Пешеходное моделирование. Модель аэропорта

Давайте создадим модель небольшого аэропорта с двумя гейтами. Пассажиры будут прибывать в аэропорт и регистрироваться на рейс (если они не сделали этого заблаговременно через интернет). Затем все пассажиры должны будут пройти процедуру предполетного досмотра, после чего они смогут направиться в зону ожидания перед гейтами, дожидаясь начала посадки на свой рейс. При объявлении начала посадки, пассажиры направляются к соответствующему гейту. У гейта служащие аэропорта проводят проверку посадочных талонов, после чего пассажиры проходят на посадку на самолет.

Процесс создания модели будет разбит на семь фаз . В шестой фазе вы научитесь считывать данные из базы данных (мы считаем информацию о совершаемых рейсах из файла Microsoft Excel).

Фаза 1. Задание потока пешеходов

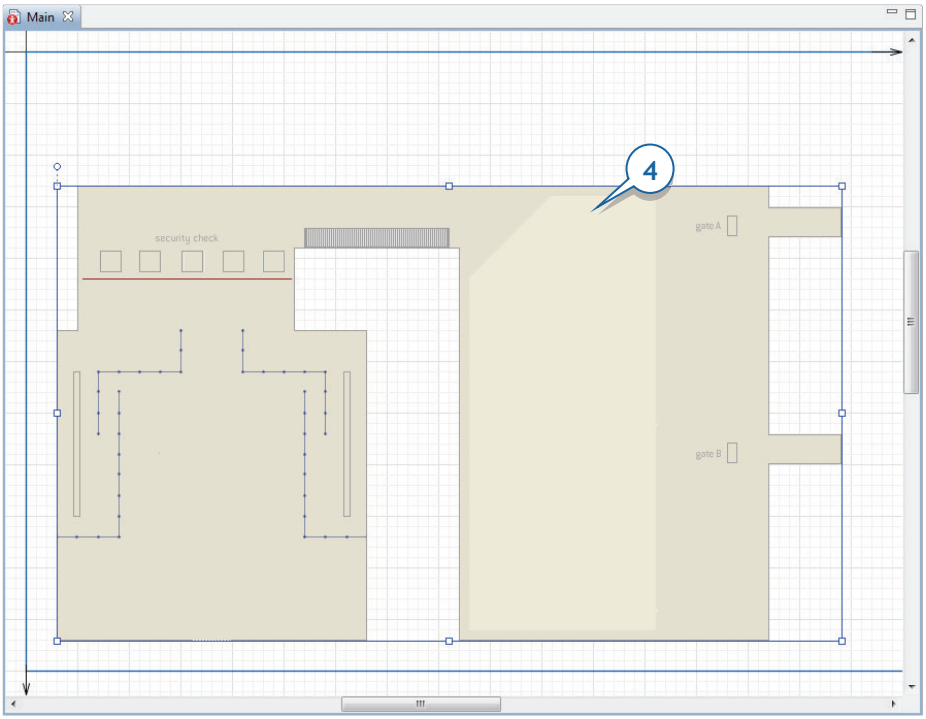
Мы начнем с создания простой модели, в которой пассажиры прибывают в аэропорт и движутся к выходу на посадку. Логику модели мы зададим с помощью блоков *Пешеходной библиотеки* AnyLogic.

Пешеходная библиотека

- Пешеходная библиотека AnyLogic является высокоуровневой библиотекой моделирования движения пешеходов в физическом пространстве. Она позволяет моделировать здания, в которых скапливаются большие количества людей (станции метро, железнодорожные вокзалы, аэропорты, торговые центры, стадионы, музеи и т.д.).
- В моделях, созданных с помощью блоков Пешеходной библиотеки, пешеходы движутся в непрерывном пространстве, реагируя на различные виды препятствий в виде стен и других пешеходов.
- Вы можете создать впечатляющую визуализацию для презентации и валидации вашего проекта, собрать статистику плотности пешеходов в различных областях модели для того, чтобы убедиться, что сервисы смогут справиться с потенциальным ростом нагрузки, вычислить время пребывания пешеходов в каких-то определенных участках модели, выявить возможные проблемы, которые можно решить путем перепланировки здания или добавления дополнительных сервисов, и т.д.

Обычно создание пешеходных моделей начинается с добавления плана моделируемого пространства и рисования стен, обозначенных на плане.

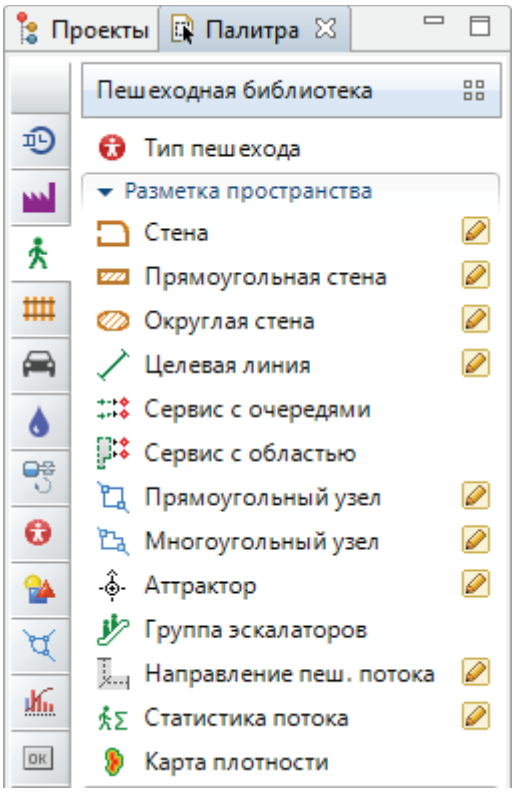
1. Создайте новую модель и назовите ее *Airport*.
2. Добавьте **Изображение**  из палитры **Презентация**  на диаграмму *Main*.
3. Выберите изображение плана аэропорта из файла *terminal.png*, находящегося в каталоге AnyLogic */resources/AnyLogic in 3 days/Airport.*



4. Поместите изображение ближе к левому нижнему углу синей рамки на диаграмме *Main*. Эта рамка задает область, которая будет отображаться в окне модели при ее запуске. Если пропорции изображения нарушены, то в свойствах изображения щелкните по кнопке **Восстановить исходный размер**, а затем выберите опцию **Блокировать**, чтобы заблокировать эту фигуру.

Теперь давайте зададим стены, ограничивающие моделируемое пространство. Для этого мы воспользуемся специальными элементами разметки пространства, созданными для пешеходного моделирования в AnyLogic. Все эти фигуры можно найти в палитре **Пешеходная библиотека**, в разделе **Разметка пространства**.

Элементы разметки пространства для пешеходных моделей



Секция Разметка пространства палитры Пешеходная библиотека

Обычно создание модели начинается с рисования стен поверх имеющегося плана моделируемого помещения. Стены в пешеходном моделировании представляют собой объекты, через которые пешеходы не могут пройти.

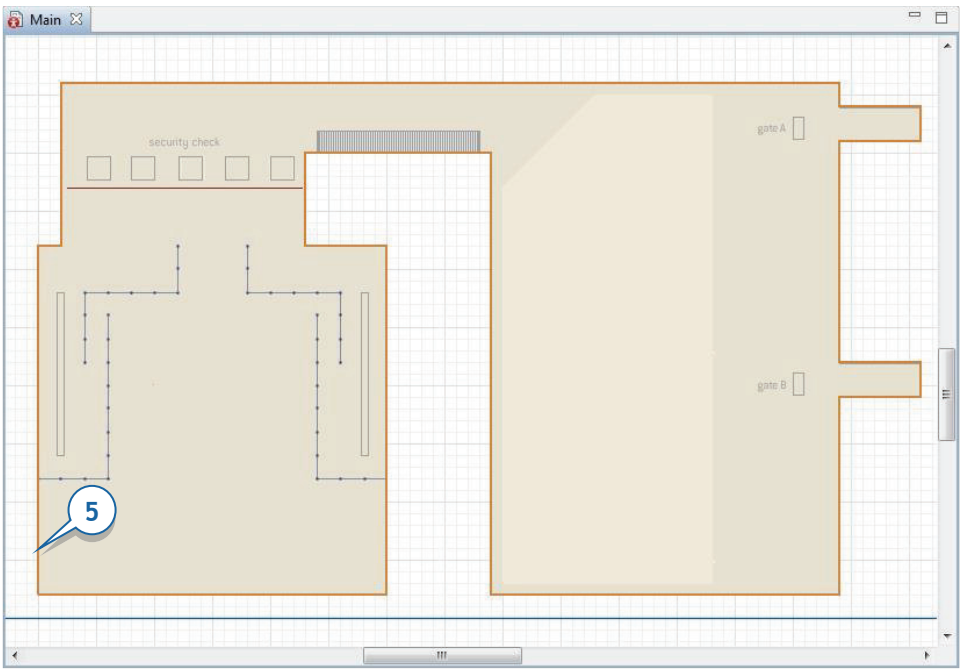
Стены

Стена – Используется для рисования всех внешних и внутренних стен моделируемого помещения.

Прямоугольная стена – Обычно используется для задания прямоугольных помещений внутри заданного стеной пространства, закрытых для пешеходов (служебные помещения и т.д.).

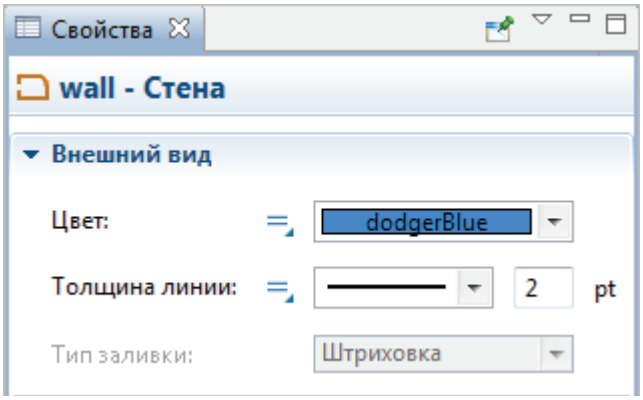
Округлая стена - Используется для задания округлых пространств, недоступных для прохода пешеходов (колонны, фонтаны и т.д.).

5. Нарисуйте стены терминала. Используйте режим рисования: вначале сделайте двойной щелчок по элементу **Стена** в секции **Разметка пространства** палитры **Пешеходная библиотека** , а затем нарисуйте стену на диаграмме, последовательно щелкая мышью в точках изгиба стены и добавив конечную точку двойным щелчком.

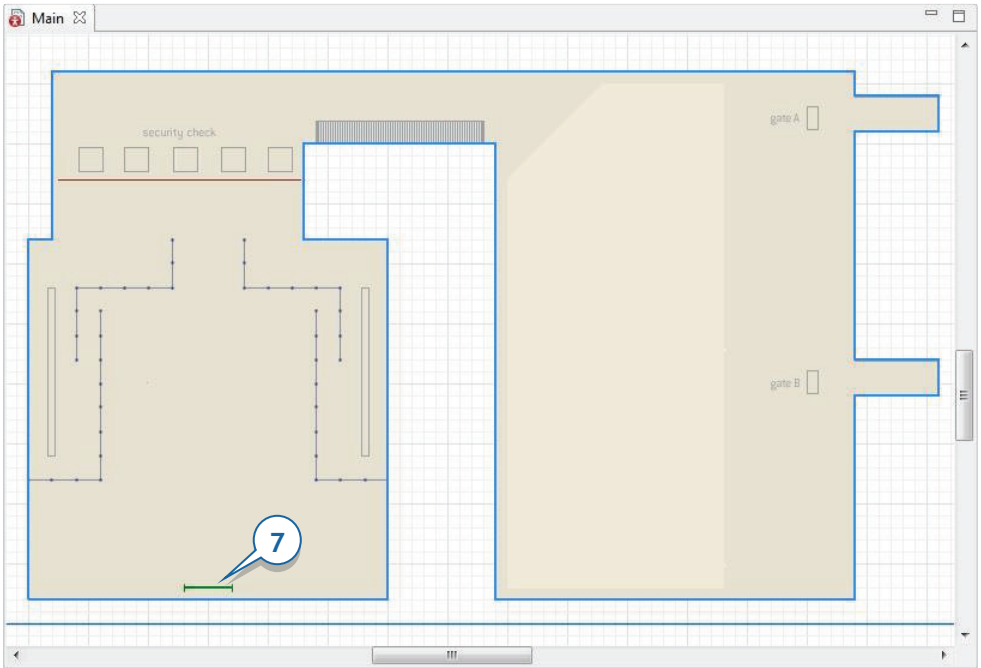


Измените внешний вид стены.

6. Перейдите в секцию свойств стены **Внешний вид** и выберите другой **Цвет**: *dodgerBlue*.



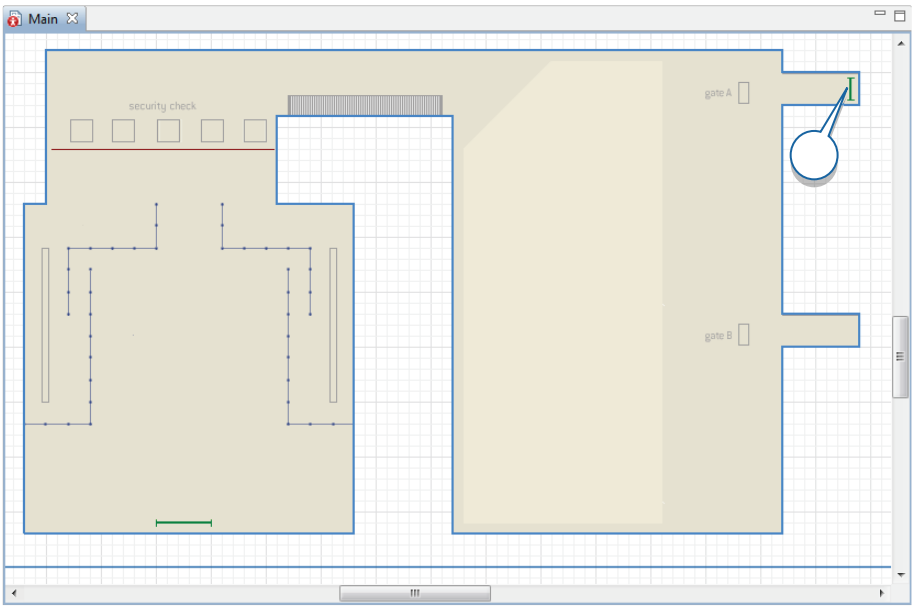
7. Задайте место, в котором будут появляться пешеходы в моделируемом пространстве. Для этого перетащите элемент **Целевая линия** из секции **Разметка пространства** палитры **Пешеходная библиотека** на графическую диаграмму, как это показано на рисунке ниже.



8. Назовите эту целевую линию *arrivalLine*.

9. Добавьте еще одну целевую линию. Поместите ее в области выхода на посадку, как показано на рисунке ниже, и назовите ее *gateLine1*. Прибывшие

в терминал пассажиры будут двигаться к этой линии (мы начнем с простейшей модели, где они просто идут на посадку в гейт).



- **Как линия, на которой будут появляться пешеходы, так и линии, задающие сервисы, очереди к ним и т.д. – все они должны быть помещены внутрь стен, ограничивающих моделируемое пространство, чтобы они были достижимы пешеходами. Иначе во время выполнения модели возникнет ошибка “Target is not reachable” – “Цель недостижима”.**

Целевая линия

Элемент разметки пространства *Целевая линия* используется в пешеходных моделях для задания следующих элементов:

- Место появления пешеходов в моделируемой среде (используется в объектах **PedSource** и **PedEnter**).
- Цель движения пешеходов (используется в объекте **PedGoTo**).
- Место ожидания пешехода (используется в объекте **PedWait**).
- Место выхода с текущего этажа и перехода на новый этаж (используется в объекте **PedChangeLevel**).

Мы нарисовали ключевые объекты моделируемого пространства с помощью фигур разметки пространства. Теперь можно приступить к заданию логики модели с помощью блоков Пешеходной библиотеки.

Задание логики движения потока пешеходов

Все процессы, происходящие в моделируемом пространстве, вы задаете с помощью диаграммы процесса, собираемые из блоков **Пешеходной библиотеки**.

Перечислим самые важные и часто используемые блоки библиотеки:

PedSource – Этот блок создает пешеходов, аналогично тому, как блок **Source** создает агентов в диаграмме, собранной из блоков Библиотеки Моделирования Процессов. Обычно с этого блока начинается рисование диаграммы логики движения пешеходного потока.

PedGoTo – Этот блок моделирует движение пешеходов в заданную точку.

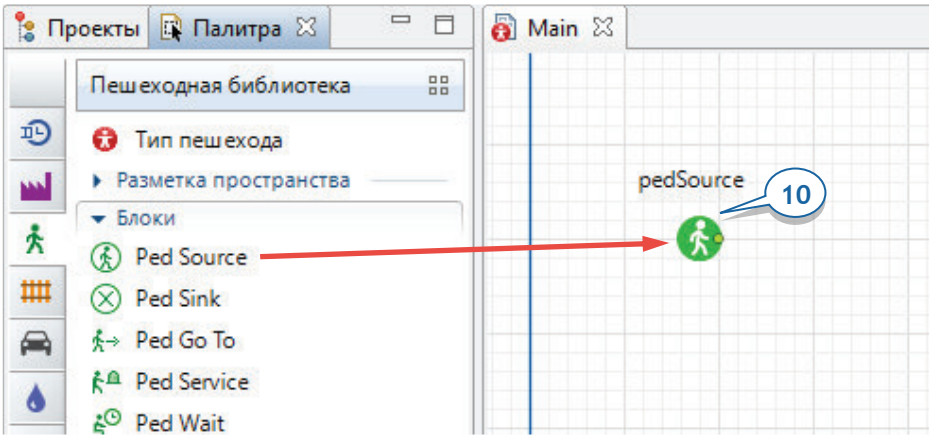
PedService – Этот блок моделирует то, как пешеходы обслуживаются в заданной точке обслуживания (сервисе).

PedWait – Этот блок моделирует то, как пешеходы ждут в течение заданного времени в указанной точке или области.

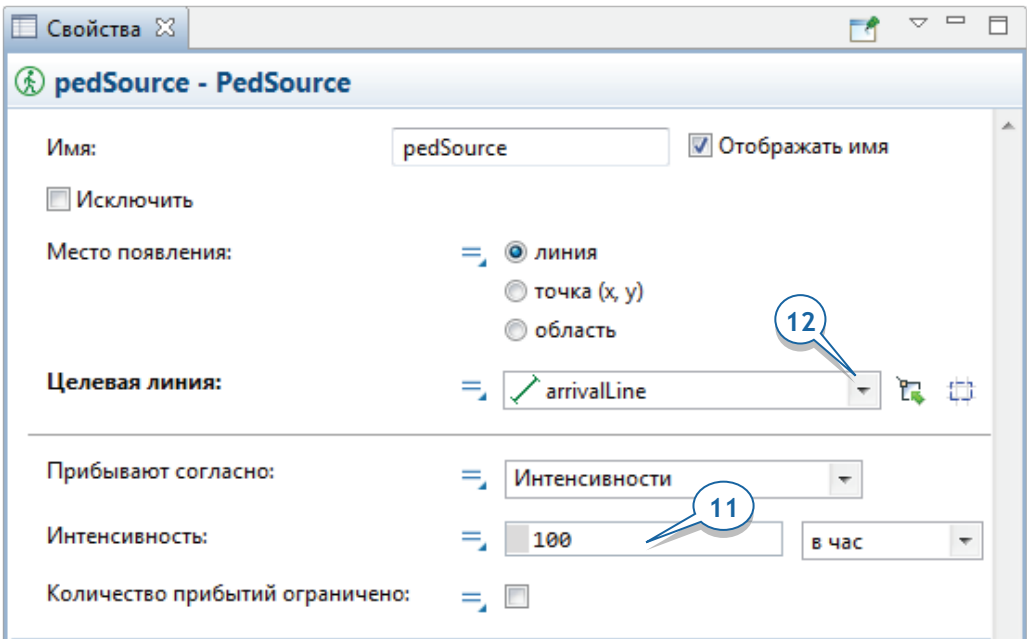
PedSelectOutput – Блок перенаправляет потоки пешеходов в разные подпроцессы в заданных пропорциях, либо же направляет пешеходов с разными характеристиками на выполнение различных действий.

PedSink – Блок удаляет из модели пешеходов, выполнивших все заданные операции. Обычно этот блок завершает диаграмму процесса.

10. Начнем создание диаграммы процесса с добавления на диаграмму *Main* блока **PedSource** из палитры **Пешеходная библиотека**.



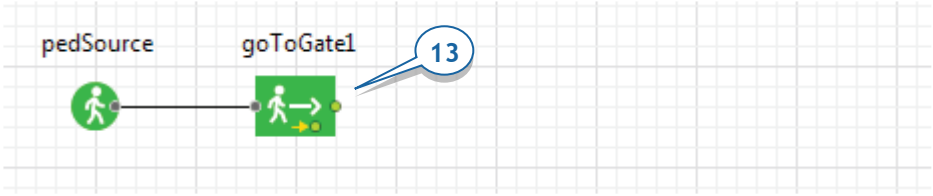
11. Пусть в среднем в терминал прибывает сто пассажиров в час. Откройте свойства блока *pedSource* и введите *100* в поле **Интенсивность**. Укажите блоку **pedSource**, чтобы он добавлял 100 пешеходов **в час**.



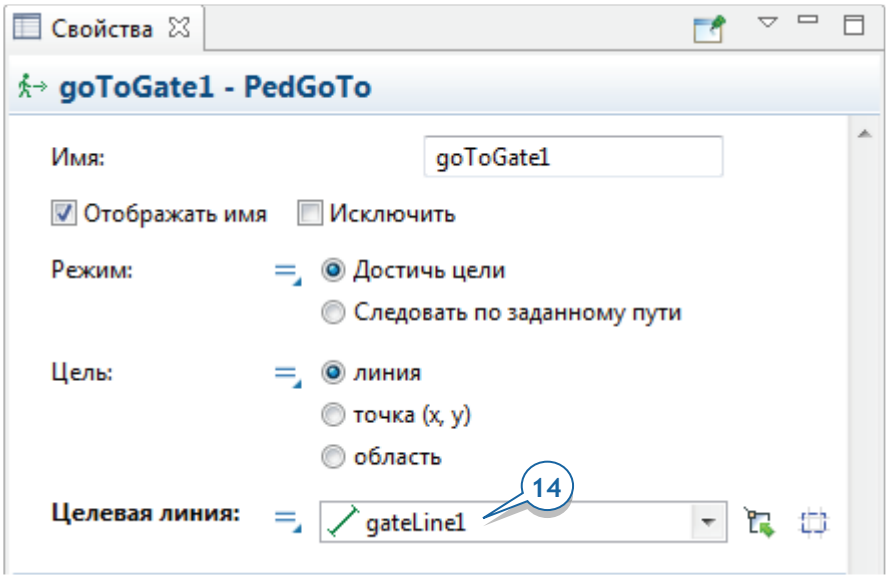
12. Задайте место появления пассажиров в моделируемой среде, выбрав *arrivalLine* в поле **Целевая линия**.

