code onExitFalse` (Действие при выходе false).

Код, выполняемый, когда заявка покидает объект через порт `outF`.

## Sink



Назначение:

Уничтожение заявок, после их выхода из канала СМО.

Метод:

`int count()` - возвращает количество заявок, уничтоженных объектом Sink.

Свойства:

`void onEnter` (Действие при входе). Код, выполняемый при поступлении заявки в объект.

## Netwok



Назначение:

Используется при моделировании транспортных сетей. Задаёт топологию сети и управляет сетевыми ресурсами. В одной модели

может быть несколько сетей, и каждая сеть задается одним объектом `Network`. Топология сети задается группой фигур: прямоугольники задают узлы сети, а линии и ломаные - сегменты.

Элемент содержит порт `NetworkResourceAccessPort` `access`, который должен быть соединен с портами объектов `NetworkResourcePool`.

Свойства:

`ShapeGroup networkGroup` (Группа фигур сети). Группа фигур анимации (для которых разрешено программное управление), которые графически задают топологию сети. Группа может содержать прямоугольники (задающие узлы сети), линии и ломаные линии (задающие сегменты).

`boolean hideNetwork` (Прятать фигуры сети). Значение по умолчанию равно `true`. Если значение равно `true`, то фигуры, используемые для задания топологии сети, не будут отображаться на презентации во время выполнения модели.

`boolean drawAtRandomPosition` (Когда объект в узле, то рисовать его). Определяет, как будут отображаться анимации заявок и ресурсов, находящихся в узлах сети и не движущиеся: будут ли они рисоваться в случайном месте внутри узла, в верхнем левом углу узла или в центре узла. Значение по умолчанию равно `true`.

## NetworkResourcePool



Назначение:

Задает набор сетевых ресурсов транспортной сети определенного типа. Единственный порт объекта должен быть соединен с портом объекта `Network`.

Свойства:

`int type` (Тип ресурса). Определяет характер вводимого ресурса: статический, движущийся или переносной. Значение по умолчанию: «Движущийся».

`int capacityDefinitionType` (Количество задано). Определяет, как задано количество ресурсов: напрямую численным значением, табличной функцией, или фигурой базового местоположения. В

последнем случае количество ресурсов будет равно количеству точек ломаной линии, соединяющей узлы базового местоположения. Значение по умолчанию «Напрямую»

`int capacity` (Количество ресурсов). Свойство доступно, если количество задано напрямую. Значение по умолчанию равно 1 .

`Shape idleUnitShape` (Фигура анимации свободного ресурса). Это

фигура, которая на презентации будет отображаться свободный ресурс.

`Shape busyUnitShape` (Фигура анимации занятого ресурса). Это фигура, которой на презентации будет отображаться занятый (захваченный заявкой) ресурс.

`int homeShapeType` (Базовое местоположение задается как). Свойство определяет, как задается базовое местоположение ресурса: как один узел или как ломаная линия, соединяющая несколько узлов.

`Rectangle homeNode` (Базовый узел). Свойство доступно, если базовое местоположение задается как один узел. В этом случае используется прямоугольник, задающий узел сети, который будет играть роль базового местоположения этих ресурсов в сети. Значение по умолчанию: «Один узел».

`PolyLine homePath` (Путь через узлы). Свойство доступно, если базовое местоположение задается как путь через узлы. В модели сети должна быть ломаная линия с точками, лежащими внутри узлов, которые будут играть роль узлов базового местоположения ресурсов в сети. При этом количество точек этой ломаной также может задавать количество ресурсов данного типа.

### NetworkEnter



Назначение:

Регистрирует заявку в сети и помещает ее в заданный узел сети. После добавления в сеть заявка может перемещаться по сети и использовать сетевые ресурсы. Заявка не может одновременно находиться сразу в нескольких сетях, поэтому перед добавлением в

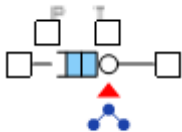
другую сеть она должна быть вначале удалена из текущей сети с помощью объекта `NetworkExit`.

Свойства:

`Network network` (Сеть). Имя объекта `Network`, задающего сеть, в которую будут добавляться заявки.