13. Затем добавьте объект **PedGoTo**, моделирующий движение пешеходов к заданному месту, и соедините его с блоком *pedSource*. Поскольку мы хотим,

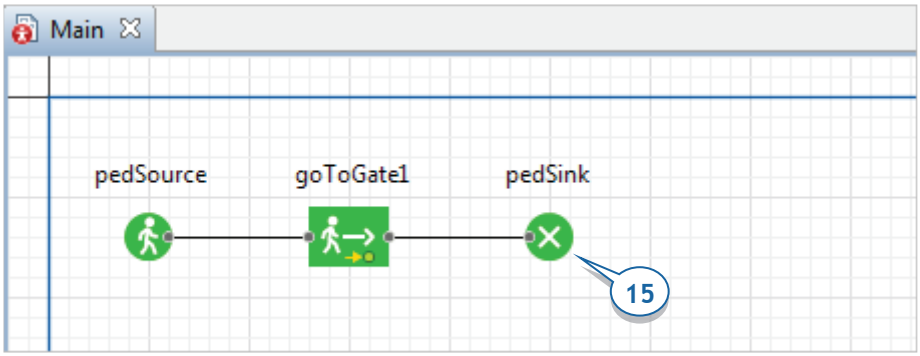
чтобы пассажиры направлялись к первому (верхнему) гейту, назовите объект *goToGate1*.



14. В свойствах этого блока, задайте цель движения пассажиров, выбрав *gateLine1* из списка **Целевая линия**.

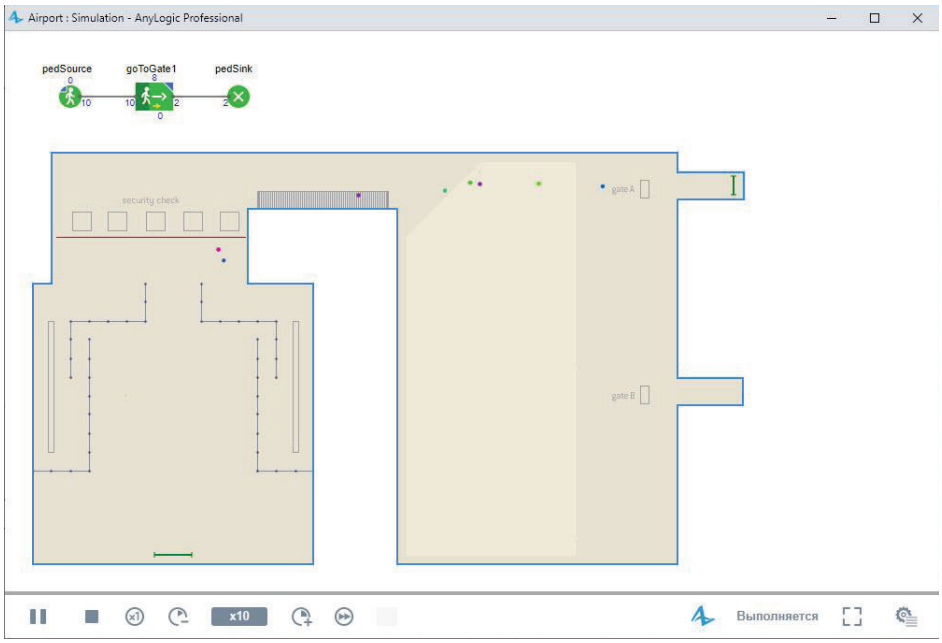


15. Завершите диаграмму блоком **PedSink**  . Он будет удалять из модели поступающих в него пешеходов. Диаграмма пешеходного процесса практически всегда начинается с объекта **PedSource**, а заканчивается объектом **PedSink**.



Ваша диаграмма процесса должна выглядеть так, как на приведенном выше рисунке.

16. Запустите модель. Вы увидите, как пассажиры движутся по аэропорту от входа к заданному выходу на посадку.

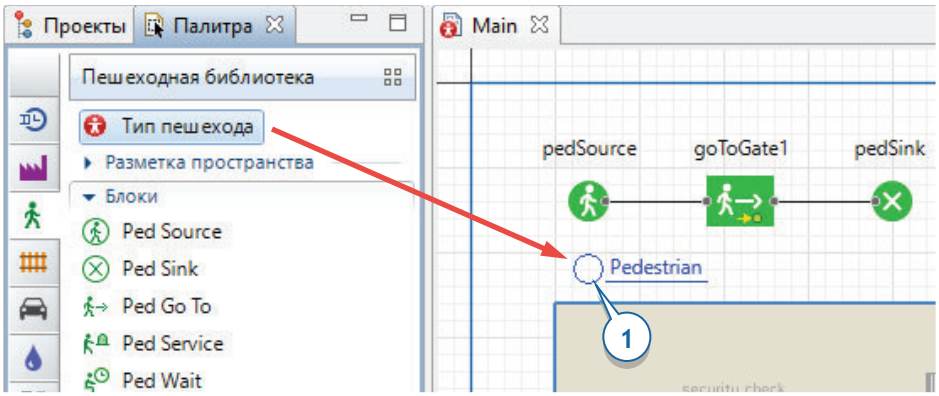


Фаза 2. Создание 3D анимации

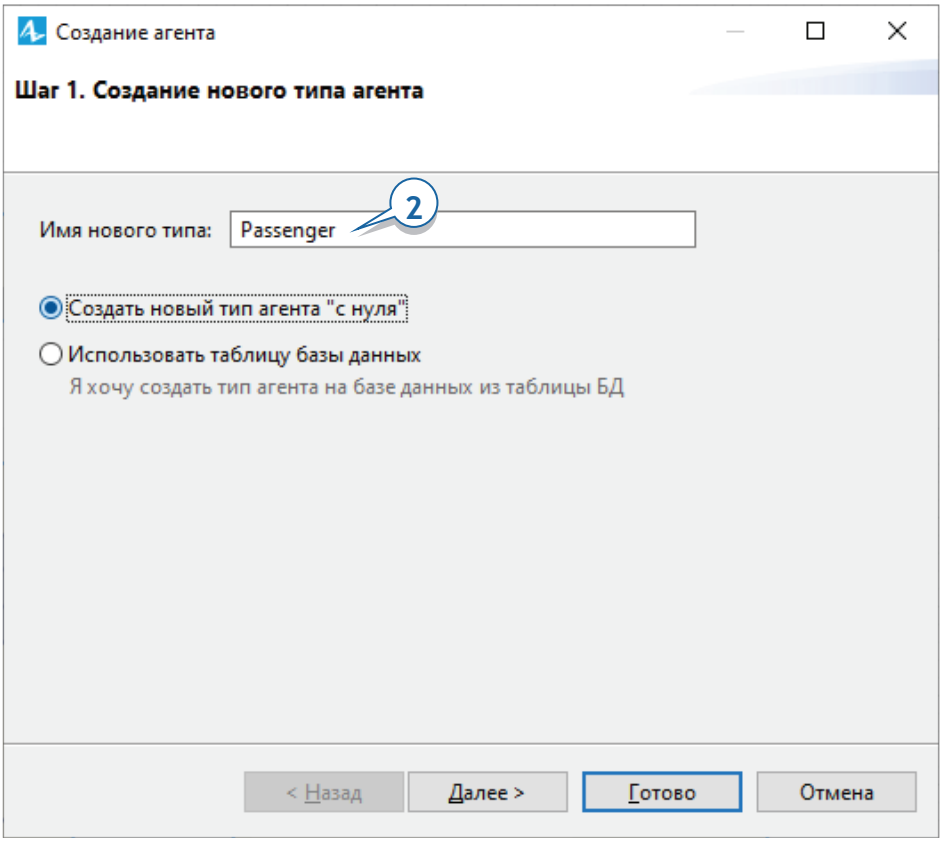
Теперь давайте добавим в нашу модель 3D анимацию. Для этого, как мы уже знаем, нам понадобится добавить 3D окно, камеру и 3D изображение пассажира. Начнем именно с задания трехмерной фигуры анимации, что потребует создания нового **Типа пешехода**.

- **Если вы хотите создать трехмерную анимацию или задать у пешехода специфические характеристики, то для этого потребуется создать тип пешехода. На диаграмме созданного типа вы сможете как нарисовать трехмерную фигуру пешехода, так и задать его характеристики с помощью параметров.**

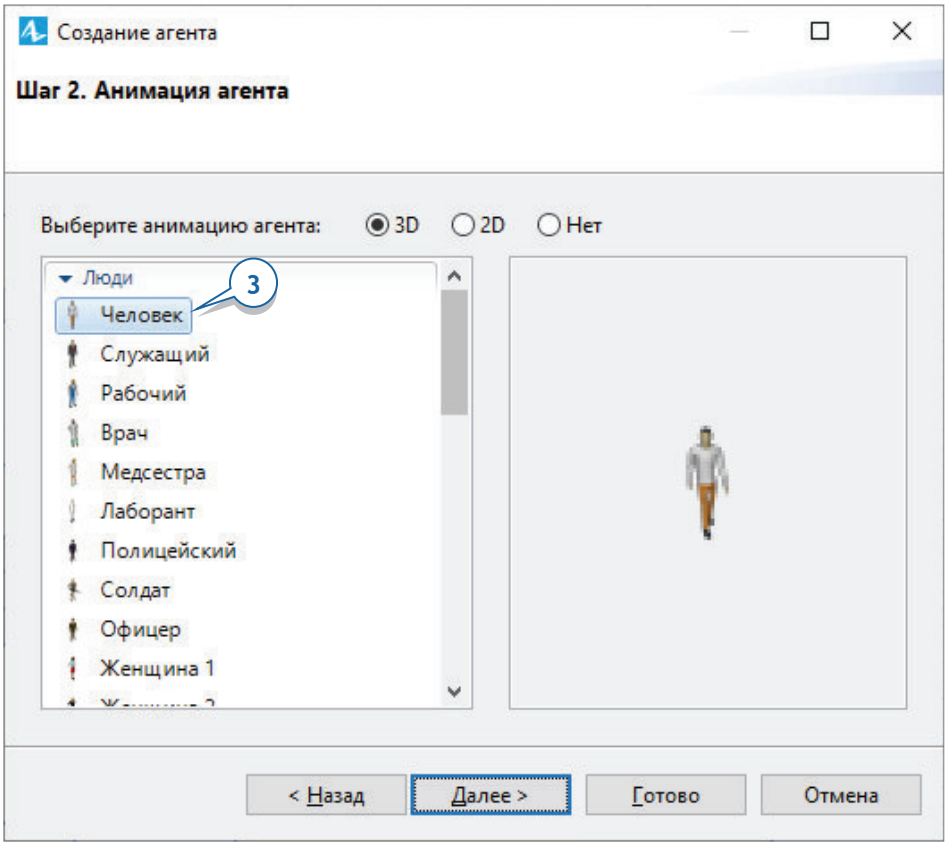
1. Перетащите элемент **Тип пешехода** из палитры **Пешеходная библиотека** на диаграмму *Main*.



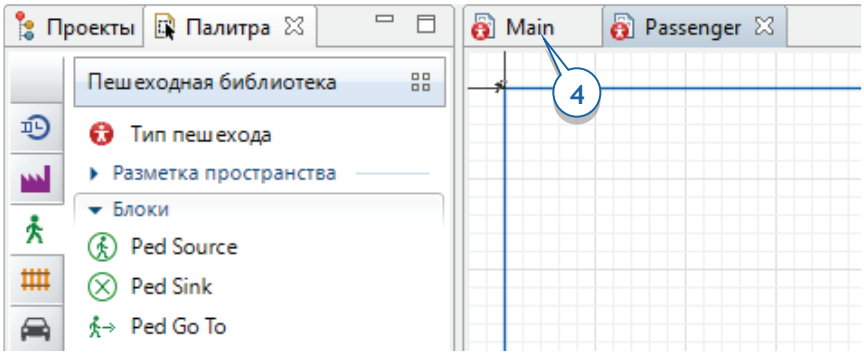
2. На первой странице Мастера создания нового агента, введите имя нового типа: *Passenger*. Щелкните по кнопке **Далее**.



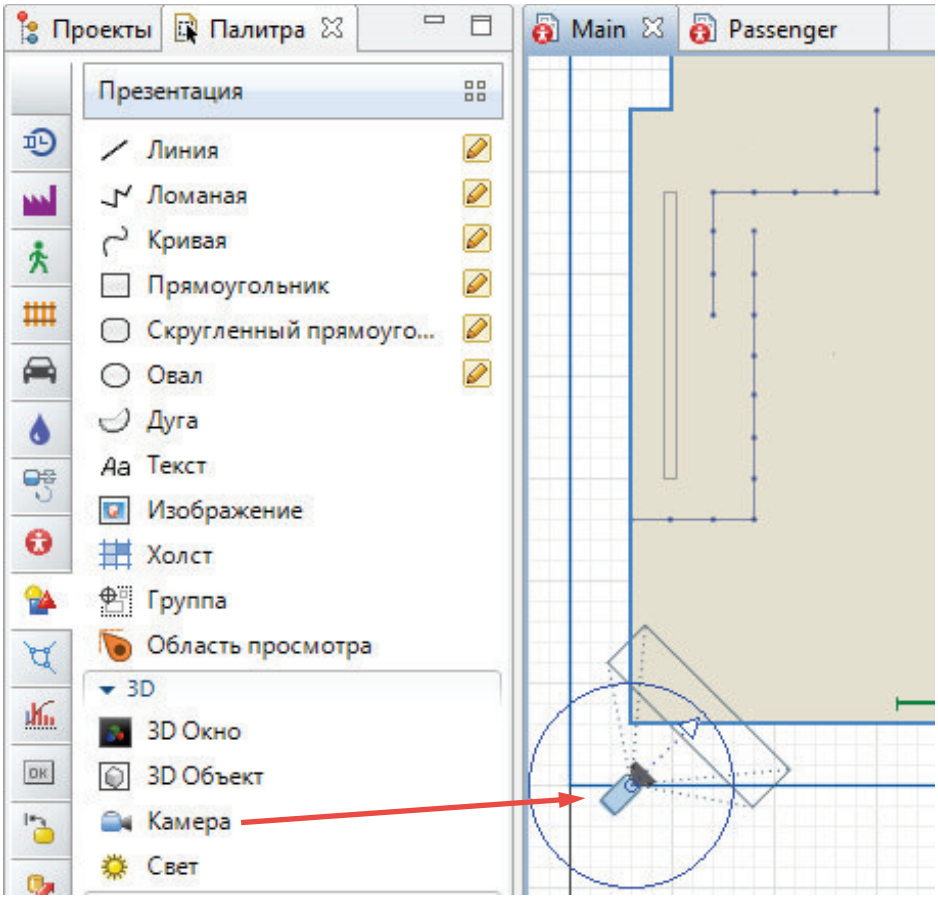
3. На второй странице Мастера оставьте выбранной опцию **3D** и фигуру **Человек**. Щелкните по кнопке **Готово**.



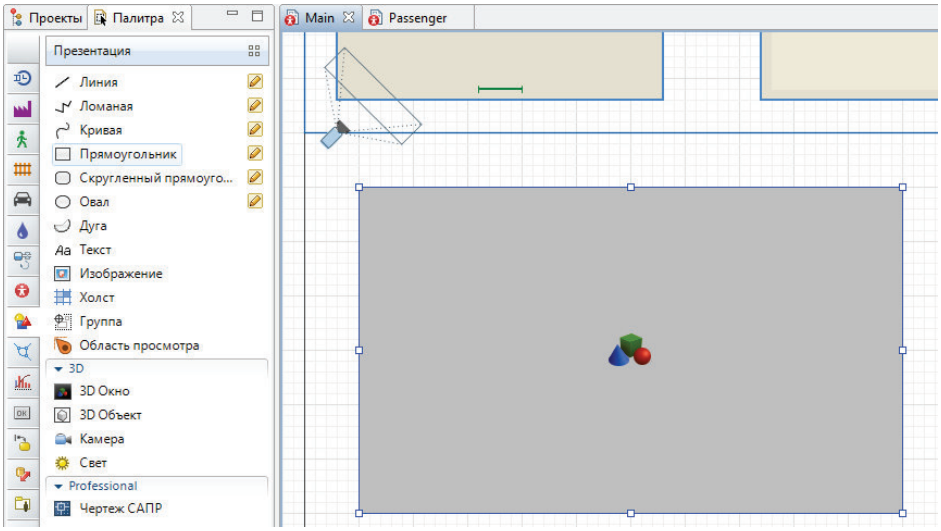
4. AnyLogic откроет диаграмму типа агента *Passenger*. Перейдите обратно на диаграмму *Main*, чтобы продолжить создание модели.



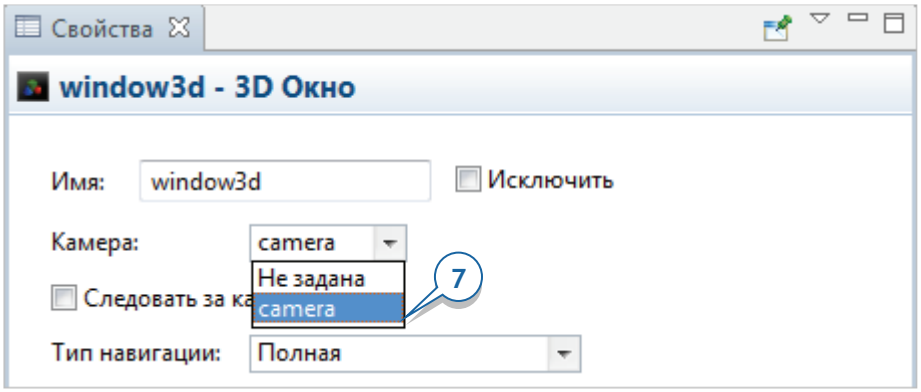
5. Перетащите элемент **Камера** из палитры **Презентация** на диаграмму *Main*. Поместите камеру так, чтобы она была направлена на план терминала (то есть, как бы "снимала" происходящее в терминале).



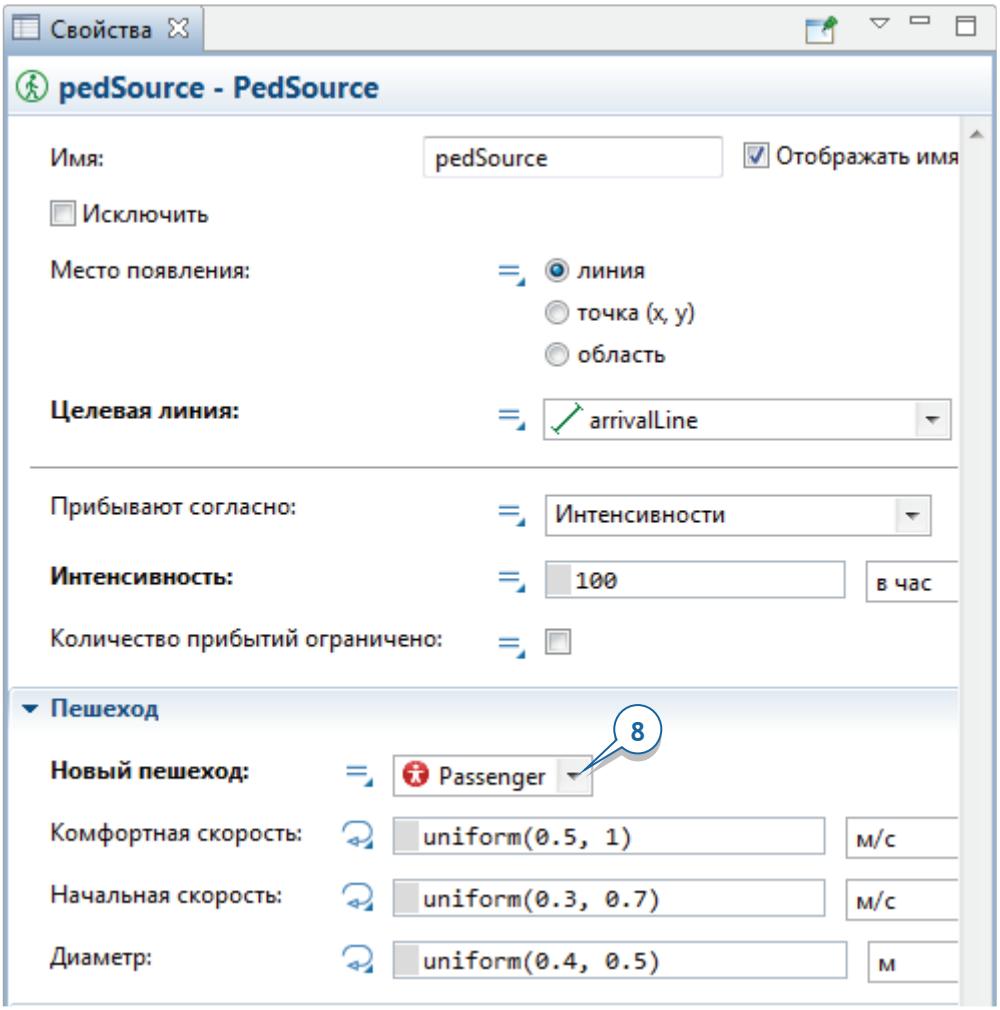
6. Перетащите элемент **3D Окно** из палитры **Презентация** на диаграмму *Main*. Расположите 3D окно под планом терминала, как показано ниже:



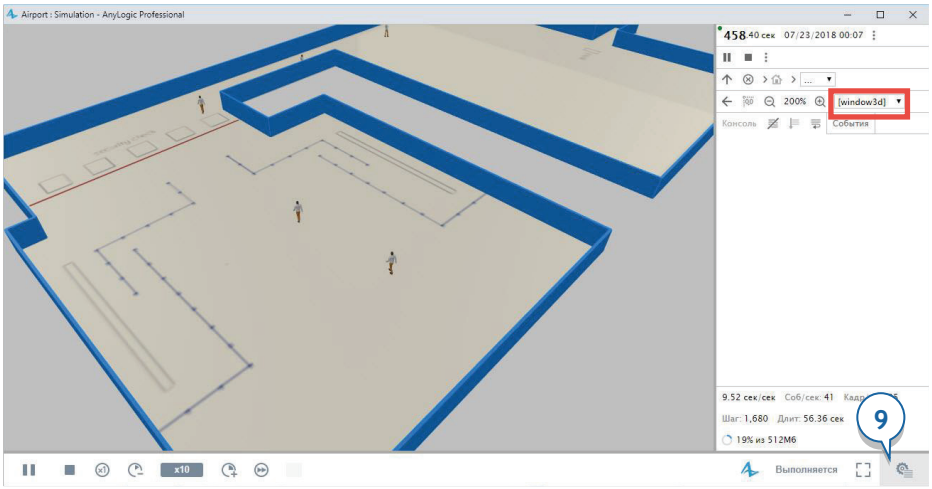
7. Откройте свойства 3D окна и выберите *camera* в свойстве **Камера**.



8. Мы хотим, чтобы блок диаграммы процесса *pedSource* создавал пешеходов созданного нами типа *Passenger*. Откройте свойства блока *pedSource* и выберите *Passenger* из списка **Новый пешеход**, расположенного в разделе свойств **Пешеход**.

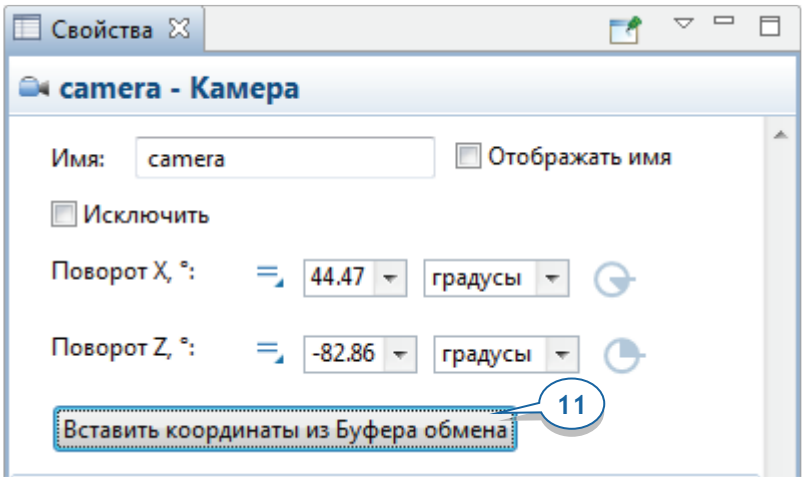


9. Запустите модель. Вы увидите, как пассажиры идут внутри здания, следуя от входа к выходу на посадку. Вы можете переключиться на 3D анимацию, щелкнув по кнопке панели разработчика **Выбрать область и показать** и выбрав опцию **[window3d]**.



10. Перемещайтесь по сцене трехмерной анимации до тех пор, пока не найдете наилучшее расположение камеры. Тогда щелкните правой кнопкой мыши по 3D сцене (Mac OS: Ctrl+щелчок) и выберите из контекстного меню команду **Копировать положение камеры**.

11. Закройте окно презентации и откройте свойства камеры. Примените оптимальное расположение камеры, щелкнув по кнопке **Вставить координаты из Буфера обмена**.



Если вы не знаете, где расположена камера на холсте графической диаграммы, то ее можно легко найти в панели **Проекты** (в ветви **Презентация** агента *Main*).

12. Снова запустите модель и удостоверьтесь, что теперь 3D анимация отображается с учетом заданного расположения камеры.

Фаза 3. Моделирование предполетного досмотра пассажиров

Теперь мы можем начать моделирование процессов, происходящих в аэропорту.

Давайте начнем с моделирования процедуры предполетного досмотра пассажиров. Для этого мы добавим в нашу модель пункты досмотра пассажиров. С точки зрения терминологии пешеходного моделирования, пункт досмотра является *сервисом* – здесь пассажиры должны быть обслужены, а если пункт досмотра в данный момент занят, то пассажирам приходится ждать в очереди, пока он не освободится.

Сервисы в пешеходных моделях

В пешеходном моделировании объекты, в которых пешеходы проводят определенное время для выполнения какой-либо операции/процедуры, называются *сервисами*. Примеры сервисов: турникеты, кассы, автоматы по продаже билетов/напитков и т.д.

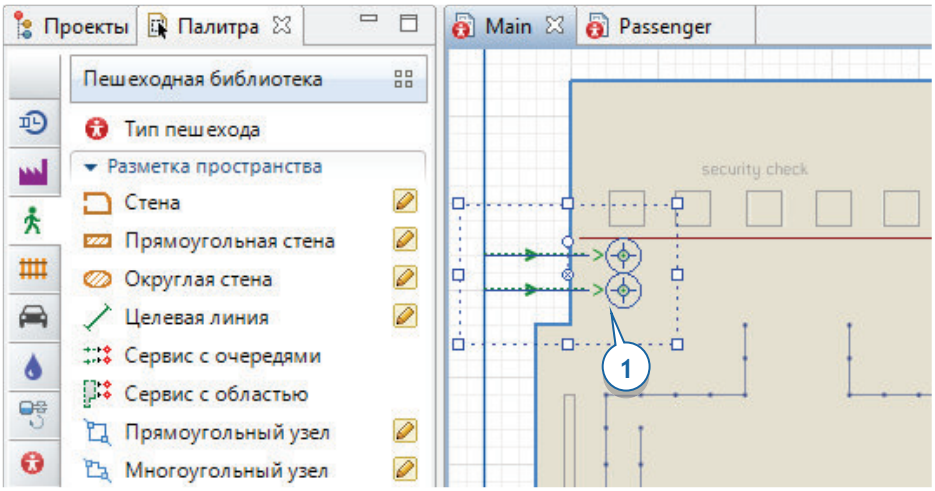
Задание процесса обслуживания состоит из двух шагов. Вначале с помощью фигур разметки пространства **Сервис с очередями** или **Сервис с областью** вы задаете, где в моделируемом пространстве располагаются точки сервиса и очереди к ним.

-  **Сервис с очередями** – С помощью этой фигуры разметки пространства вы можете задать те сервисы, при ожидании доступа к которым люди стоят в очередях (турникеты, билетные кассы и т.д.).
-  **Сервис с областью** – Эта фигура используется для задания сервисов с электронной очередью, когда люди получают номерки и ждут своей очереди, располагаясь в прилегающей области ожидания. Электронная очередь применяется все чаще в банковских отделениях, информационных пунктах на вокзалах и т.д.

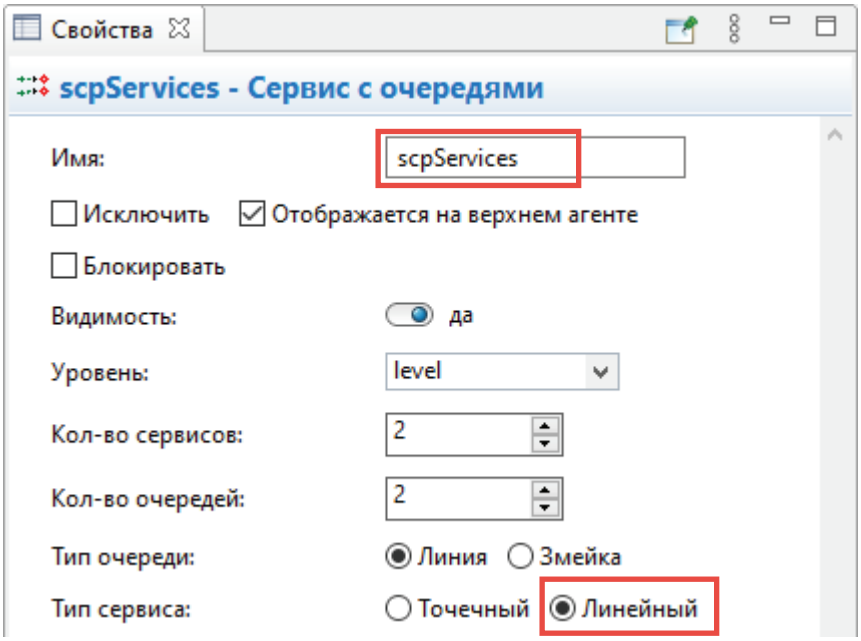
Задав сервисы графически, нужно добавить в диаграмму процесса блок **PedService**, в свойствах которого следует указать соответствующую фигуру сервиса и задать время прохождения процедуры в данном пункте обслуживания.

В моделируемом нами терминале находятся пять пунктов досмотра, поэтому нам нужно будет добавить пять пунктов сервиса, к каждому из которых должна вести очередь.

1. Перетащите элемент **Сервис с очередями** из палитры **Пешеходная библиотека** на диаграмму и поместите его поверх плана терминала аэропорта. По умолчанию этот элемент имеет две точки сервиса и две линии очереди, ведущие к этим сервисам.



2. Откройте свойства элемента **Сервис с очередями**, назовите эту фигуру *scpServices* (*scp* – сокращение от *security check points*) и смените **Тип сервиса** на **Линейный**.



После того, как вы смените тип сервиса с точечного на линейный, точки обслуживания сменят свою форму на линии.

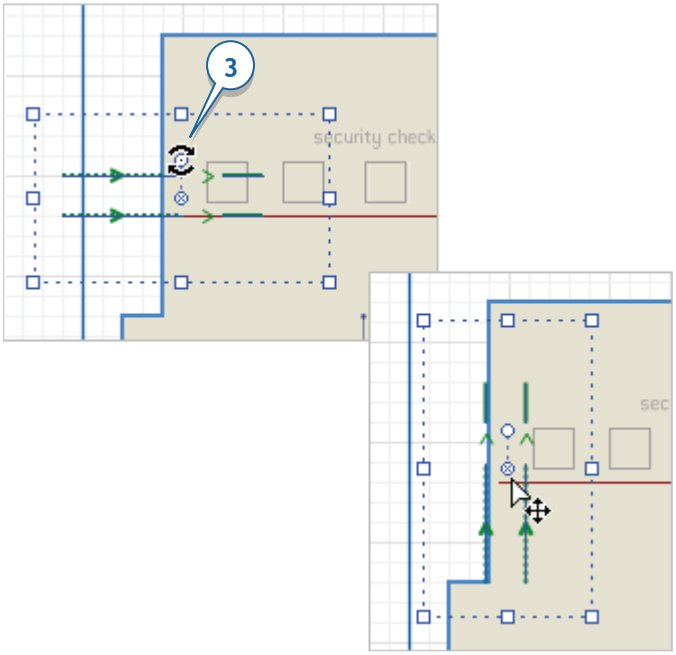
Линейные и точечные сервисы в пешеходных моделях

Существует два типа сервисов в пешеходных моделях: *линейные* и *точечные*.

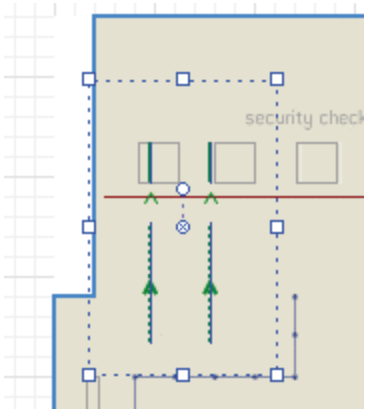
- *Линейный* сервис задается линией. Пешеход начинает процедуру обслуживания в начальной точке линии и затем продвигается к ее конечной точке, откуда он может покинуть сервис. Примеры линейного сервиса: турникет, рамка металлодетектора.
- *Точечные* сервисы задаются точкой. Во время обслуживания пешеход будет находиться у заданной точки сервиса. Примеры точечного сервиса: билетная касса, банкомат.

Мы используем линейные сервисы, потому что хотим, чтобы пассажир следовал вдоль линии сервиса и таким образом проходил через рамку металлодетектора, как это и происходит при предполетном досмотре в аэропорту. Нам будет нужно развернуть и переместить линии сервисов так, чтобы они пересекали предполагаемые места расположения рамок металлодетекторов на плане аэропорта.

3. Поверните сервисы с помощью круглого маркера, расположенного чуть выше центра фигуры.



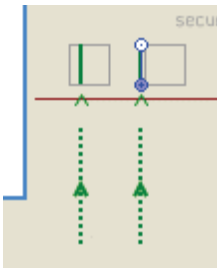
4. Переместите фигуру так, чтобы линия первого сервиса пересекала квадрат, обозначающий рамку металлодетектора.



Как перемещать элементы, игнорируя сетку

Если вам нужно переместить элемент на графической диаграмме без его привязки к сетке, то вы можете нажать клавишу Alt на клавиатуре и переместить объект, не отпуская эту клавишу. Либо же отключите привязку к сетке с помощью кнопки панели управления **Включить/Отключить сетку**.

5. Выделите вторую линию сервиса.



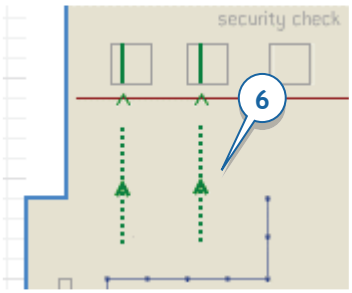
Сложные фигуры разметки пространства

Некоторые фигуры разметки пространства состоят из нескольких отдельных фигур. Так, например, фигура **Сервис с очередями** состоит из фигур **Сервис** и **Очередь**, а фигура **Сервис с областью** состоит из фигур **Сервис** и **Прямоугольная область**.

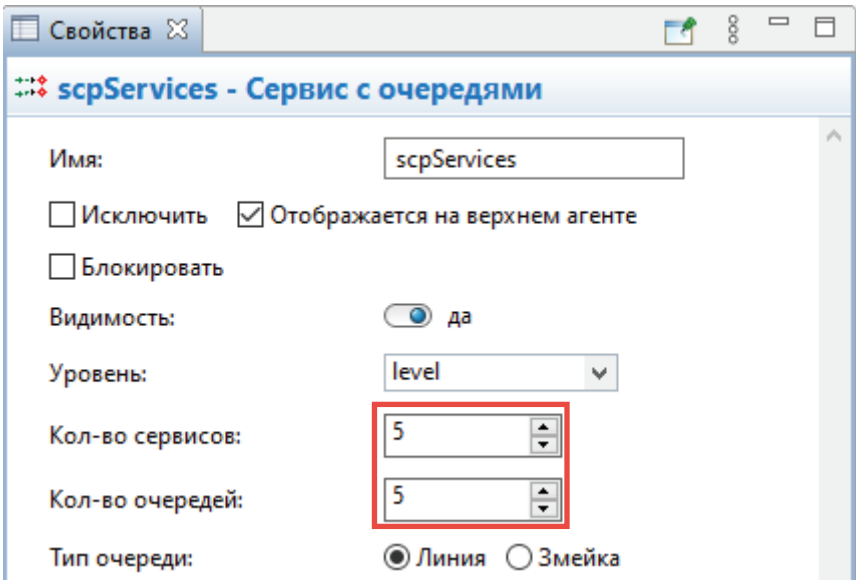
Обратите внимание на правила выделения фигур, входящих в состав сложных фигур разметки пространства:

- Первый щелчок мыши по фигуре выделит саму сложную фигуру разметки пространства (**Сервис с очередями**).
- После того, как вы выделите сложную фигуру разметки пространства, вы можете выделить любую простую фигуру, входящую в ее состав (в данном случае - **Сервис** или **Очередь**), щелкнув по ней мышью.

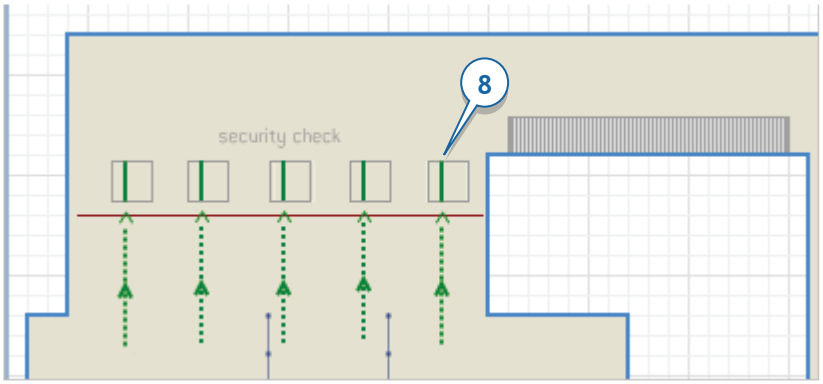
6. Переместите линию второго сервиса в то место плана, где находится второй пункт досмотра. Не забудьте соответственно переместить и линии очереди, ведущие к этим сервисам.



7. Перейдите в свойства фигуры **Сервис с очередями** и увеличьте **Количество сервисов** и **Количество очередей** до *5*.



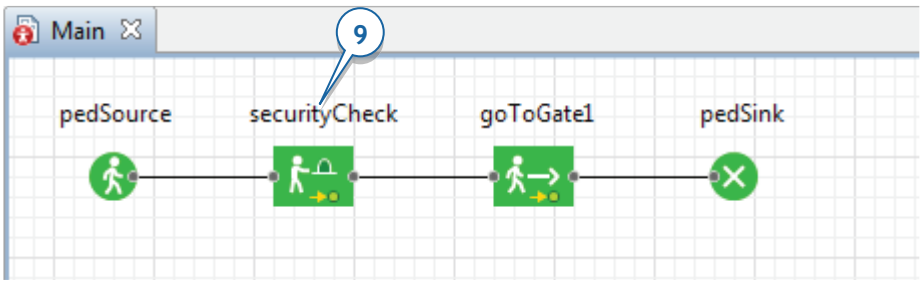
8. Если нужно, поправьте расположение новых линий сервисов и линий очередей. Сервисы и линии должны в итоге быть расположены так, как показано на рисунке ниже.



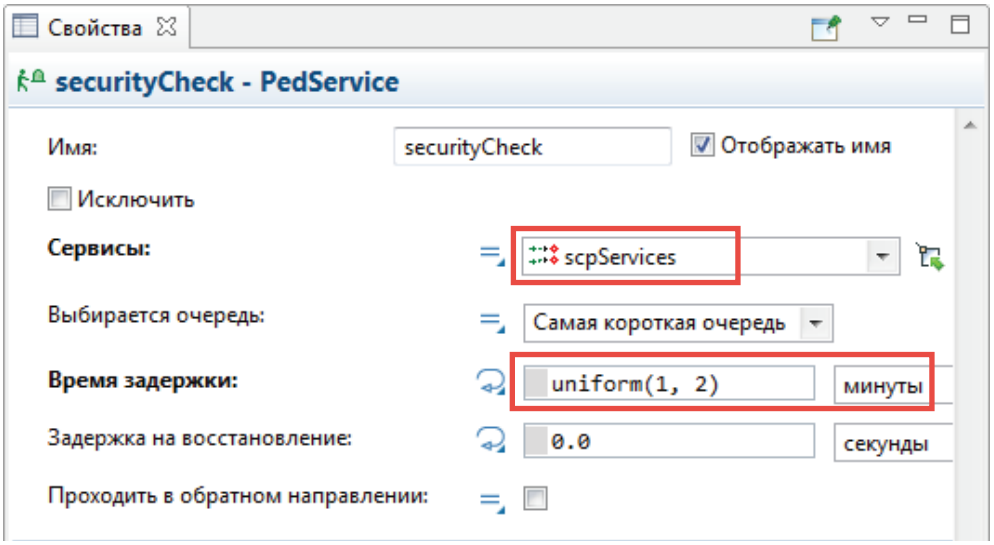
Теперь, когда мы закончили рисование пунктов досмотра с помощью фигур разметки пространства, мы можем добавить и сам процесс прохождения предполетного досмотра в текущую логику модели. Для этого мы воспользуемся специальным блоком **Пешеходной библиотеки PedService** , в свойствах которого мы укажем наш элемент разметки пространства **Сервис с очередями** *scpServices*.

Давайте добавим еще один блок диаграммы процесса, который будет моделировать предполетный досмотр пассажиров.

9. Добавьте блок **PedService** . Поместите его в диаграмму процесса между блоками **PedSource** и **PedGoTo**. Теперь пешеходы будут проходить через заданные пункты сервиса (в нашем случае это пункты досмотра). Назовите этот блок *securityCheck*.

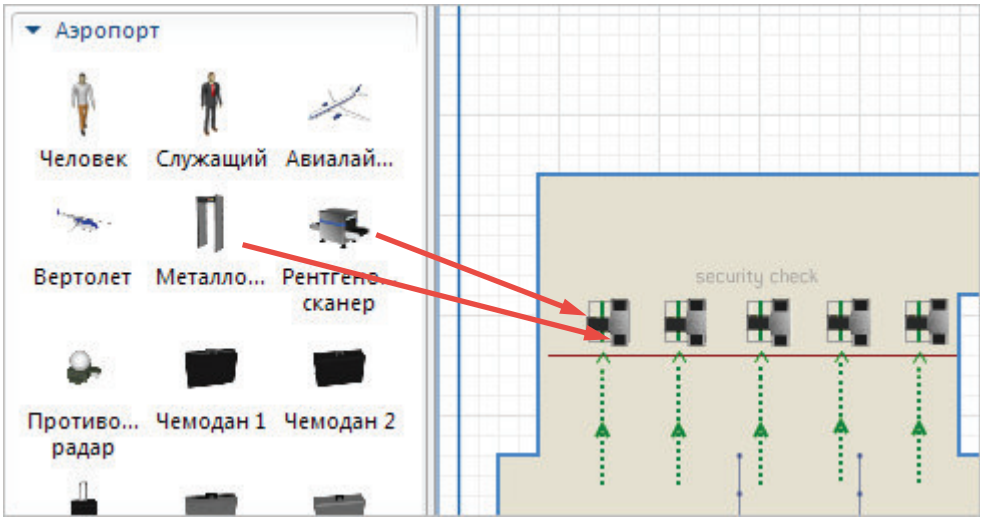


10. Откройте свойства блока *securityCheck*. В поле **Сервисы** выберите *scpServices* (имя добавленной ранее фигуры разметки пространства).



11. Мы считаем, что процедура предполетного досмотра в среднем занимает от одной до двух минут, поэтому введите *uniform(1, 2)* в поле **Время задержки** и выберите **минуты** в поле справа.

12. Теперь давайте нарисуем пять пунктов досмотра с помощью объектов **Металлодетектор** и **Рентгеновский сканер** из раздела **Аэропорт** палитры **3D Объекты**. После того, как вы добавите объекты **Рентгеновский сканер**, измените их **Масштаб** на *75%*.

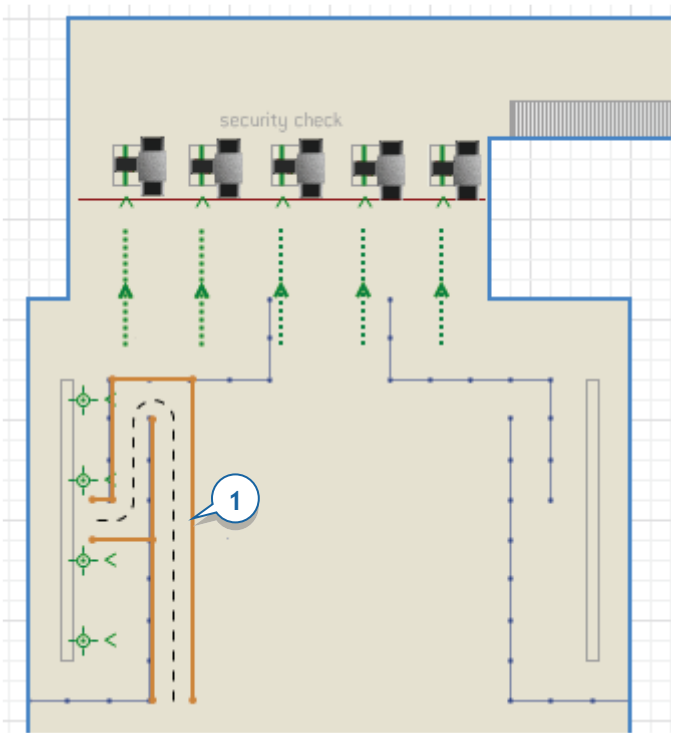


13. Запустите модель. Вы увидите, что теперь пассажиры проходят процедуру предполетного досмотра.