`Rectangle destinationNode` (Узел входа). Имя прямоугольника, задающего узел сети, в который будут попадать заявки при добавлении в сеть.

### Networksize



Назначение:

Объект захватывает для заявки заданное количество сетевых ресурсов. При необходимости может пересылать захваченные ресурсы в заданное место сети и (опять же, при необходимости) присоединять их к заявке. Вначале ресурсы запрашиваются для первой заявки из очереди, и пока эта заявка не захватит ресурсы (или не покинет объект по какой-либо другой причине), ресурсы для последующих заявок не выделяются даже если они и могли бы быть выделены.

Порты:

`Port in` (Входной порт).

`Port outTimeout` (Выходной порт для заявок, покидающих объект по вследствие истечения заданного времени ожидания).

`Port outPreempted` (Выходной порт для заявок, покидающих объект в результате вытеснения).

`Port out` (Выходной порт).

Свойства:

`NetworkResourcePool[] resources` (Список ресурсов). Задается в виде: `{pool1, pool2, ...}`. Это имена объектов `NetworkResourcePool`, задающих те сетевые ресурсы, которые будут захватываться данным объектом. Указав имя объекта один раз, заявка захватит один ресурс того типа, который задается этим

объектом. Поэтому, например, если нужно захватить два ресурса одного и того же типа список нужно составить в виде:

```
{doctor,nurse,nurse}.
```

### NetworkSendTo



Назначение:

Перемещает сетевые ресурсы из их текущего местоположения в заданный узел сети. Могут перемещаться только движущиеся ресурсы или переносные ресурсы в сопровождении движущихся ресурсов.

Свойства:

`NetworkResourcePool[] resources` (Список ресурсов).

Перечень ресурсов задается в виде: `{pool1, pool2, ...}`.

`int destinationType` (Отсылать ресурсы). Определяет, будет ли этот объект отсылать ресурсы в заданный узел, к заявке, к захваченному ресурсу или в базовое местоположение захваченного ресурса.

по умолчанию: «В заданный узел».

Имя объекта `NetworkResourcePool`. Задаёт тип того ресурса, к которому (или к чьему базовому местоположению) будут пересылаться захваченные ресурсы. Если захвачено несколько ресурсов одного типа, то будет выбираться первый ресурс из списка. Свойство доступно, если объект будет отсылать ресурсы к захваченному ресурсу или в базовое местоположение захваченного ресурса.

### NetworkAttach



Назначение:

Присоединяет к заявке указанные сетевые ресурсы. После присоединения ресурсы будут перемещаться вместе с заявкой (сопровождать заявку) до того времени, пока они не будут отсоединены или освобождены заявкой. На момент



присоединения ресурсы должны быть уже захвачены заявкой и располагаться в том же узле сети, что и сама заявка.

Свойство:

`boolean attachAll` (Присоединять). Определяет, будет ли этот объект присоединять Все захваченные ресурсы в месте нахождения заявки или заданные ресурсы, указанные в свойстве «Список ресурсов».

Значение по умолчанию равно `false` т.е. присоединяются заданные ресурсы.

### NetworkMoveTo



Назначение:

Задаёт движение заявки.

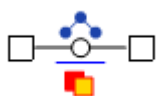
Свойства:

`boolean destinationIsNode` (Заявка перемещается). Определяет, будет ли заявка двигаться в заданный узел или к захваченному ресурсу. Значение по умолчанию : «Движение в заданный узел.

`NetworkResourcePool destinationResource` (Ресурс).

Свойство доступно, если заявка перемещается к захваченному ресурсу. Если захвачено несколько ресурсов одного типа, то будет выбираться первый ресурс из списка.

### NetworkDetach



Назначение:

Отсоединяет от заявки, ранее присоединенные сетевые ресурсы. Отсоединенные ресурсы будут по-прежнему захвачены заявкой, но уже не будут сопровождать заявку при ее перемещении. Операция отсоединения занимает нулевое время.

Свойство:

`boolean detachAll` (Отсоединять). Определяет, будет ли этот объект отсоединять все захваченные ресурсы в месте нахождения

заявки или заданные ресурсы, указанные в поле Список ресурсов. Значение по умолчанию равно `false` : «Заданные ресурсы».