Фаза 4. Добавление стоек регистрации

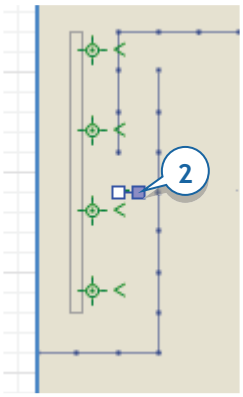
Пассажиры могут регистрироваться на рейсы различными способами – заранее (онлайн-регистрация) или привычным образом - в аэропорту у стоек регистрации. Давайте добавим в нашу модель стойки регистрации на рейсы, и направим туда тех пассажиров, кто не зарегистрировался на свой рейс заблаговременно.

1. Добавьте еще один объект **Сервис с очередями** на план аэропорта, чтобы промоделировать стойки регистрации. В этот раз нам необходимо задать точечный тип сервиса, состоящий из четырех пунктов сервиса и одной общей очереди. Назовите этот элемент *checkInServices*.

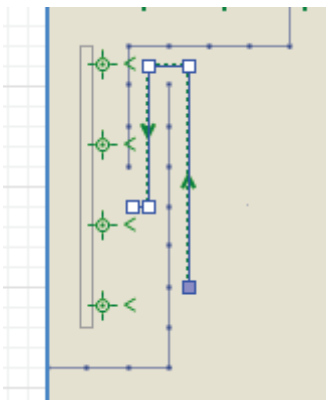


Переместите диаграмму процесса, собранную из блоков Пешеходной библиотеки, выше оси X, вынеся ее тем самым за пределы области, видимой во время работы модели. Если во время работы модели вам понадобится изучить ход моделируемых процессов с помощью этой диаграммы, вы всегда можете посмотреть на нее, просто переместив полотно окна модели с помощью мыши.

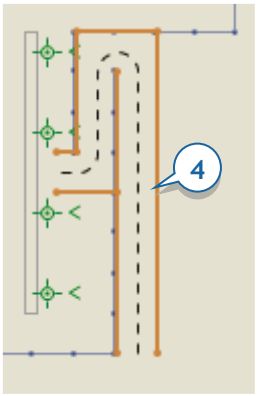
2. Разместите фигуру сервисов так, как показано на рисунке ниже. Обратите внимание, что мы начинаем рисование очереди, ограниченной ленточными барьерами с того, что помещаем начальную точки линии очереди у подхода к стойке регистрации. Вторую точку линии нужно поместить точно посередине ограниченного барьерами прохода, в месте первого изгиба очереди.



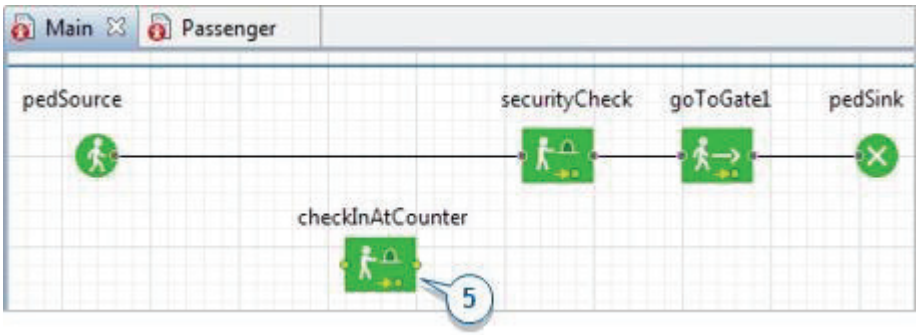
3. Теперь давайте продолжим рисование линии очереди, чтобы она следовала обозначенным на плане терминала ленточным барьерам. Сделайте щелчок правой кнопкой мыши по очереди и выберите пункт контекстного меню **Добавить точки**. Добавьте новые точки, щелкая мышью в точках, выделенных на рисунке ниже. Завершите рисование, сделав двойной щелчок мышью в точке окончания очереди- «змейки». В итоге должна будет получиться «змейка» следующей формы:



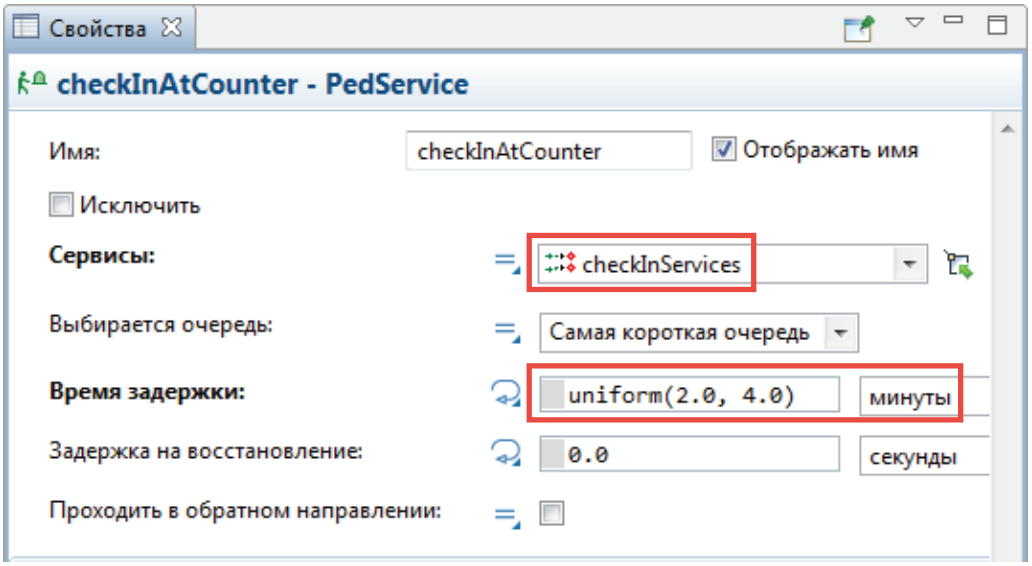
4. В свойствах этой фигуры **Сервис с очередями**, измените **Тип очереди** на **Змейка**. Вы увидите, что теперь у очереди есть границы. На сцене 3D анимации они будут отображаться в виде ленточных барьеров.



5. Добавьте еще один объект **PedService** и назовите его *checkInAtCounter*.



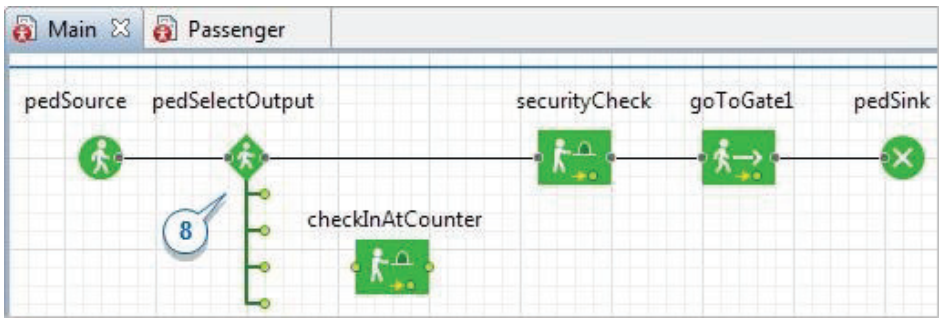
6. В свойствах объекта выберите сервис *checkInServices* в поле **Сервисы**.



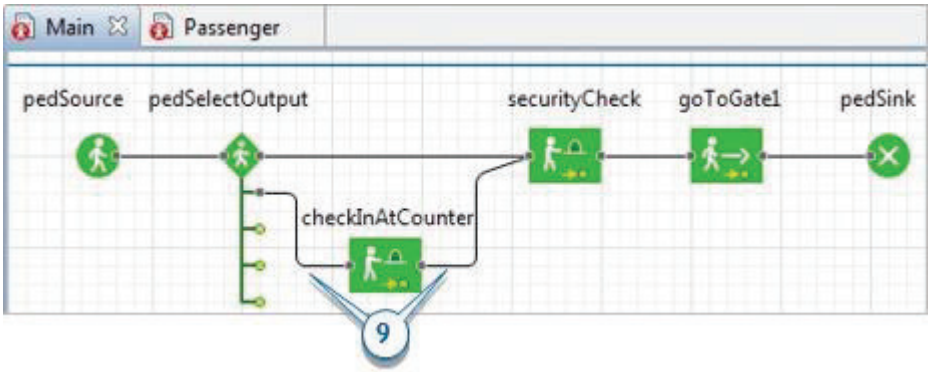
7. Поскольку мы полагаем, что в среднем процесс регистрации на рейс занимает от двух до четырех минут, то давайте введем в поле **Время задержки** вызов функции равномерного распределения *uniform(2, 4)* и выберем **минуты** из расположенного справа списка.

Поскольку пассажиры могут регистрироваться на рейсы различными способами, нам нужно направить разные группы пассажиров в разные подпроцессы.

8. Чтобы направлять пешеходов в различные ветви диаграммы процесса, мы используем объект **PedSelectOutput**.



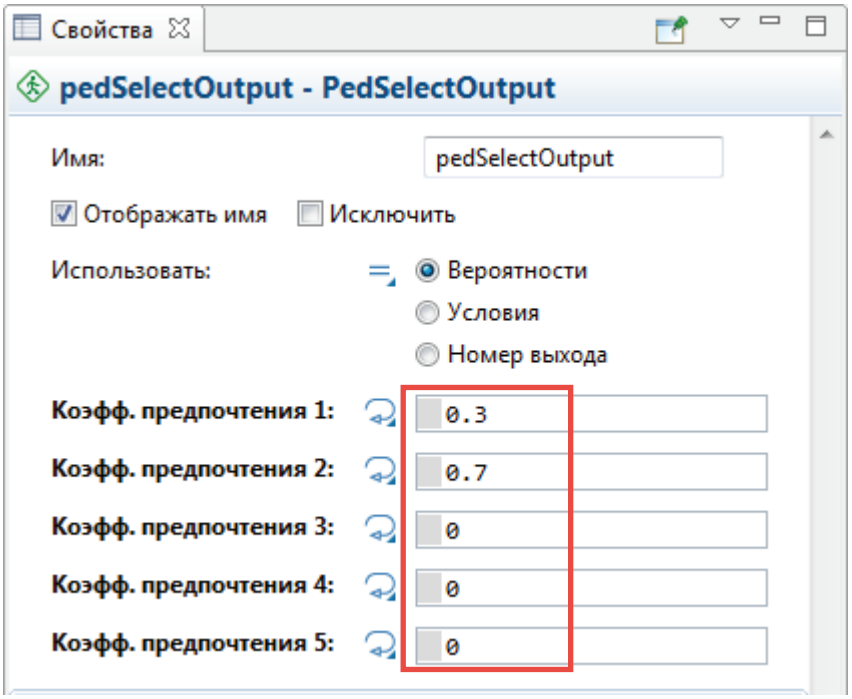
9. Соедините блок *checkInAtCounter* с текущей диаграммой процесса так, как показано на рисунке.



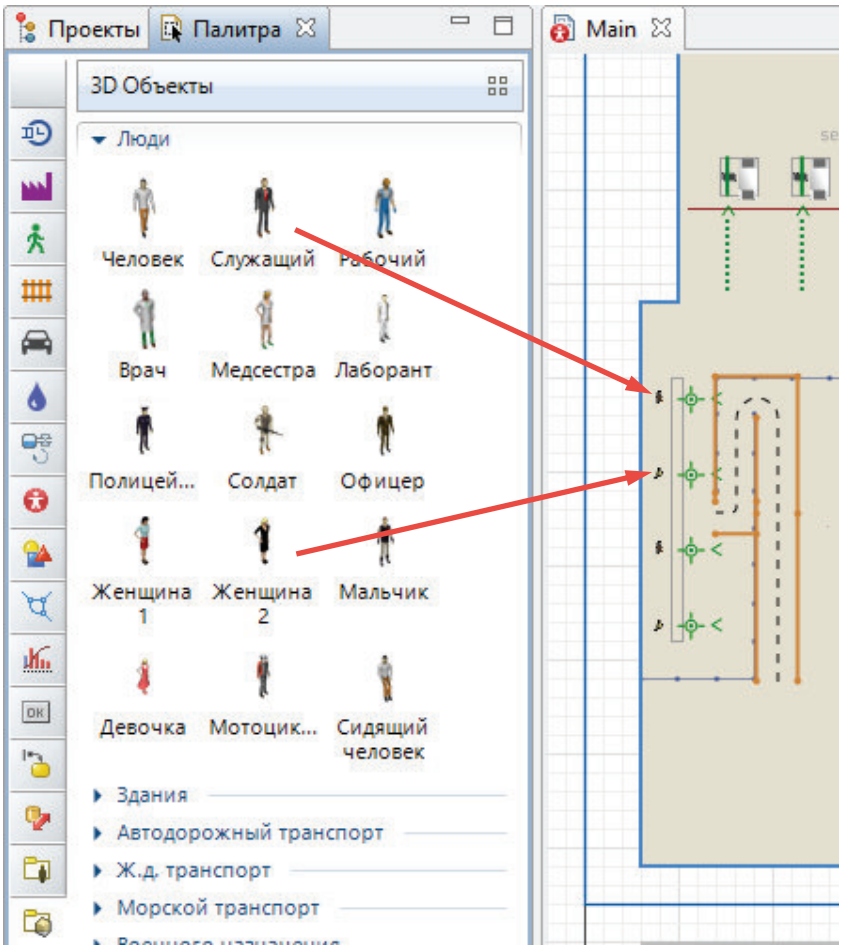
Как нарисовать соединители сложной формы

Когда вы добавляете блоки из палитры на графическую диаграмму, ближайшие друг к другу порты расположенных поблизости блоков будут соединяться автоматически. Если же вам понадобится нарисовать соединитель более сложной формы (например, как на рисунке выше), то это можно сделать, нарисовав такой соединитель вручную. Для этого нужно сделать двойной щелчок по первому соединяемому поту, затем щелкнуть по диаграмме в точках изгиба соединителя, и закончить рисование, сделав щелчок по второму соединяемому порту. Чтобы удалить точку изгиба соединителя, сделайте двойной щелчок по этой точке.

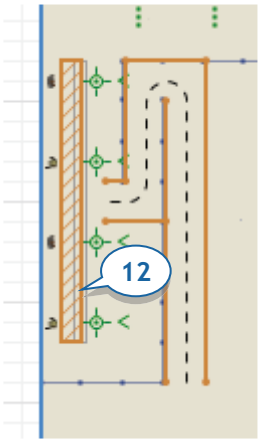
10. Мы полагаем, что 30 процентов пассажиров регистрируются онлайн, а 70 процентов – у стоек регистрации. Чтобы задать такое деление потока пассажиров, задайте в свойствах блока *pedSelectOutput* **Коэфф. предпочтения 1** равным *0.3*, а **Коэфф. предпочтения 2** равным *0.7*. При этих значениях блок будет перенаправлять 30 процентов пешеходов в верхнюю ветвь диаграммы процесса, а 70 - в нижнюю. Нужно будет также задать значение 0 в полях **Коэфф. предпочтения 3**, **Коэфф. предпочтения 4** и **Коэфф. предпочтения 5**, чтобы предотвратить перенаправление пешеходов в три нижних порта блока, которые не соединены ни с какими другими блоками.



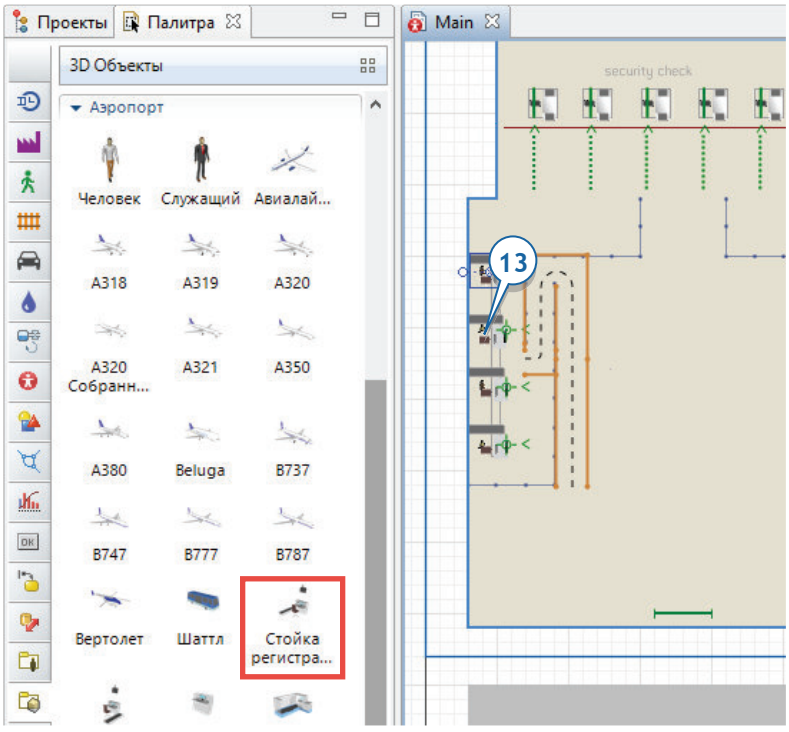
11. Теперь давайте добавим 3D модели объектов служащих авиакомпании у стоек регистрации. Добавьте на диаграмму четыре фигуры, используя объекты из раздела **Люди** палитры **3D Объекты**: *Служащий* и *Женщина 2*.



12. Сделайте область стоек регистрации недоступной для прохождения пешеходов. Для этого добавьте элемент **Прямоугольная стена** из раздела **Разметка пространства** палитры **Пешеходная библиотека** и поместите ее так, как показано на рисунке ниже. В свойствах стены, смените значение свойства **Видимость** на **нет**, чтобы сделать эту фигуру невидимой во время работы модели (иначе эта стена будет отображаться на трехмерной анимации в виде параллелепипеда).



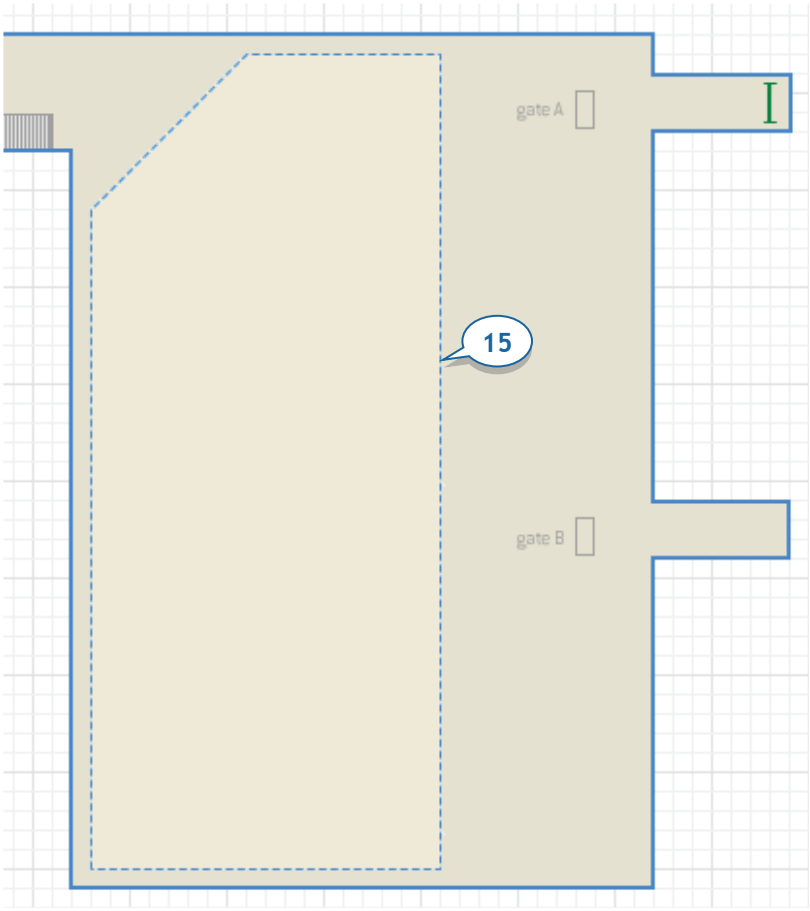
13. Добавьте в область стоек регистрации четыре 3D объекта **Стойка регистрации** из секции **Аэропорт** палитры **3D Объекты**. Поверните эти фигуры, выбрав в поле **Поворот, Z**: *-90 градусов* (в секции свойств **Расположение**).



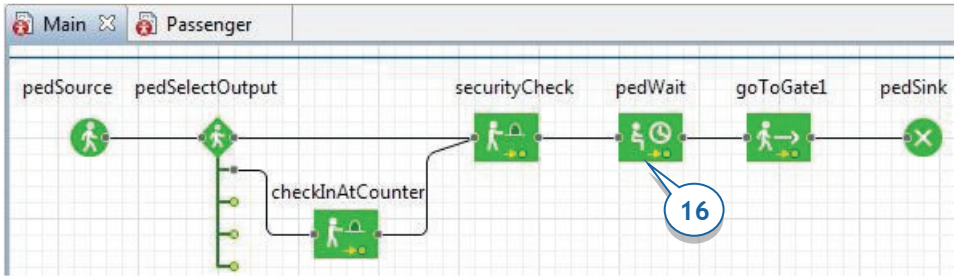
14. Запустите модель. Вы увидите, что теперь многие пассажиры вначале регистрируются на рейс у стойки регистрации, и только затем проходят к выходу на посадку.

Мы хотим, чтобы прежде чем отправиться на посадку, пассажиры проводили определенное время в зоне перед гейтами. Для этого нам нужно будет нарисовать область ожидания, в которой пассажиры будут ожидать начала посадки на рейс, а затем добавить в диаграмму процесса блок (**PedWait**), который будет моделировать само ожидание.

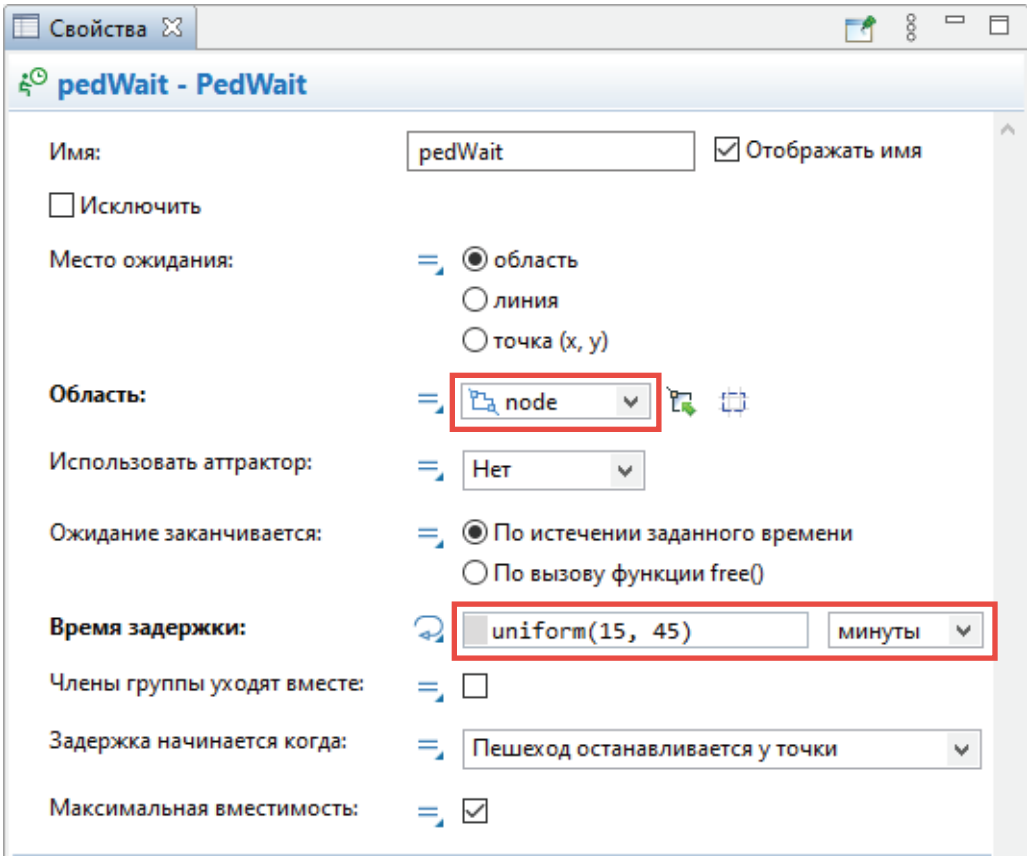
15. Нарисуйте область ожидания перед выходами на посадку с помощью элемента разметки пространства **Многоугольный узел**  (совет - используйте режим рисования). Последовательно щелкайте мышью в углах области на плане терминала аэропорта, последовательно добавляя их по направлению часовой стрелки. Если потребуется, вы можете отредактировать получившуюся фигуру уже после того, как вы закончите ее рисование.



16. Добавьте блок **PedWait** в диаграмму процесса и поместите его между блоками **PedService** и **PedGoTo**.



17. Измените свойства блока **pedWait**. Выберите в поле **Область** нарисованный нами ранее многоугольный узел *node*, а затем укажите время ожидания в свойстве **Время задержки**: `uniform(15, 45)` **минут**.



18. Снова запустите модель. Вы сможете наблюдать, как пассажиры на какое-то время задерживаются в области ожидания и только затем отправляются на посадку.

Вы можете добавить дополнительные стойки в заготовленной для этого части на плане терминала (это потребует также соответствующего изменения свойств блока **PedSelectOutput**, который будет разделять поток пешеходов на большее количество подпотоков).

доп.задание: промоделируйте терминалы для самостоятельной регистрации пассажиров.

Фаза 5. Моделирование посадки на самолет

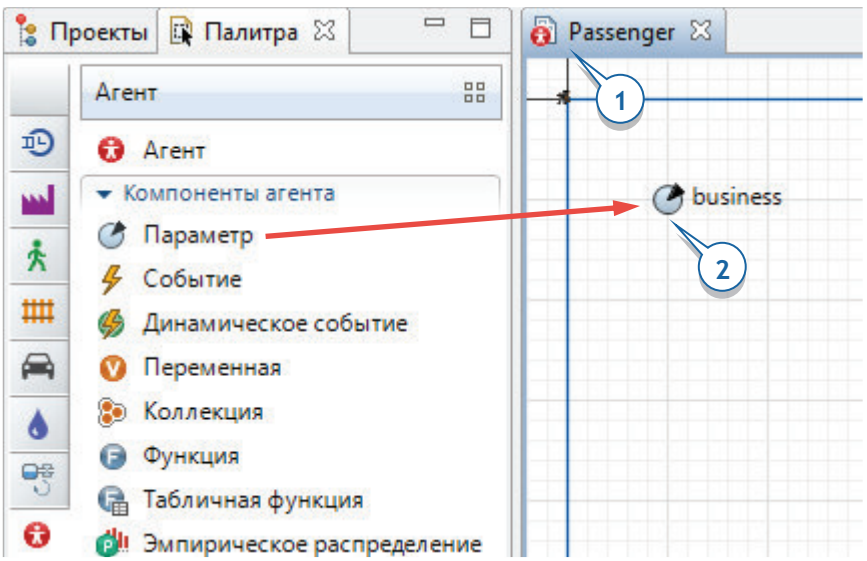
Пассажиры будут ожидать начала посадки на рейс в области ожидания (общей для обоих выходов на посадку). Когда начнется посадка, они должны будут пройти процедуру проверки посадочных талонов, после чего они смогут пройти на борт самолета.

К стойке проверки посадочных талонов и документов ведут две очереди – одна для пассажиров бизнес-класса, другая – для пассажиров эконом-класса. Проверка документов каждого пассажира занимает в среднем от 1 до 3 секунд.

1. Откройте диаграмму типа агента *Passenger*, сделав двойной щелчок по элементу *Passenger* в панели **Проекты**.

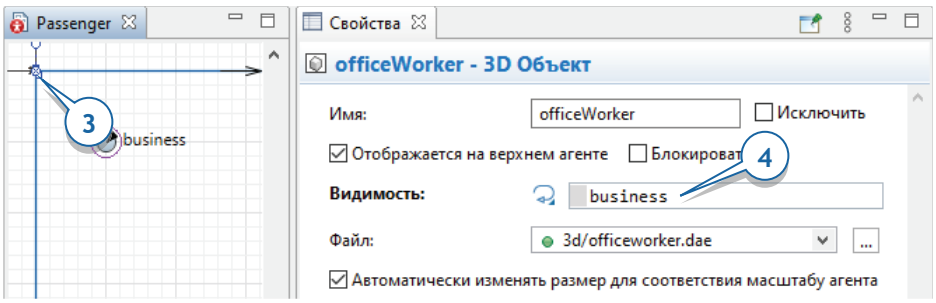
Нам нужно как-то отличать в нашей модели пассажиров бизнес-класса от пассажиров эконом-класса. Для этого потребуется хранить дополнительную информацию о пассажире в заданном нами ранее типе пешехода.

2. Добавьте **Параметр** из палитры **Агент**. Назовите этот параметр *business* и выберите **Тип**: *boolean*. Этот параметр будет определять, летит ли данный пассажир бизнес-классом. Если значение параметра равно *true*, то это пассажир бизнес-класса, в противном случае (если значение параметра равно *false*), это пассажир экономического класса.



Мы хотим, чтобы пассажиров, летящих разными классами, можно было легко различить визуально на анимации модели. Для этого мы будем использовать две разные 3D модели людей для отображения пассажиров бизнес-класса и эконом-класса.

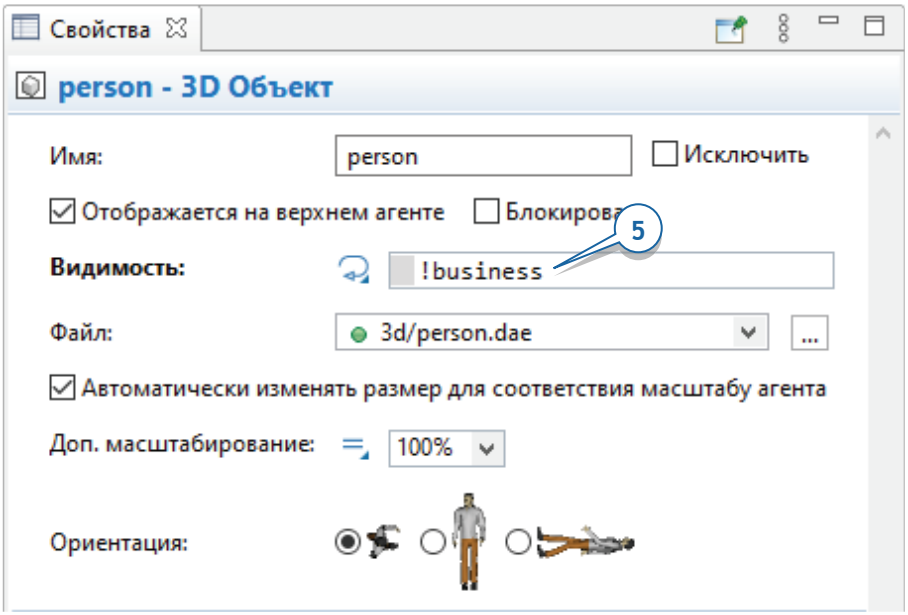
3. Добавьте 3D объект **Служащий**, чтобы задать фигуру анимации пассажира бизнес-класса. Поместите эту фигуру в точку начала координат (0,0) диаграммы типа пешехода *Passenger.*



4. Измените свойство **Видимость** для обоих 3D объектов на диаграмме типа пешехода *Passenger*. Вначале щелкните по фигуре анимации *Служащий*. Мы хотим, чтобы эта фигура была видна только тогда, когда это пассажир бизнес-класса, то есть, значение его параметра *business* равно *true*. Для этого переключите поле задания значения свойства **Видимость** в режим задания динамического выражения, щелкнув по расположенному слева от поля значку , а затем введите *business* в этом поле.

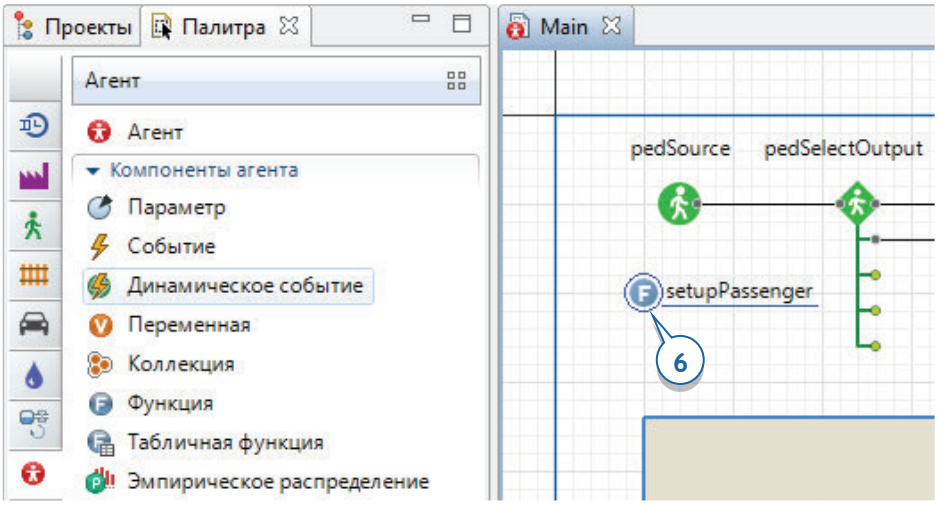
5. Теперь выберите фигуру *person* (это можно сделать из дерева в панели **Проекты**, фигура находится в ветке **Презентация** типа агента *Passenger*), и аналогично введите следующее динамическое выражение в поле **Видимость**: *! business* . Теперь эта фигура будет отображаться только в том случае, если данный пассажир летит эконом-классом.

   Символ ! является булевским оператором НЕ в языке Java. Выражение *! business* возвращает *true*, когда значение параметра *business* НЕ РАВНО *true* (не истинно), то есть это пассажир не бизнес-класса, а эконом-класса.



Мы будем задавать класс пассажира в момент его создания блоком **PedSource**.

6. Перейдите обратно на диаграмму *Main* и добавьте **Функцию** из палитры **Агент**. Назовите эту функцию *setupPassenger*.

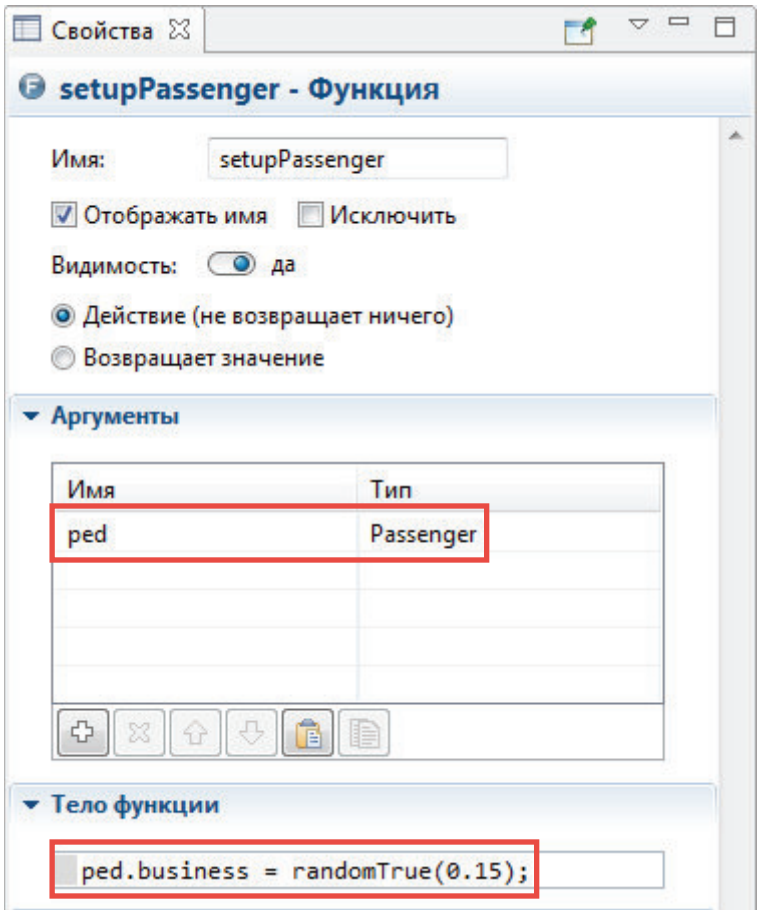


7. Задайте следующие параметры функции:

- Добавьте аргумент, чтобы с его помощью передать данной функции ссылку на только что созданного пешехода.
  Задайте для этого аргумента:
  Имя: *ped*
  Тип: *Passenger*

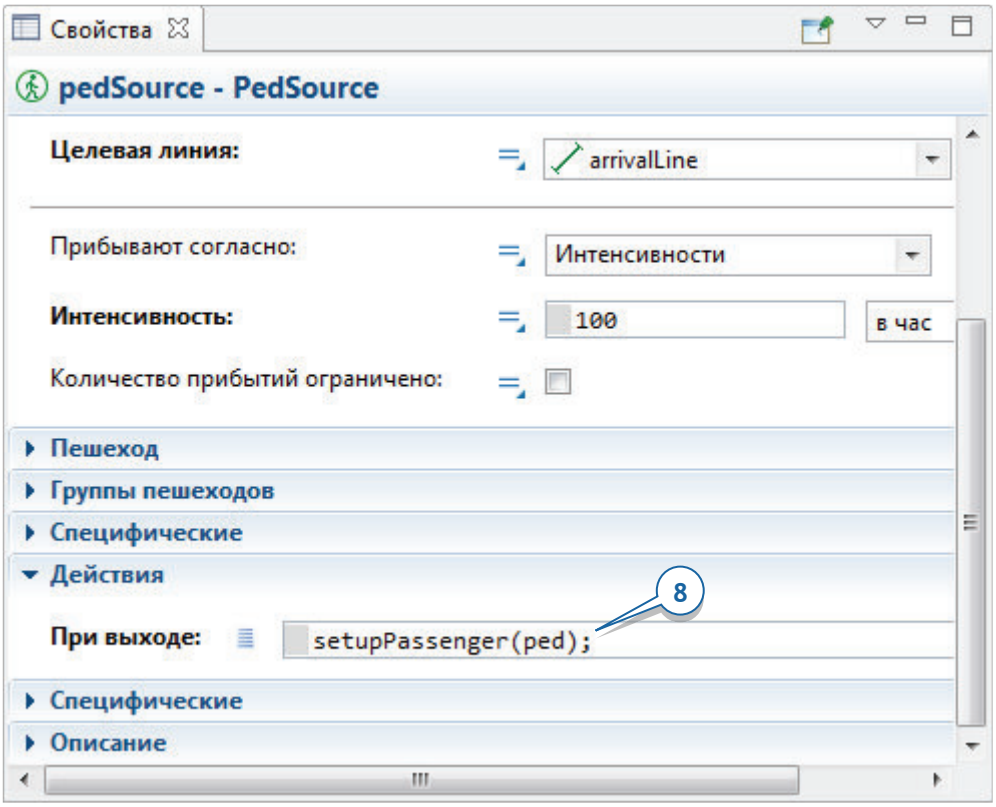
- Введите следующий код функции:
  ped.business = randomTrue(0.15);

  Эта строка будет присваивать полю пешехода *business* значение *true* с вероятностью 15 процентов:



Здесь *ped* – это заданный нами аргумент функции, пешеход типа *Passenger*. Выбрав *Passenger* в качестве типа аргумента, мы можем напрямую обращаться к параметру этого пешехода *business* просто как *ped.business*. Функция *randomTrue(0.15)* возвращает *true* в среднем в 15 процентах случаев, что означает, что в среднем бизнес-классом в нашей модели будет лететь 15 процентов пассажиров.

8. Будем вызывать эту функцию в объекте *pedSource* при создании нового пешехода. Разверните секцию свойств **Действия** и введите код в поле параметра **При выходе**:
   setupPassenger(ped);

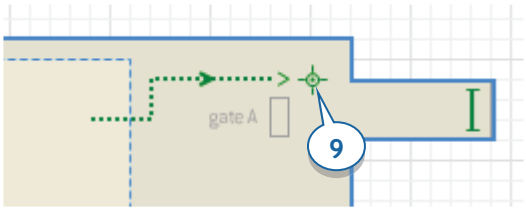


В данном случае мы вызываем нашу функцию *setupPassenger* для каждого созданного пешехода. Аргумент нужен нам для того, чтобы передать функции ссылку на текущего пешехода. Тогда мы сможем выполнить в коде функции необходимые действия с этим пешеходом (в нашем случае – изменить значение его параметра).

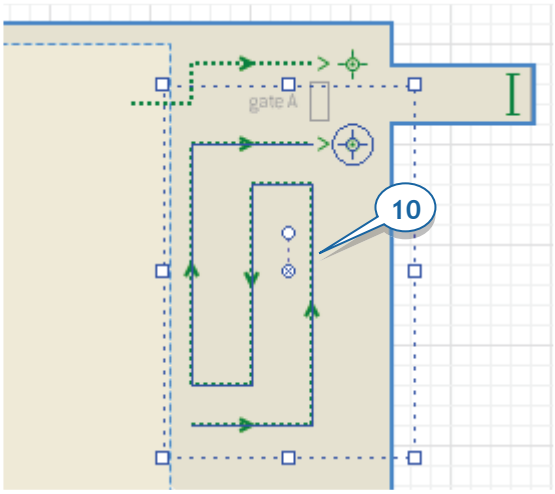
Нарисуйте два сервиса с очередями у верхнего выхода на посадку: один для пассажиров бизнес-класса, другой – для эконом-класса.

Обратите внимание, что это два разных сервиса, а не просто две очереди у одной фигуры.

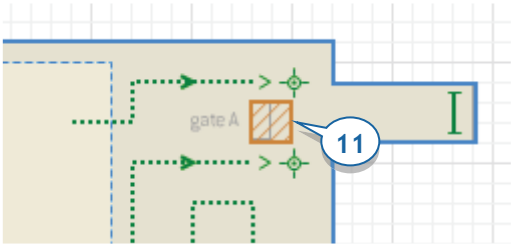
9. Добавьте на диаграмму **Сервис с очередями**, который будет отвечать за регистрацию пассажиров бизнес-класса (это должен быть точечный сервис с одной точкой сервиса и одной очередью). Назовите эту фигуру **Сервис с очередями** *business1*.



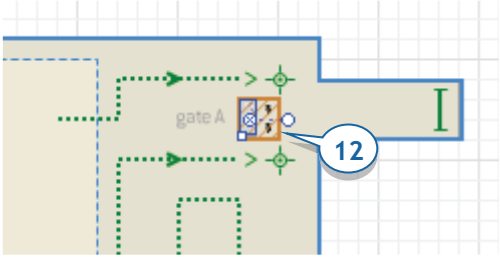
10. Добавьте еще один **Сервис с очередями** (как показано на рисунке ниже). Назовите этот сервис *economy1*.



11. С помощью элемента **Прямоугольная стена** нарисуйте область стойки проверки документов около выхода на посадку.

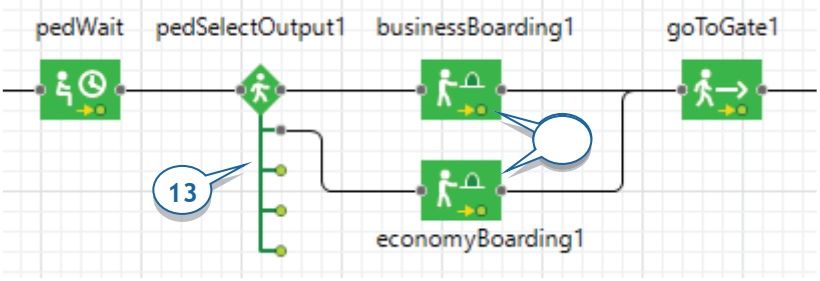


12. Добавьте 3D фигуру **Стойка у гейта** из секции **Аэропорт** палитры **3D Объекты**. Поверните ее (**Поворот Z** в секции свойств **Расположение**: *90* **градусов**) и добавьте две фигуры служащих за стойку. Чтобы стена не была видна на анимации во время работы модели, смените значение свойства **Видимость** этой стены на **нет**.



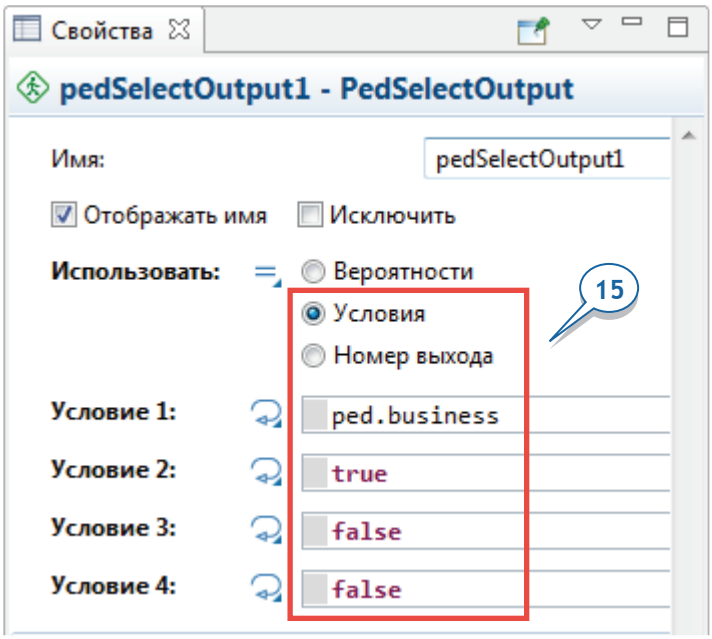
Добавьте новые объекты Пешеходной библиотеки между объектами *pedWait* и *goToGate1*.

13. Добавьте блок **PedSelectOutput** , который будет перенаправлять пассажиров бизнес-класса и пассажиров эконом-класса в разные очереди.



14. Добавьте два блока **PedService** : *businessBoarding1* и *economyBoarding1*. Они будут моделировать процесс проверки посадочных талонов у пассажиров бизнес-класса и эконом-класса соответственно.