### NetworkRelease



Назначение:

Освобождает все или какие-то определенные сетевые ресурсы, ранее захваченные заявкой с помощью объекта `NetworkSeize`. В случае освобождения каких-то определенных ресурсов, они выбираются из общего числа захваченных ресурсов согласно заданному списку ресурсов. Вся операция занимает нулевое время.

Свойства:

`boolean releaseAll` (Освободить). Определяет, будет ли данный объект освобождать все захваченные ресурсы или только определенные заданные ресурсы, указанные списке ресурсов.

Значение по умолчанию равно `false`: «Заданные ресурсы».

`boolean movingGoHome` (Движущиеся ресурсы).

Определяет, должны ли освобождаемые движущиеся ресурсы возвращаться в базовое местоположение или продолжать оставаться в том месте, где они были освобождены.

Значение по умолчанию равно `true`: «Возвращаются в базовое местоположение».

### NetworkExit



Назначение:

Удаляет заявку из сети. Заявка при этом перестает отображаться на анимации сети. При удалении заявки из модели с помощью объекта `Sink` она обязательно должна быть предварительно удалена из сети.

Свойство:

`code onExit` (Действие при выходе). Код, выполняемый, когда заявка покидает объект.

## Элементы библиотеки Pedestrian Library

### PedSource



#### Назначение:

Генерация заявок – пешеходов и или групп пешеходов.

#### Свойства:

`int arrivalType` (Пешеходы прибывают согласно). Свойство доступно если объект создает заявку - пешеход. Определяет, будут ли пешеходы прибывать согласно:

интенсивности эквивалентной экспоненциально распределенному времени между прибытиями со средним значением, равным  $1/\text{интенсивность}$ ;

времени между прибытиями заявок заданных выражением. таблице интенсивностей в которой указано как интенсивность прибытия пешеходов изменяется с течением времени; расписанию, заданному в виде табличной функции, в которой заданы времена прибытия пешеходов и количество пешеходов, прибывающее в каждый указанный момент времени;

вызовам метода `inject()`.

Значение по умолчанию: «Прибытие согласно интенсивности»

`double rate` (Интенсивность прибытия, пешеходов в единицу времени). Свойство доступно если объект создает пешеходов, и пешеходы прибывают согласно интенсивности. Значение по умолчанию равно `1000/hour()`.

`PedGround ground` (Этаж для создаваемых пешеходов - `PedGround`). Объект, задающий этаж, на котором появляются пешеходы.

`Shape location` (Место появления: линия, ломаная). Линия, на которой появляется пешеход в моделируемой среде. Это может быть линия или ломаная.

## PedService



### Назначение:

Направляет поток пешеходов через группу сервисов и очередей, заданных в объекте PedServices. Сервис может быть указан как двунаправленный, тогда пешеходы смогут проходить сервис в любом направлении.

### Порты:

`in` (Входной порт).

`OutPortPush out` (Выходной порт, через который пешеходы покидают объект в случае успешного обслуживания).

`OutPortPush ccl` (Выходной порт, через который пешеходы покидают объект в случае события "отмены").

### Свойства:

`PedServices services` (Сервис). Задаёт сервис, в котором будут обслужены пешеходы, проходящие через этот блок. Значение по умолчанию `null`.

`boolean reverse` (В обратном направлении). Если значение равно `true`, то пешеход будет проходить через этот сервис в обратном направлении. Значение по умолчанию `false`.

`void onEnter` (**Действие при входе**). Код, который выполняется, когда пешеход поступает в объект.

`void onExit` (Действие при выходе). Код, который выполняется, когда пешеход покидает объект через порт `out`.

`void onCancel` (Действие при отмене). Код, который выполняется, когда пешеход покидает объект через порт `ccl`.

### Методы:

`int size()` - Возвращает количество пешеходов, находящихся в этом объекте.

`void cancel(T ped)` - Заставляет заданного пешехода немедленно покинуть блок через порт `ccl`. Для пешехода выполняется код свойства "Действие при отмене".