15. Чтобы блок **PedSelectOutput** направлял пассажиров бизнес-класса и эконом-класса в разные очереди, выберите опцию **Использовать: Условия**, и введите *ped.business* в поле **Условие 1**. Это выражение вернет значение *true* («истина») для всех пассажиров бизнес-класса, и вследствие этого эти пешеходы проследуют по той части диаграммы процесса, которая соединена с верхним выходным портом блока, и пройдут на проверку посадочных талонов, встав в очередь приоритетного обслуживания. После того, как вы зададите условия для оставшихся выходных портов блока (*true*, *false*, *false*), все оставшиеся пассажиры (летящие эконом-классом) проследуют во второй выходной порт.



16. В свойствах блока **PedService** *businessBoarding1*, выберите *business1* в поле **Сервисы**. Поскольку время проверки документов в среднем занимает от двух до пяти секунд, задайте соответствующее значение в поле **Время задержки**.

17. В свойствах блока *economyBoarding1*, выберите *economy1* в поле **Сервисы** и аналогично измените **Время задержки**.

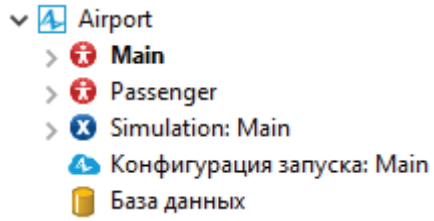
18. Запустите модель. Вы увидите, как служащие аэропорта проверяют посадочные талоны пассажиров. Пассажиры встают в одну из двух очередей в зависимости от того, имеют ли они право на приоритетное обслуживание или нет

Фаза 6. Считывание данных о рейсах из файла MS Excel

Теперь мы добавим в нашу модель рейсы, считав расписание вылетов из таблицы Excel.

База данных AnyLogic

В каждой модели AnyLogic есть своя встроенная база данных, в которой могут храниться значения входных параметров модели, а также результаты работы модели.



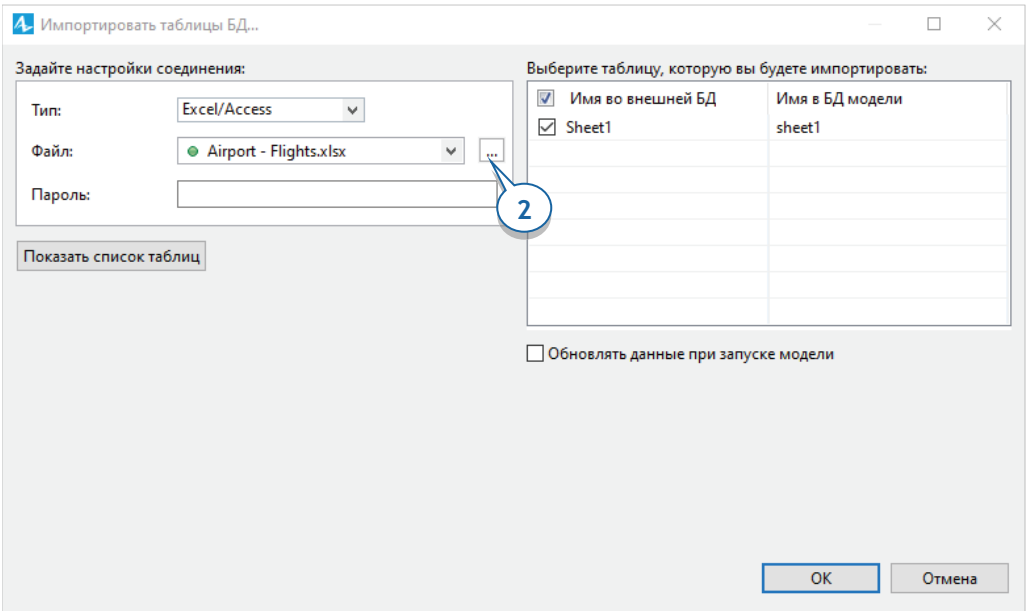
База данных AnyLogic позволяет:

- Конфигурировать модель согласно заданным в БД параметрам.
- Параметризовать популяции агентов.
- Задавать частоту прибытия агентов-заявок в процессных моделях.
- Импортировать данные из других БД или таблиц Excel и хранить их в доступной форме.
- Записывать логи (журналы выполнения) модели, в которые добавляется информация обо всех произошедших событиях в диаграммах процессов, диаграммах состояний, статистика по взаимодействию и перемещению агентов, и т.д.
- Сохранять и экспортировать статистику, хранящуюся в наборах данных.
- Экспортировать данные в электронные таблицы MS Excel.
- Создавать резервные копии БД.

Мы покажем, как импортировать данные из внешней базы данных во встроенную БД модели.

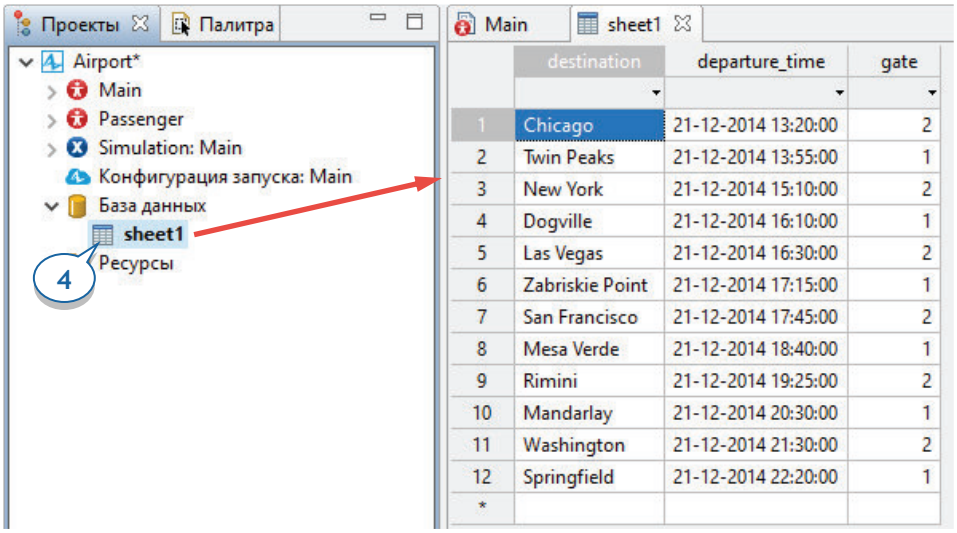
1. В дереве **Проекты**, щелкните правой кнопкой мыши по элементу **База данных** и выберите **Импортировать таблицы БД...** из контекстного меню.

2. Вы увидите диалоговое окно **Импортировать таблицы БД**. Выберите файл базы данных, хранящей необходимые для нашего проекта данные. Щелкните по кнопке ... и выберите файл *Flights.xlsx* (его можно найти в папке *каталог установки AnyLogic/resources/AnyLogic in 3 days/Airport*.



3. В расположенный в правой части окна список **Выберите таблицу, которую вы будете импортировать** будет добавлена таблица *Sheet1*. Щелкните по кнопке **OK**, чтобы импортировать данные из этой таблицы БД в вашу модель.

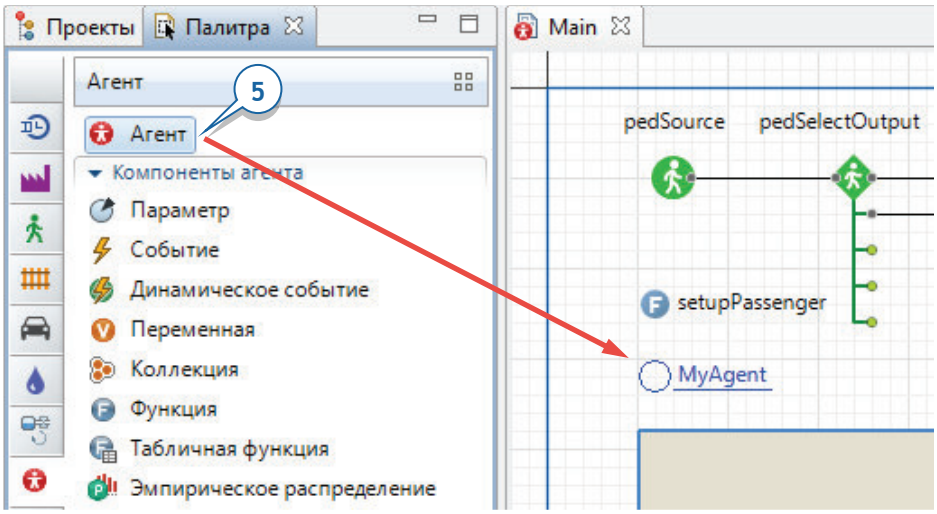
4. Вы увидите, что в ветке **База данных** в панели **Проекты** появится элемент *sheet1*. Сделайте двойной щелчок по этому элементу. При этом в центре рабочего пространства AnyLogic откроется редактор таблицы. Здесь вы сможете увидеть данные, предварительно считанные из таблицы Excel в эту таблицу встроенной базы данных AnyLogic. Таблица содержит три столбца, хранящих следующую информацию: пункт назначения рейса, время отправления и номер выхода на посадку (гейта).



	destination	departure_time	gate
1	Chicago	21-12-2014 13:20:00	2
2	Twin Peaks	21-12-2014 13:55:00	1
3	New York	21-12-2014 15:10:00	2
4	Dogville	21-12-2014 16:10:00	1
5	Las Vegas	21-12-2014 16:30:00	2
6	Zabriskie Point	21-12-2014 17:15:00	1
7	San Francisco	21-12-2014 17:45:00	2
8	Mesa Verde	21-12-2014 18:40:00	1
9	Rimini	21-12-2014 19:25:00	2
10	Mandarlay	21-12-2014 20:30:00	1
11	Washington	21-12-2014 21:30:00	2
12	Springfield	21-12-2014 22:20:00	1
*			

Теперь мы создадим новый тип агента: *Flight*.

5. Добавьте пустую популяцию агентов типа *Flight*, перетащив элемент **Агент** из палитры **Агент** на диаграмму *Main*.



6. Вы увидите окно Мастера создания агентов. Выполните следующие шаги:

a. На первой странице Мастера, выберите пункт **Популяция агентов**.

b. Выберите пункт **Я хочу создать новый тип агента**. Щелкните по кнопке **Далее**.

c. Задайте **Имя типа агента**: *Flight*. Имя популяции автоматически изменится на *flights*. Выберите опцию **Использовать таблицу базы данных** и щелкните по кнопке **Далее**.

d. На следующей странице Мастера, оставьте заданные по умолчанию настройки (мы будем считывать данные из таблицы *sheet1* базы данных нашей модели). Щелкните по кнопке **Далее**.

e. На следующей странице Мастера вам будет предложено создать параметры *destination*, *departureTime* и *gate* для каждого агента типа *Flight*. Это как раз то, что нам нужно. Щелкните по кнопке **Далее**.

f. Поскольку мы не планируем отображать рейсы на анимации, на странице выбора анимации агента выберите опцию **Нет**.

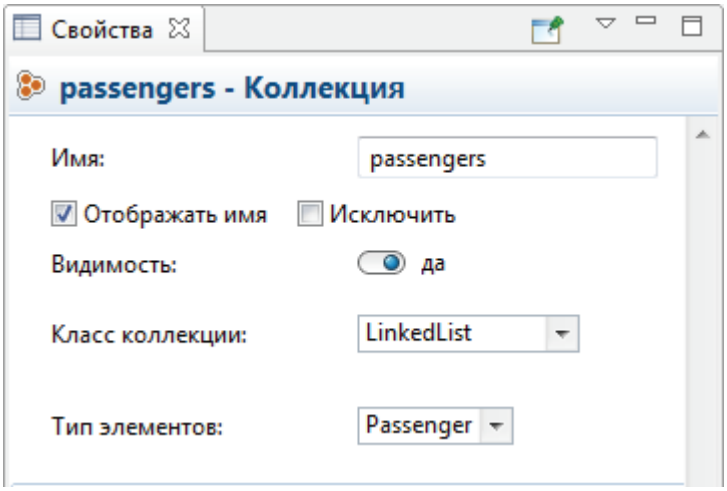
g. Щелкните по кнопке **Готово**.

7. В панели **Проекты**, сделайте двойной щелчок мышью по элементу *Flight*. На открывшейся диаграмме типа агента *Flight* вы должны будете увидеть три параметра:

- *destination*. **Тип**: *String*.
- *departureTime*. **Тип**: *Date*.
- *gate*. **Тип**: *int*.

Эти параметры будут использоваться для хранения времени вылета рейса, пункта назначения, а также номера выхода на посадку, используемого этим рейсом.

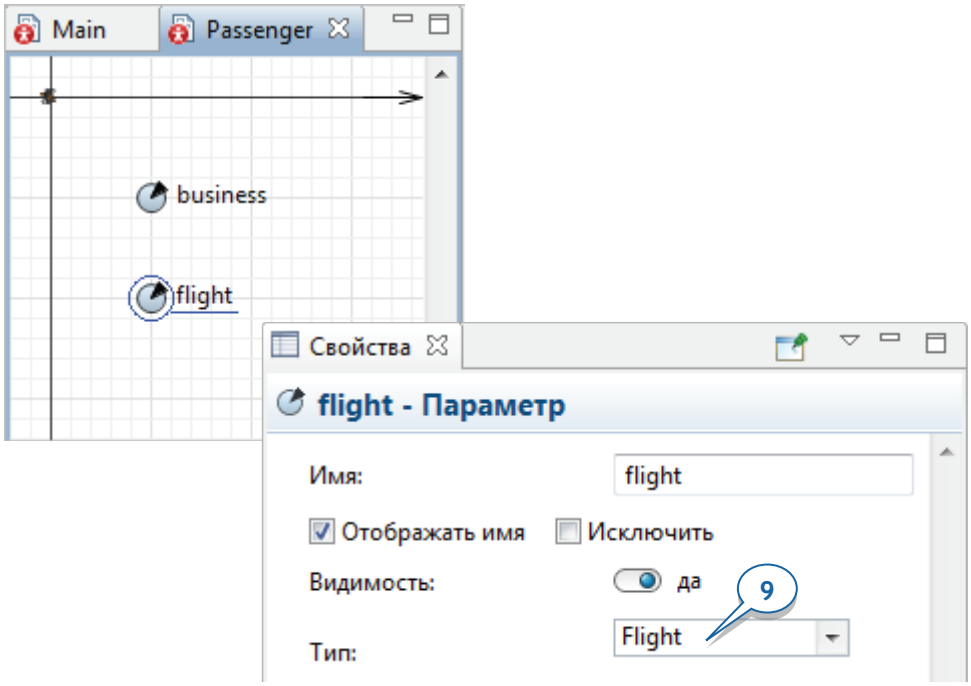
8. Добавьте на диаграмму *Flight* элемент **Коллекция**  из палитры **Агент**. Назовите эту коллекцию *passengers* и смените **Класс коллекции** на *LinkedList*, а **Тип элементов** на *Passenger*. В этой коллекции будет храниться список пассажиров, приобретших билеты на рейс.



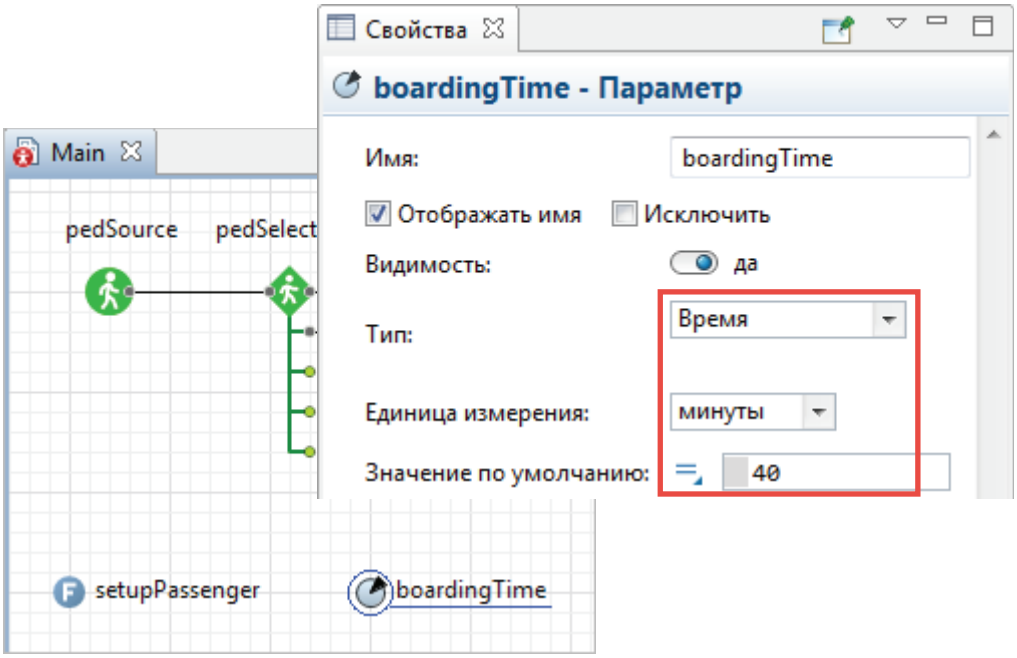
Коллекции

Коллекция представляет собой структуру данных, объединяющую вместе несколько однотипных элементов, и предоставляющую удобные механизмы для доступа к ним, управления этими элементами и сбору агрегированной статистики по элементам. Обычно элементы коллекции образуют логическую группу – это может быть список контактов человека, список невыполненных заказов и т.д.

9. Теперь, когда мы создали тип агента *Flight*, мы добавим параметр *flight* на диаграмму пешехода *Passenger* и сменим **Тип** параметра на *Flight*. Этот параметр будет хранить информацию о рейсе пассажира (с технической точки зрения, это ссылка на объект *Flight*, задающий этот рейс).

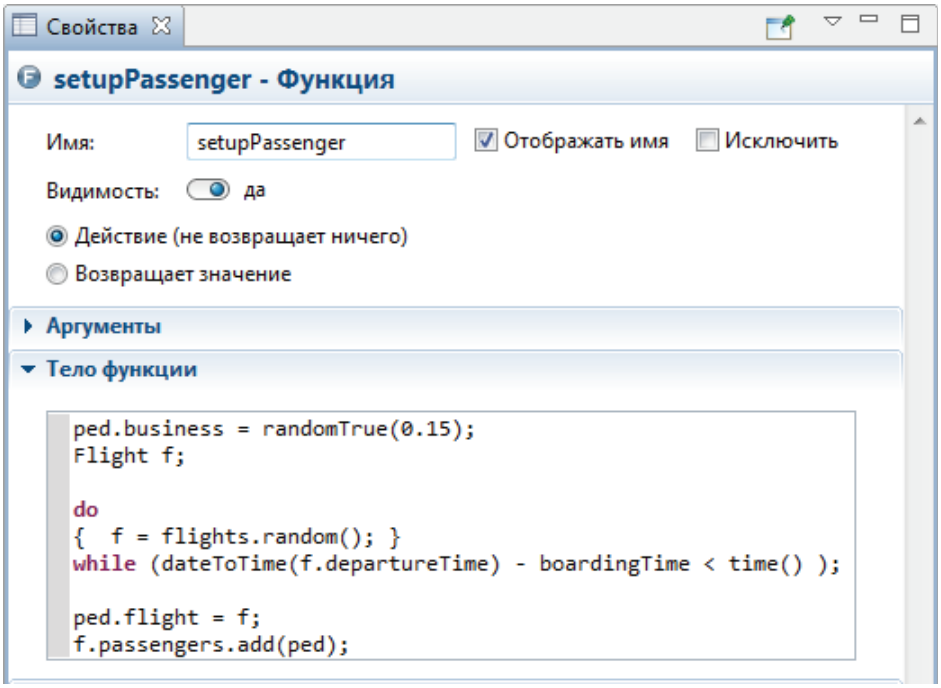


10. Вернитесь на диаграмму *Main* и добавьте параметр, который будет задавать длительность времени посадки в данном аэропорту. Назовите новый параметр *boardingTime*, выберите для него **Тип**: **Время**, **Единица измерения**: **минуты** и **Значение по умолчанию:** *40*.



11. Выберите ранее созданную функцию *setupPassenger* и завершите процесс инициализации пешеходов. Теперь функция использует метод *random()* для случайного выбора рейса. Выбранный рейс сохраняется в параметре пешехода *flight*. Сам пешеход добавляется в коллекцию пассажиров данного рейса.

    Измените Java код в поле **Тело функции**:



Функция *dateToTime()* преобразовывает заданную дату в модельное время с учетом начальной даты эксперимента и выбранных единиц модельного времени.

Функция *add()* добавляет элемент в коллекцию.

Работа с содержимым коллекции

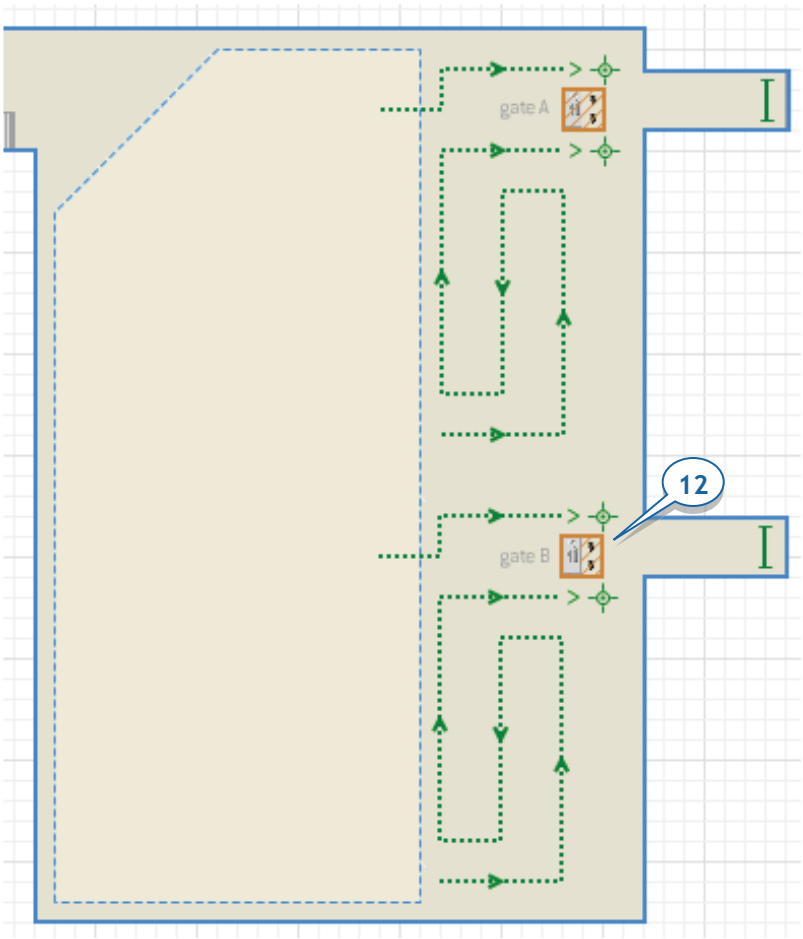
Используйте следующие функции для работы с содержимым коллекции:

- *int size()* – Возвращает количество элементов в коллекции.
- *boolean isEmpty()* – Возвращает *true*, если в коллекции нет элементов, иначе возвращает *false*.
- *add(element)* – Добавляет заданный элемент в коллекцию (на последнюю позицию).
- *clear()* – Удаляет все элементы из коллекции.
- *get(int index)* – Возвращает элемент коллекции с указанным порядковым номером (нумерация начинается с нуля).