`void cancelAll()` - Заставляет всех пешеходов немедленно покинуть блок через порт `ccl`. Для каждого пешехода выполняется код параметра "Действие при отмене".

## PedGoTo



**Назначение:**

Заставляет пешеходов перейти в заданное место моделируемого пространства, которое может быть задано линией или точкой. Переход будет считаться выполненным, когда пешеход пересечет заданную линию или достигнет заданной точки. Пешеходы будут искать путь к заданному транзиту в пределах текущего этажа.

**Порт:**

`Port ccl`. Выходной порт, через который пешеходы покидают объект в случае события "отмены".

`Shape target` (Цель: точка, линия). Фигура линия или точка, задающая место назначения движения пешехода.

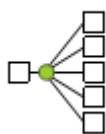
**Методы:**

`void cancel(T ped)` - Заставляет заданного пешехода немедленно покинуть блок через порт `ccl`.

`void cancelAll()` - Заставляет всех пешеходов немедленно покинуть блок через порт `ccl`.

`int size()` - Возвращает количество пешеходов, находящихся внутри объекта.

## PedSelectOutput



**Назначение:**

Выбор порта в соответствии с заданными весовыми коэффициентами предпочтения - вероятностями. Например, если значение коэффициента предпочтения равно 85, то это означает выбор порта с вероятностью 0,85.

Порт может так же выбираться по условию. Условия вычисляются последовательно от первого порта до пятого.

**Свойство:**

Выходной порт будет выбираться:

либо в соответствии с заданными коэффициентами предпочтения,

либо в зависимости от того, для какого из этих портов будет выполнено заданное условие.

## PedSink



**Назначение:**

Удаляет поступивших в объект пешеходов из моделируемой среды. Обычно объект используется в качестве конечной точки блок-схемы, формализующей поток пешеходов. Объект автоматически ведет подсчет пешеходов. Также объект можно использовать для удаления любого пешехода.

**Свойство:**

`void onEnter` (Действие при входе). Код, выполняемый, когда пешеход попадает в среду.

## PedConfiguration



**Назначение:**

Поддерживает перемещение пешеходов и их анимацию. Позволяет задавать общие параметры, относящиеся ко всем объектам Pedestrian Library, и настраивать модель для конкретной задачи с целью получения максимальной производительности. Его присутствие в модели обязательно.

**Свойство:**

`boolean hideEnvironmentShapes` (Скрыть фигуры среды). Если значение равно `true`, то на анимации не будут отображаться нарисованные пользователем объекты среды (стены, границы областей, сервисы и т.п.) Значение можно менять во время исполнения модели. Значение по умолчанию равно `false`.

## PedGround



**Назначение:**

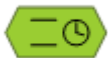
Позволяет задавать двумерное пространство в моделируемой среде, представляющее собой «этаж», т.е. поверхность, по которой

будут перемещаться пешеходы. Этажи могут быть ограничены какой-то стеной или быть неограниченными. Стены - это объекты, которые пешеходы не могут пересекать. Стены являются частью этажа, то есть одна стена не может быть использована несколькими этажами.

Свойства:

`Group walls` (Стены группа, необязательный). Стены, заданные на данном этаже. После запуска модели объект `PedGround` анализирует заданный набор стен и создает информацию о прохождении этажа с точки зрения транзитных линий и возможных путей. Процедура анализа конфигурации стен запускается только один раз, при запуске модели. Изменение этого параметра в процессе исполнения модели будет проигнорировано. Допустимое значение: группа, содержащая одну или несколько линий, ломаных, прямоугольников. Ломанная линия должна быть замкнута. Если в моделируемой области нет стен, то данный параметр можно опустить. Значение по умолчанию - `null`.

### **PedServices**



Назначение:

Объект задает группу одинаковых физических объектов обслуживания (например, несколько турникетов или автоматов по продаже билетов). Объект позволяет задавать очереди и сервисы в любой комбинации и задавать правила выбора сервисов.

Свойства:

`int serviceType` (Тип сервиса). Сервис может быть:

Протяженный - пешеходы двигаются от начальной точки линии сервиса к конечной.