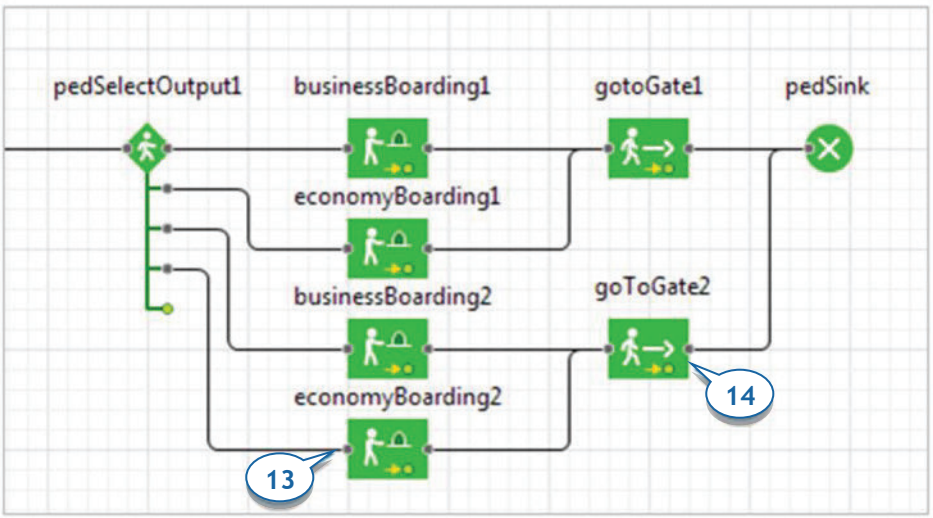
- *boolean remove(element)* – Удаляет заданный элемент из коллекции (если он в ней присутствовал). Возвращает *true*, если коллекция содержала этот элемент.
- *boolean contains(element)* – Возвращает *true*, если коллекция содержит заданный элемент.

12. Добавьте второй выход на посадку (гейт):

- Добавьте два элемента **Сервис с линиями**: *business2* и *economy2*.
- Добавьте **Прямоугольную стену**, 3D модели стола и служащих.
- Нарисуйте **Целевую линию** *gateLine2*.



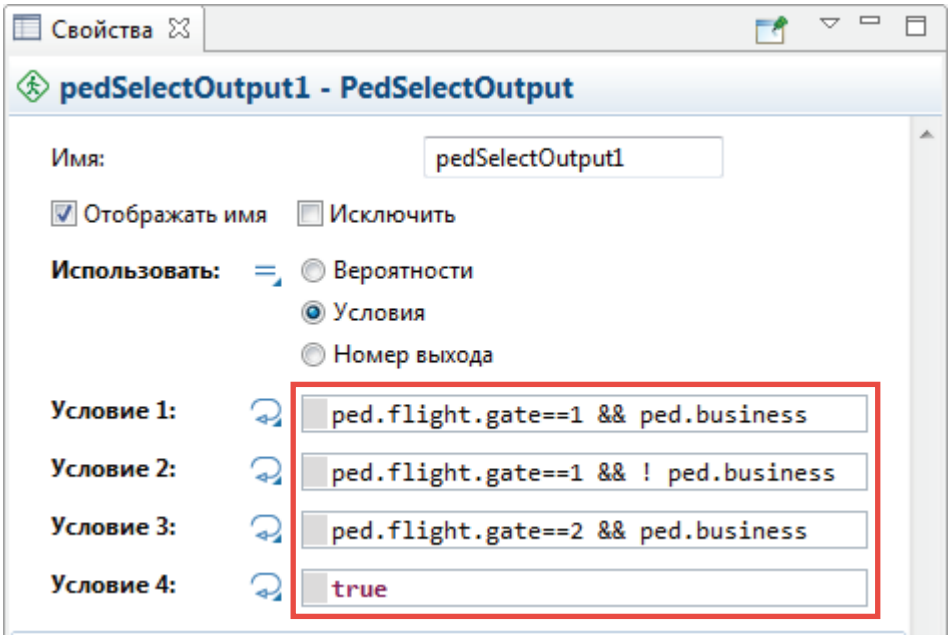
13. Добавьте еще два блока **PedService** : *businessBoarding2* и *economyBoarding2*. Соедините эти блоки с указанными ниже на рисунке выходными портами блока *pedSelectOutput1*. Теперь блок *pedSelectOutput1* будет перенаправлять пассажиров в один из четырех альтернативных выходных портов.



14. Добавьте еще один блок **PedGoTo** . Этот блок будет моделировать то, как пассажиры направляются на посадку ко второму гейту. Выберите *gateLine2* в свойстве **Целевая линия** этого блока. Соедините этот порт с ранее созданными блоками *businessBoarding2* и *economyBoarding2*.

15. В свойствах блока *businessBoarding2*, задайте **Сервисы**: *business2*. У блока *economyBoarding2*, укажите **Сервисы**: *economy2*. Для обоих этих блоков, задайте **Время задержки**: *uniform(2, 5)* **секунд**.

16. Теперь, когда мы добавили всю логику, связанную с выбором рейса, давайте изменим условия блока *pedSelectOutput1*, чтобы пассажиры направлялись к нужному им гейту.



Теперь давайте перейдем к заданию логики вылетающих рейсов. Для этого мы воспользуемся динамическими событиями, с помощью которых мы зададим действия, связанные с объявлением посадки на рейсы и вылетом рейсов.

Динамические события

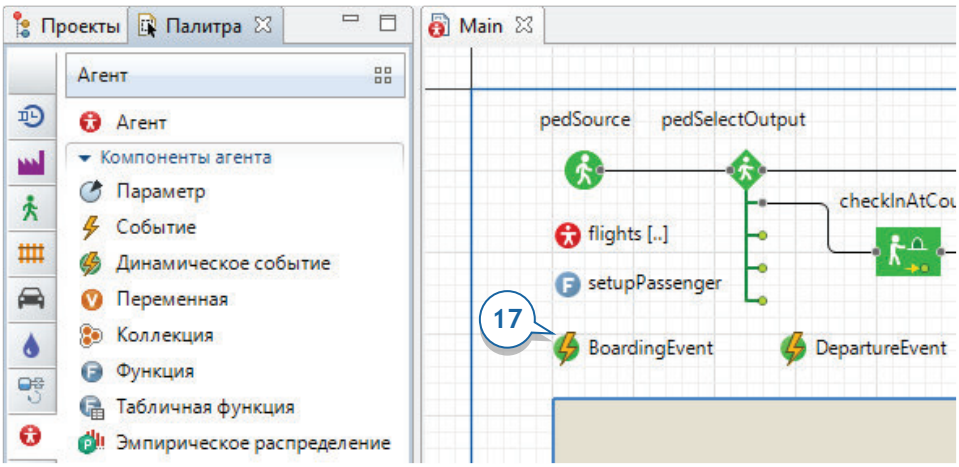
Динамические события позволяют планировать выполнение заданных действий в будущем, в определенные моменты времени.

Используйте динамические события:

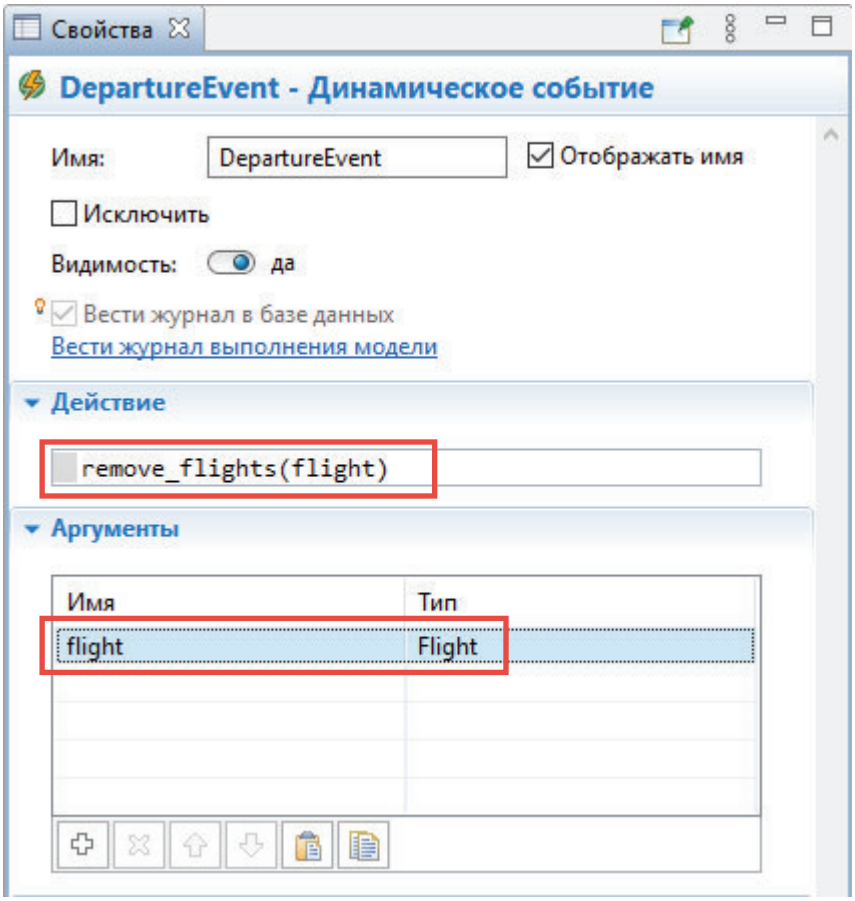
- Если в модели предполагается планирование большого количества однотипных событий (возможно, даже запланированных одновременно).
- Если действие события зависит от специфической информации (в этом случае можно передать событию необходимую информацию с помощью параметров этого события).

> **Поскольку AnyLogic задает шаблон динамического события в виде Java класса, то имена динамических событий должны начинаться с заглавной буквы (в соответствии с соглашением о наименовании объектов в Java).**

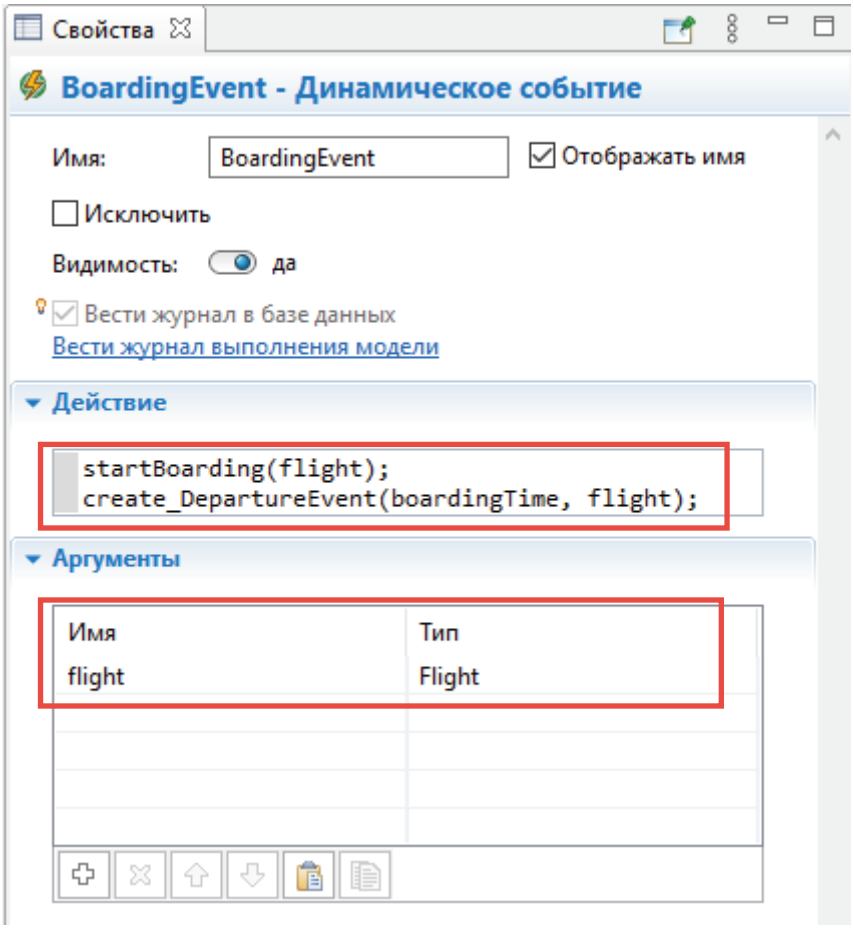
17. Добавьте на диаграмму *Main* два **Динамических события** из палитры **Агент**.



18. Динамическое событие *DepartureEvent* планирует вылет самолета путем удаления рейса из популяции предстоящих рейсов. Настройте это динамическое событие так, как показано на рисунке ниже.



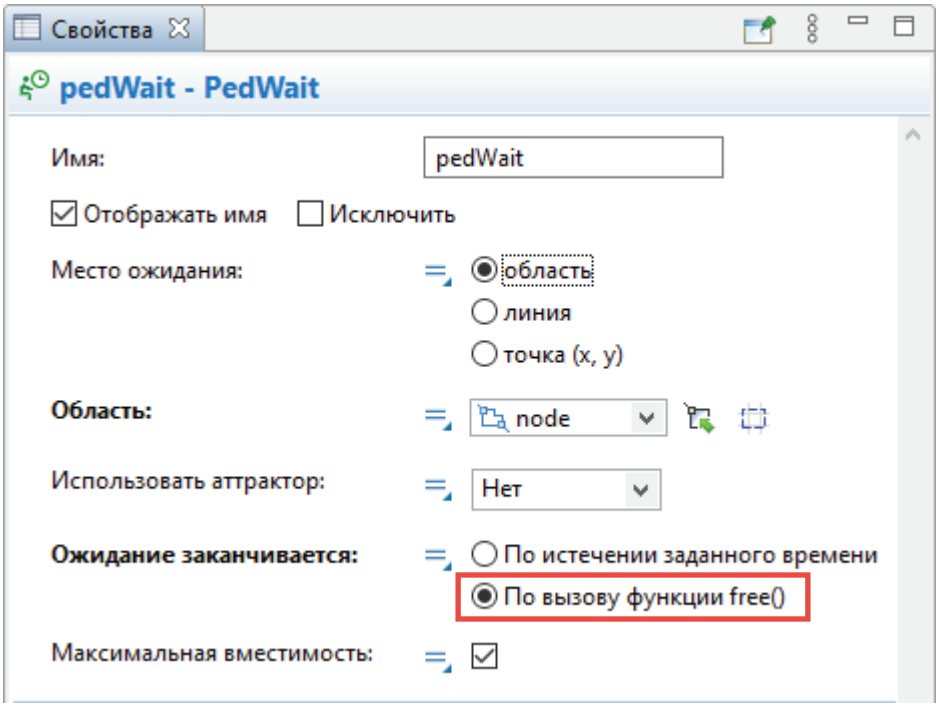
19. Другое динамическое событие, *BoardingEvent*, планирует начало посадки на рейс и затем создает экземпляр динамического события *DepartureEvent*, которое планирует вылет самолета через 40 минут после начала посадки.



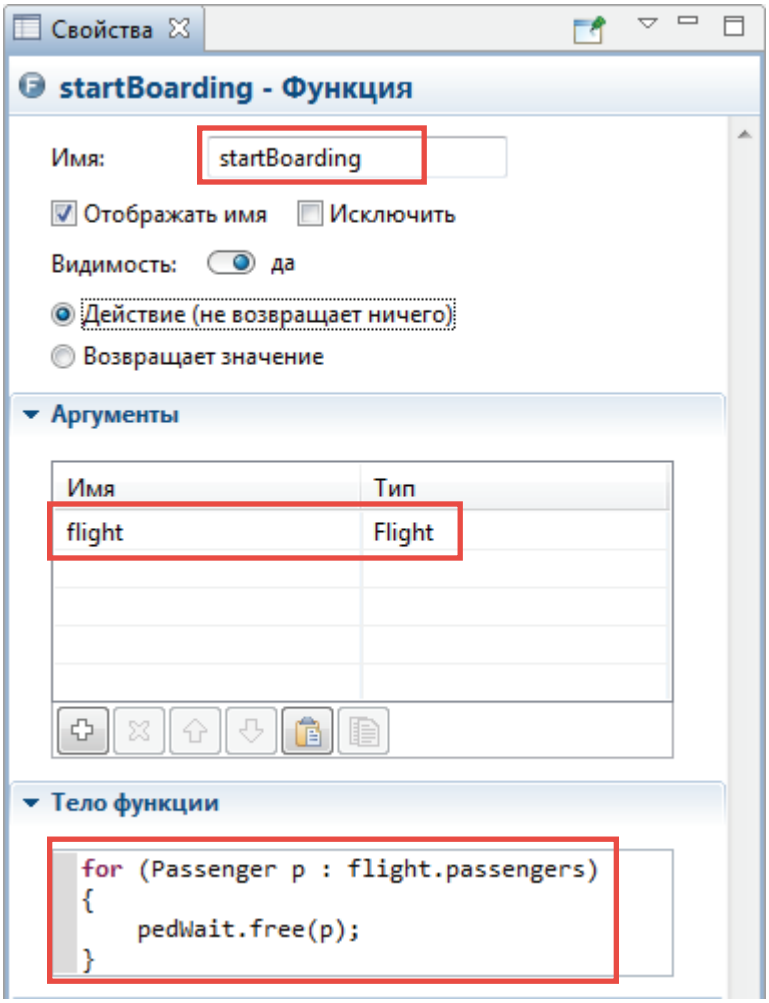
Чтобы запланировать новое динамическое событие (начать отсчет времени до его происхождения), нужно вызвать функцию *create_<ИмяДинамическогоСобытия>*. В нашем случае имя функции будет *create_DepartureEvent()*. В качестве аргумента нужно передать длительность таймаута – количество единиц модельного времени, через которое это событие произойдет (отсчет начнется с момента вызова этой функции). После этого, обязательного, аргумента могут следовать значения заданных у динамического события аргументов. В нашем случае у события один аргумент – ссылка на рейс *flight*.

20. У блока *pedWait* измените значение параметра **Ожидание заканчивается** со значения **По истечении заданного времени** на **По вызову функции free()**.

Теперь пассажиры будут находиться в области ожидания, пока не услышат объявление о начале посадки на рейс.

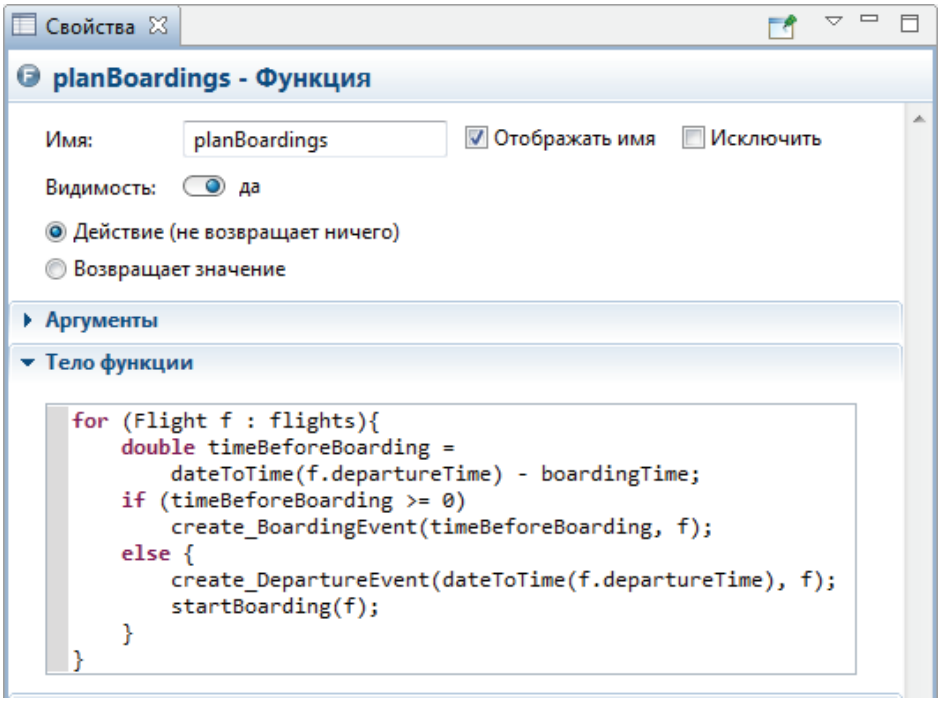


21. Добавьте **Функцию** *startBoarding*, которая будет моделировать начало посадки на рейс. Эта функция проходит в цикле по всем пассажирам указанного рейса и завершает для них процедуру ожидания путем вызова функции *free()* блока *pedWait*.



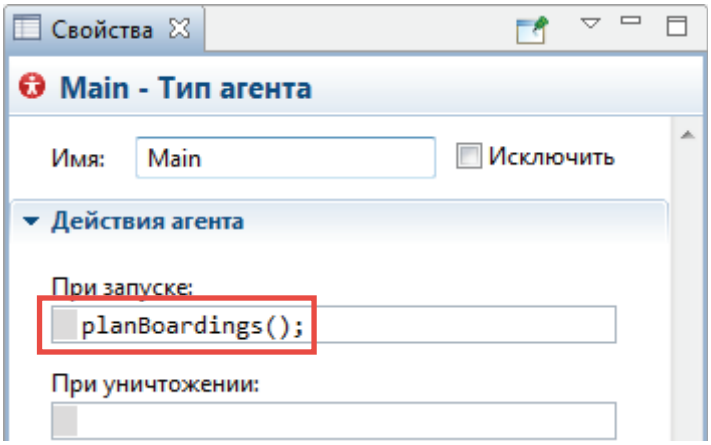
В нашем случае мы используем цикл *for* для того, чтобы пройти по всем элементам коллекции *passengers* (пассажирам заданного рейса *flight*). Все пассажиры данного рейса завершат ожидание в блоке *pedWait* и покинут область ожидания у гейта.

22. Создайте функцию *planBoardings*, которая запланирует посадку на все рейсы. Эта функция итеративно проходит в цикле по всем вылетающим рейсам - агентам популяции *flights*.



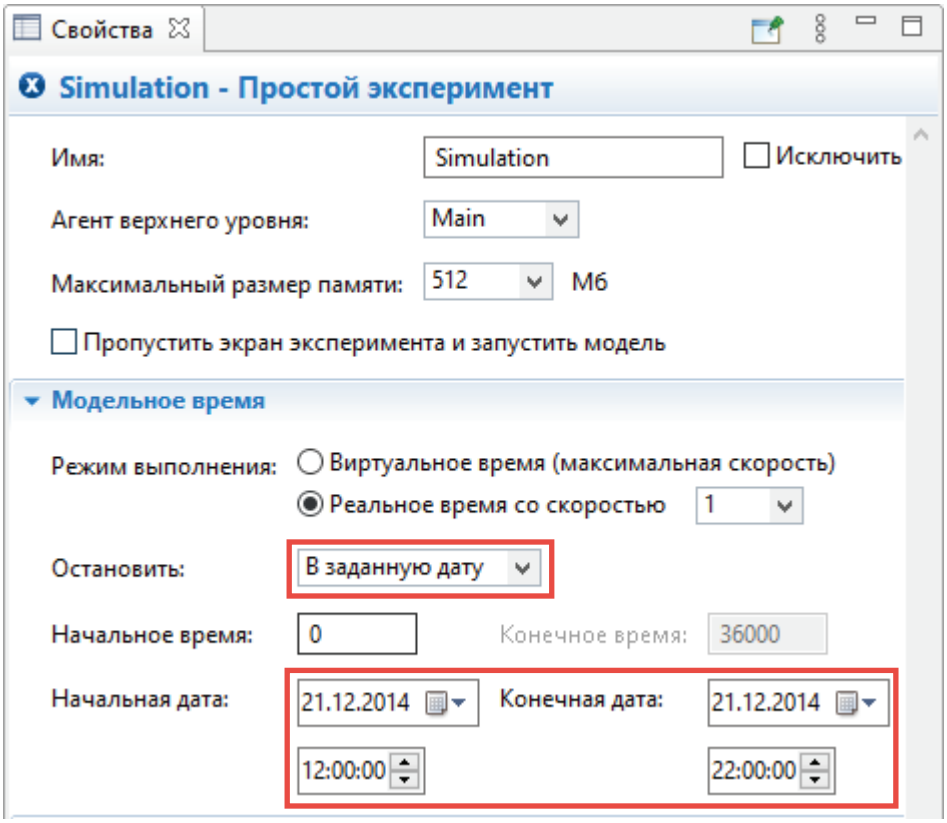
Оператор принятия решения *if* проверяет заданное условие. Если посадка на данный рейс еще не начата, то происходит планирование динамического события *BoardingEvent* на определенный момент времени в будущем. В противном случае происходит планирование вылета, и вызывается функция *startBoarding*, разрешающая посадку (при этом ссылка на рейс передается с помощью аргумента функции).

23. В секции свойств **Действия агента** *Main*, в поле **При запуске**, поместите вызов функции *planBoardings()*.

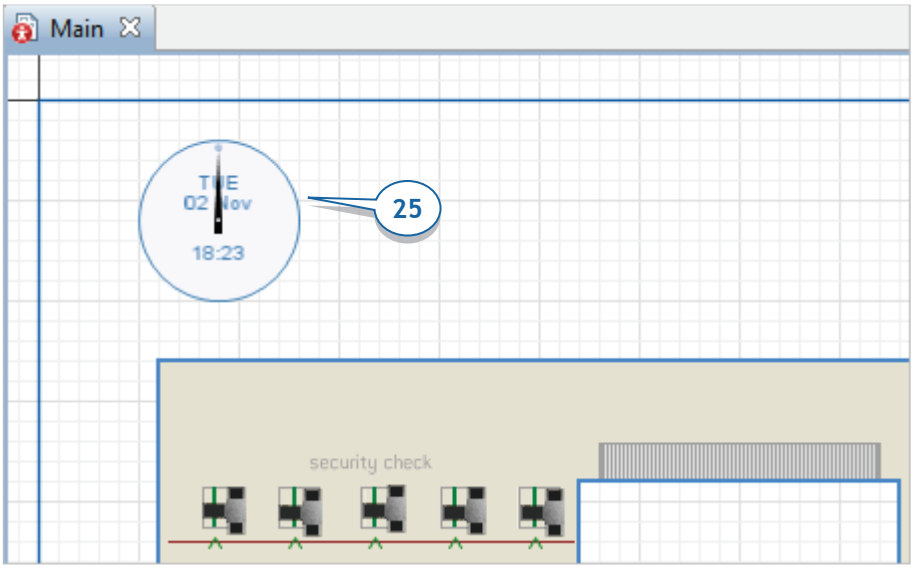


Теперь нам осталось лишь задать у эксперимента ту же начальную дату, что используется в таблице БД.

24. В панели **Проекты**, выберите эксперимент *Simulation*. Откройте раздел свойств эксперимента **Модельное время**. Задайте **Начальную дату**: *21/12/2014, 12:00:00*. В списке **Остановить**, выберите опцию **В заданную дату**, а затем задайте **Конечную дату**: *21/12/2014, 22:00:00*.



25. Для того, чтобы иметь представление о модельном времени по ходу выполнения модели, добавьте на графическую диаграмму **Часы** из палитры **Картинки**.

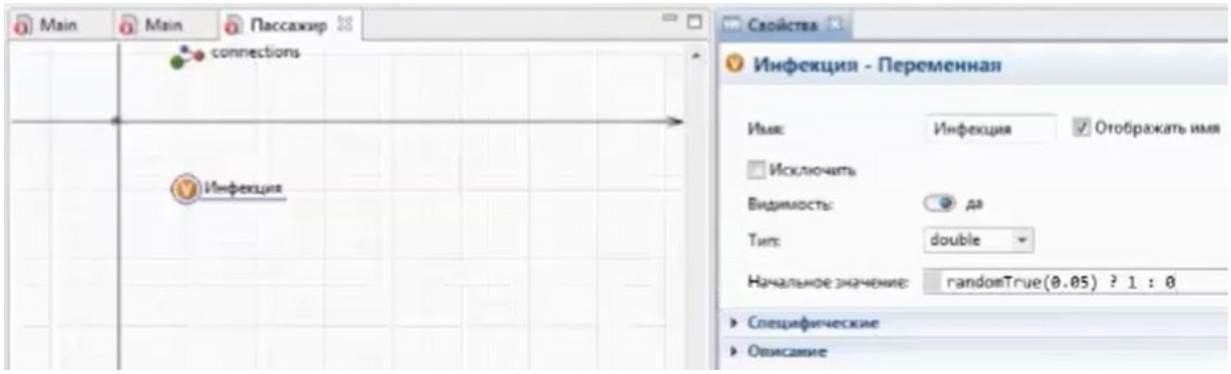


26. Запустите модель. Вы увидите, как пассажиры ожидают объявления о начале посадки на рейс, после чего направляются к гейту для прохождения процедуры проверки посадочных талонов.

Фаза 7. Добавление индивидуального поведения агенту

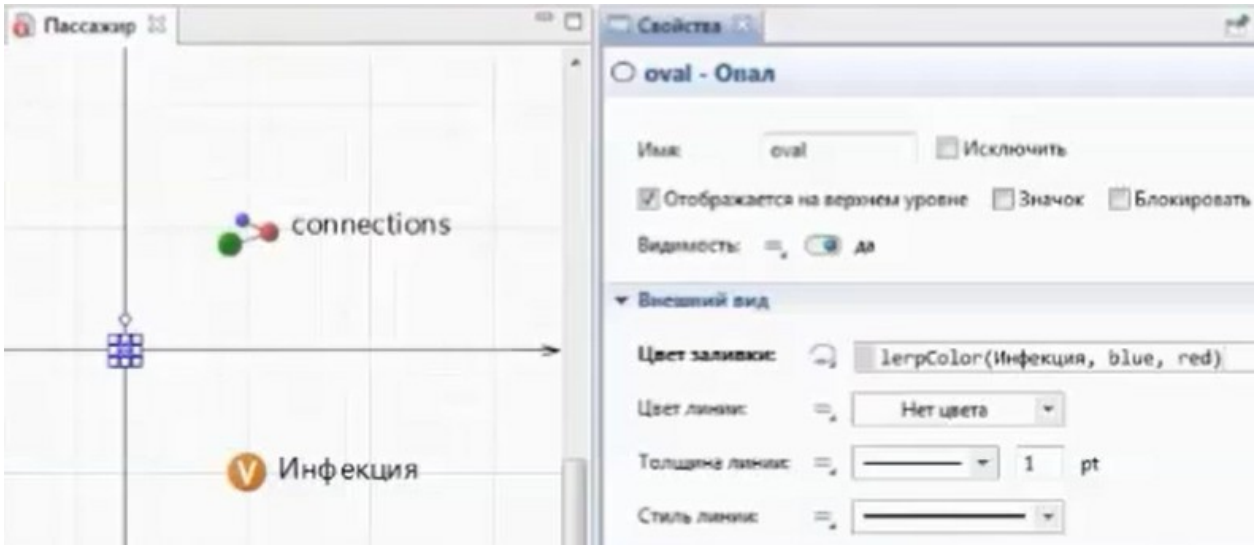
Пассажиры могут иметь собственное поведение, сделаем пассажира более живым человеком. План такой: скажем, что некоторые из пассажиров больны гриппом. Аэропорт - это место, где люди находятся рядом друг с другом, и возможно распространение инфекции.

Добавим пассажиру (Passenger) внутреннюю переменную «Инфекция». Это переменная, и если ее значение 0 - человек здоров, если 1 - болен максимально. Начальное значение будет случайным, есть такая функция как ***randomTrue()*** и с вероятностью 5% (0.05) человек болен.



Чтобы увидеть это в анимации, сделаем фигурки пассажира с добавлением кружка (Oval). Например, с радиусом 5. Цвета линии у кружка нет, а цвет заливки у него зависит от степени болезни. Значением дляцвета будет плавный переход от синего«Здоров» к красному «Болен».

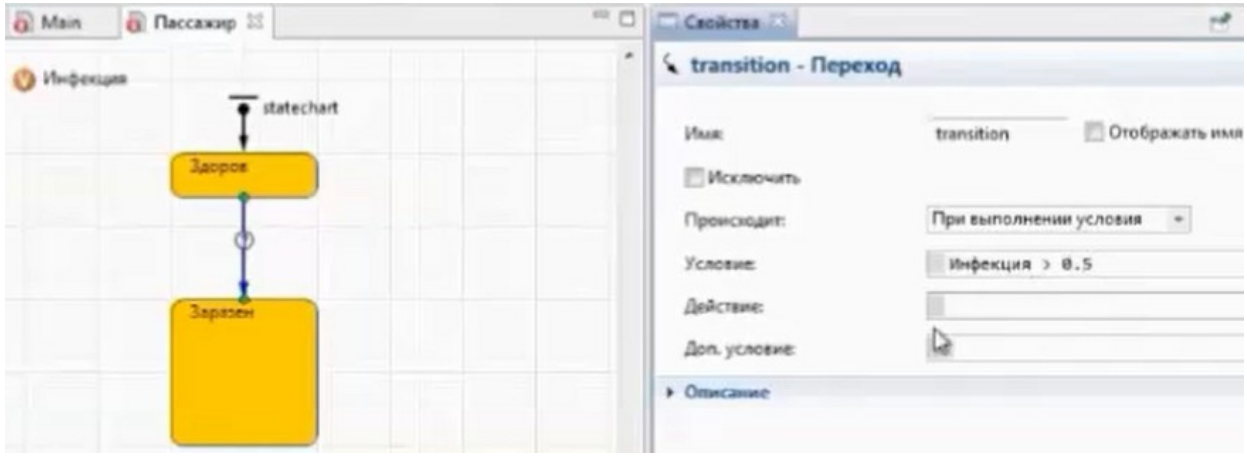
Для этого запишем в строке выбор цвета с использованием такой функции ***lerpColor()***, где первым аргументом будет «Инфекция» и если инфекция 0 будет использоваться ответ blue (синий), а если инфекция 1 – red (красный).



Для других значений инфекции цвет будет иметь промежуточное значение, плавный переход от синего к красному. Далее, этот кружочек будем видеть только в двумерном плане, и переходим на закладку трехмерного человечка – и его 3Dobject будем видеть только в 3D, поэтому сделаем его статическим.

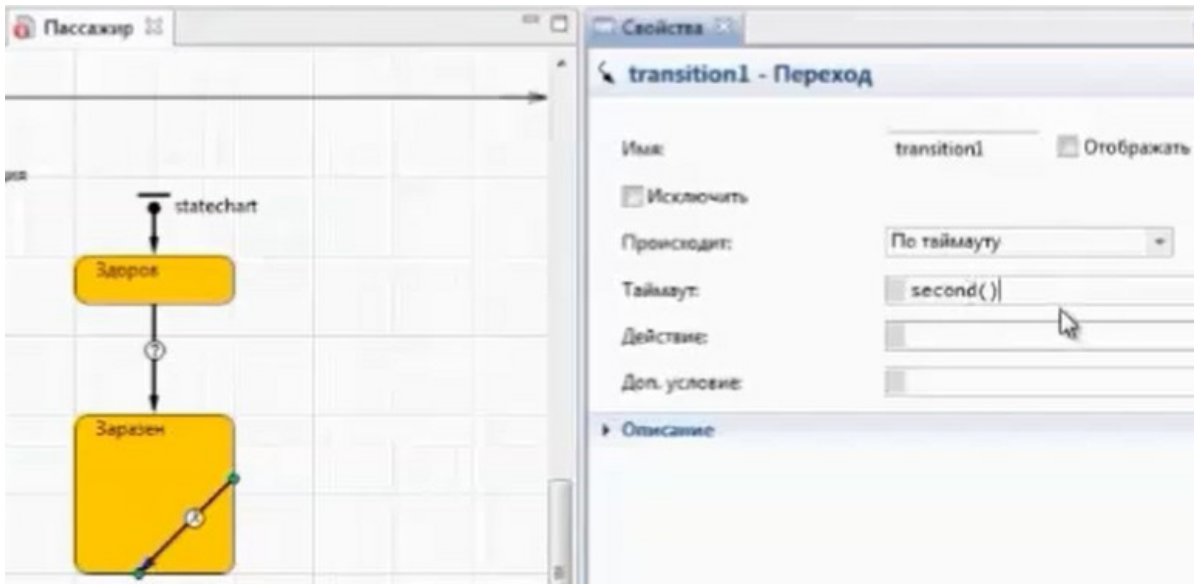
Если теперь мы запустим модель, то кружки в основном будут синими, а некоторые должны быть красными. С большой вероятностью человек здоровый, но если он болен, то пока он это не знает и никого не заражает. Как мы моделируем распространение инфекции? Например, пересылкой сообщений между близко стоящими. Для этого давайте сделаем из палитры диаграмму состояний

или по-английски ***Statechart***. Добавим два состояния: первое состояние «Здоров» и второе «Заразен». Переход из Здоров к Заразен будет осуществляться, если уровень инфекции в организме превышает 0,5.



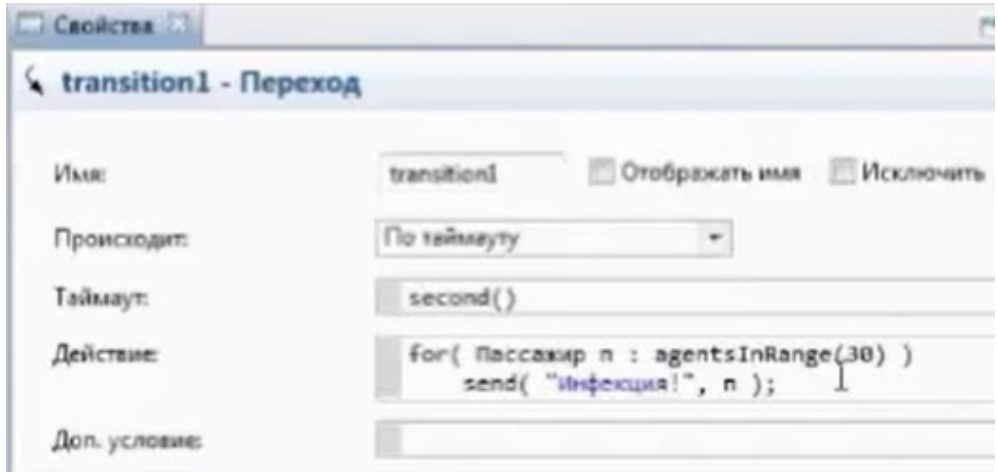
Обратите внимание, что на переходе между состояниями есть иконка, которая соответствует типу перехода. Если часы, то значит по таймауту, а здесь нам нужно по условию. В данном случае, как только Инфекция равна 0,5 , то в этом состоянии он распространяет инфекцию. Это мы будем производить циклической посылкой сообщений всем, кто находится на определённом расстоянии от данного пассажира.

Давайте у нас это будет секунда – функция ***second()***. Эта посылка будет происходить через секунду, а не через единицу модельного времени (1). Эту функцию нужно вписать, если требуются более частые посылки сообщений.



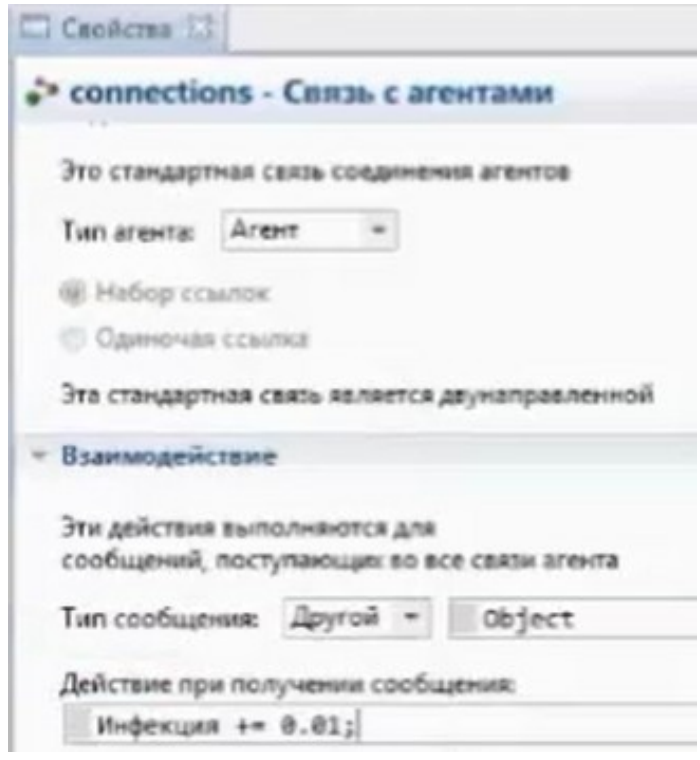
Обратите внимание, что пешеходные агенты (*passenger*) все находятся в одном пространстве. То есть у каждого объекта в модели есть всегда координаты X и Y. Они все согласованы, поэтому можно написать следующее выражение:

для всех пассажиров, которые находятся не дальше чем на дистанции, например, 20 пикселей, то всем этим пассажирам приходит сообщение «инфекция!». Это первая часть взаимодействия.



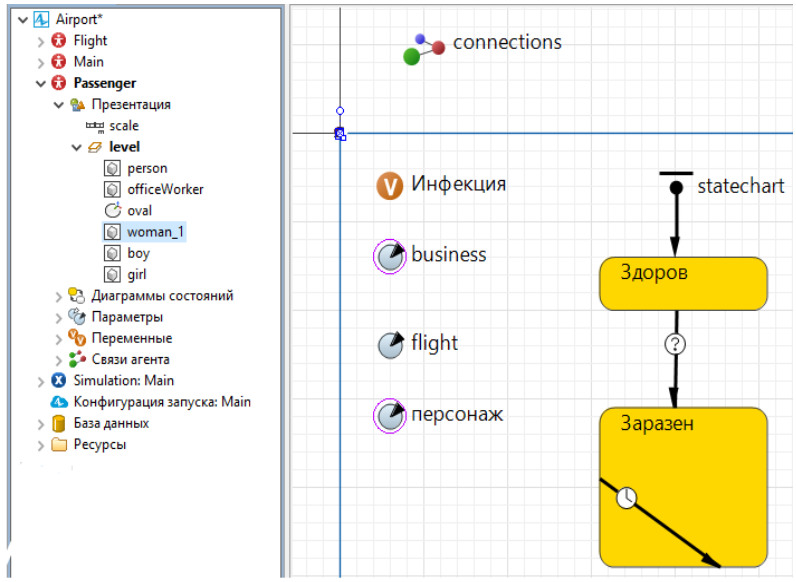
Вторая часть - это определение реакции на получение сообщения «инфекция!». Специальный элемент **connections** есть в каждом агенте, и он годится для задания связей между агентами. В данном случае между пассажирами нет постоянной связи, т.е. мы считаем - они все не знакомы. ***Connections*** может использоваться для обеспечения взаимодействия, получения сообщений.

Здесь будет такое единственное сообщение - «Инфекция!». Таким образом, действием при получении сообщений будет увеличение внутренней переменной «Инфекция» на какую-то величину, например, на = 0.01 , или лучше = 0.002.

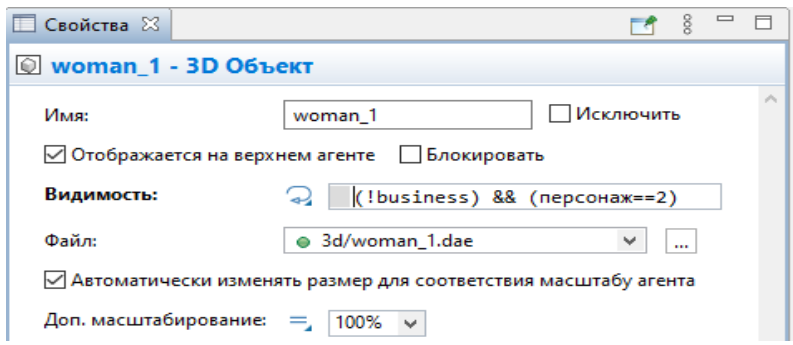


Все агенты и в пешеходной модели являются такими же полноправными агентами. В AnyLogic все агенты могут иметь собственное поведение, в данном случае, поведение состояло из диаграммы состояний. Но, в принципе, в агенте можно определить, всё что угодно, например, ещё один процесс.

Для разнообразия представления пассажиров добавим параметр Персонаж для Passenger.
Тип параметра - int; Начальное значение - uniform_discr(1,4).



Добавим дополнительные варианты презентации пассажиров - вставьте из панели 3D объектов фигурки Женщина1, Мальчик, Девочка. Поместите эти фигуры в точку начала координат (0,0) диаграммы агента пешехода Passenger.
В панели проекта раскройте в дереве объектов агента Passenger ветвь Презентация, и в разделе объекта Level выбирайте привязанные 3D-объекты для настройки свойства Видимость в виде выражения (!business) && (персонаж==1), изменяя значение 1,2,3,4.



Перетащите элемент Карта плотности пешеходов из секции Разметка пространства палитры Пешеходная библиотека в схему агента Main. В отличие от других объектов библиотек, у этого объекта отображается шкала, и на анимации отображается легенда карты. Легенда карты плотности помогает понять, какие цвета соответствуют каким значениям плотности.
Откройте панель Свойства этого блока и сбросьте флажок Отображать имя.

Если вас не устраивает плотность Карты на чертеже тем, что его слишком ярко видно, можете увеличить степень прозрачности отображаемой на анимации карты плотности с помощью бегунка Прозрачность в свойствах элемента Карта плотности пешеходов.

Дополнительно, настроим более индивидуальную реакцию на инфекцию. Предположим, пассажир с номером персонажа =2 будет иммунен. Т.е. он не будет подвержен заражению. Для этого в модели при передаче сообщения Инфекция в настройке элемента связи с агентами connections в действии при получении сообщения запишем:

```
Инфекция += ((персонаж == 2) ? 0 : 0.002);
```

Тогда пассажиры персонажа =2 не будут подвергаться заражению, т.е. менять значение параметра Инфекция.