Точечный сервис - пешеходы проводят определенное время в какой-то точке сервиса. Значение по умолчанию: Точечный

`Group services` (Сервисы - группа линий). Группа, содержащая фигуры (линии), задающие сервисы.

`Group queues` (Очереди - группа линий, ломаных). Группа, содержащая линии и ломаные, задающие геометрию очередей.  
`double delay` (Время задержки). Задержка сервиса,

задается в единицах модельного времени. Время, в течение которого пешеход стоит у точки входа в сервис протяженного типа, или в точке точечного сервиса.

Значение по умолчанию: `uniform(2.0*second(), 3.0*second())`

`int queueChoicePolicy` (Правило выбора очереди). Задает, должен ли пешеход выбрать самую короткую очередь, ближайшую очередь, или пользователь должен в ручном режиме задать очередь индивидуально для каждого пешехода в свойстве «Выбрать очередь (линия, ломаная)». Значение по умолчанию: «Самая короткая очередь»

`int servePedsFrom` (Обслуживается). Определяет, будет ли сервис обслуживать пешеходов из самой длинной очереди, ближайшей очереди, или пользователь должен самостоятельно в ручном режиме выбрать очередь, из которой нужно обслуживать пешехода, используя свойство «Обслуживать очередь (линия, ломаная)». Значение по умолчанию: «Ближайшая очередь».



## Сбор статистики

### Объект "Данные гистограммы" (HistogramData)

Методы:

`public abstract void add(double val)` - добавление значения в набор данных гистограммы;  
`public int count()` - число данных в наборе;  
`public double mean()` - среднее значение набора данных;  
`public double max()` - максимальное значение набора;  
`public double min()` - минимальное значение набора;  
`public double deviation()` – стандартное квадратичное отклонение набора данных.

### Объект Очередь (Queue)

Объект поддерживает свойство `statsSize`. С помощью этого свойства можно вызвать методы для вычисления статистических показателей очереди:

`max()` – максимальное число заявок в очереди;  
`min()` - минимальное число заявок;  
`mean()` - среднее число заявок;  
`variance()` – дисперсия.

Пример:

```
double var;  
  
var = queue.statsSize.variance();
```

Обращение к этим методам возможно только в том случае, если для объекта включен режим сбора статистики с помощью соответствующего свойства.

## Объект Delay

Сбор статистики для этого объекта выполняется путем обращения к свойству `statsUtilization`.

Вызывая соответствующие методы можно определить временные характеристики: максимальное и минимальное время задержки, среднее время задержки, дисперсию времен задержки заявки.

Пример:

```
double mean;  
mean=delay.statsUtilization.mean();
```

Обращение к методам определения параметров статистики возможно только в том случае, если для объекта включен режим сбора статистики с помощью соответствующего свойства.

### Функции законов распределения

`double uniform(double min, double max)` - возвращает равномерно распределенное случайное число в диапазоне от `min` до `max`:

$$f(x) = \frac{1}{max - min}$$

Если аргумент `min` опущен, то возвращается случайное число, равномерно распределенное в диапазоне от 0 до значения `max`.

`double triangular(double min, double max, double mode)` - возвращает случайное число соответствующее треугольному закону распределения:

$$f(x) = \begin{cases} \frac{2(x - min)}{(max - min)(mode - min)}, & min < x \leq mode \\ \frac{2(max - x)}{(max - min)(max - mode)}, & mode < x \leq max \end{cases}$$

Здесь `mode` – предпочтительное значение аргумента в диапазоне от `min` до `max`.

`double exponential(double lambda)` - возвращает случайное число, распределенное по экспоненциальному закону распределения:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Здесь `lambda` интенсивность потока заявок или процесса обслуживания.

### **Список литературы**

1. Гнеденко Б.В., Коваленко Н.Н. Введение в теорию массового обслуживания. –М.: Наука, 2007.
2. Карпов Ю. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. – Спб.: БХВ С.-Петербург, 2005.
3. Осипов Л.А. Проектирование систем массового обслуживания. – М.: «Адвансед Солюшнз», 2011.
4. Советов Б.Я., Яковлев С.А. Моделирование систем: Учебник. – М.: Высшая школа, 2009.