

Модель сортировочной горки

Эта презентация является частью
стандартной программы обучения



Модель сортировочной горки

- Постройте имитационную модель сортировочной горки. Сортировочная горка – это вид сортировочной станции, использующий для перемещения вагонов земное тяготение, то есть скатывание вагонов и групп вагонов с уклона.
- Прибывающий поезд содержит 5 разных типов вагонов, которые отправляются на 5 путей назначения. Как только на пути накапливается 9 вагонов одного типа, из них формируется новый поезд, который покидает этот путь.



Сортировочная горка. Фаза 1

- Давайте сначала промоделируем, как вагоны отправляются на путь, который называют **стрелочной горловиной**. Оттуда вагоны отправляются через серию стрелок, называемую **стрелочной улицей**, на пути сортировки.



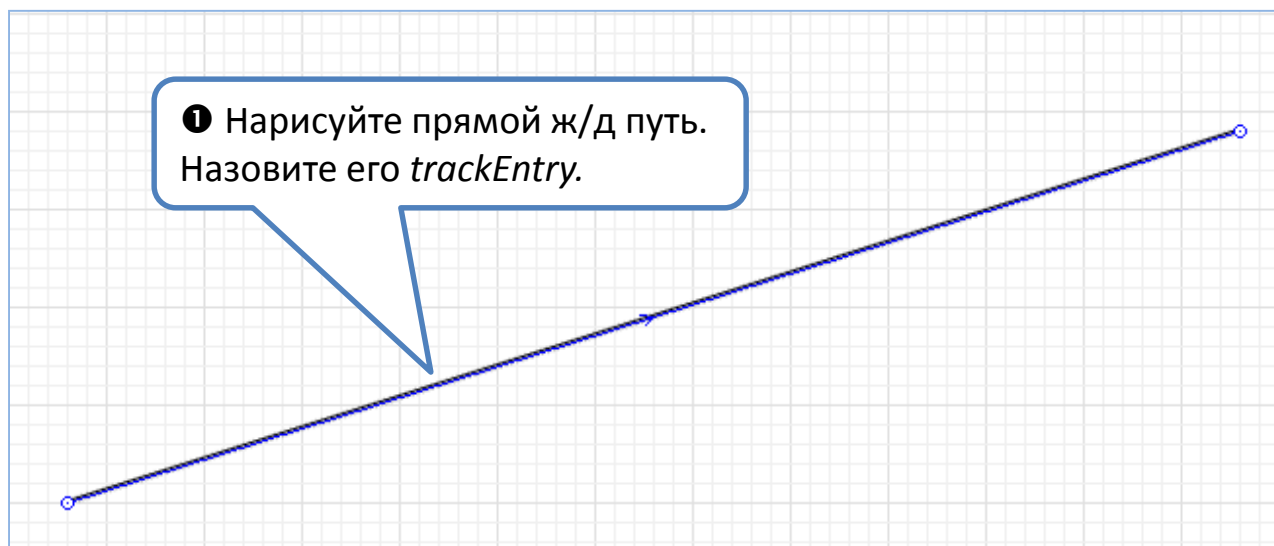
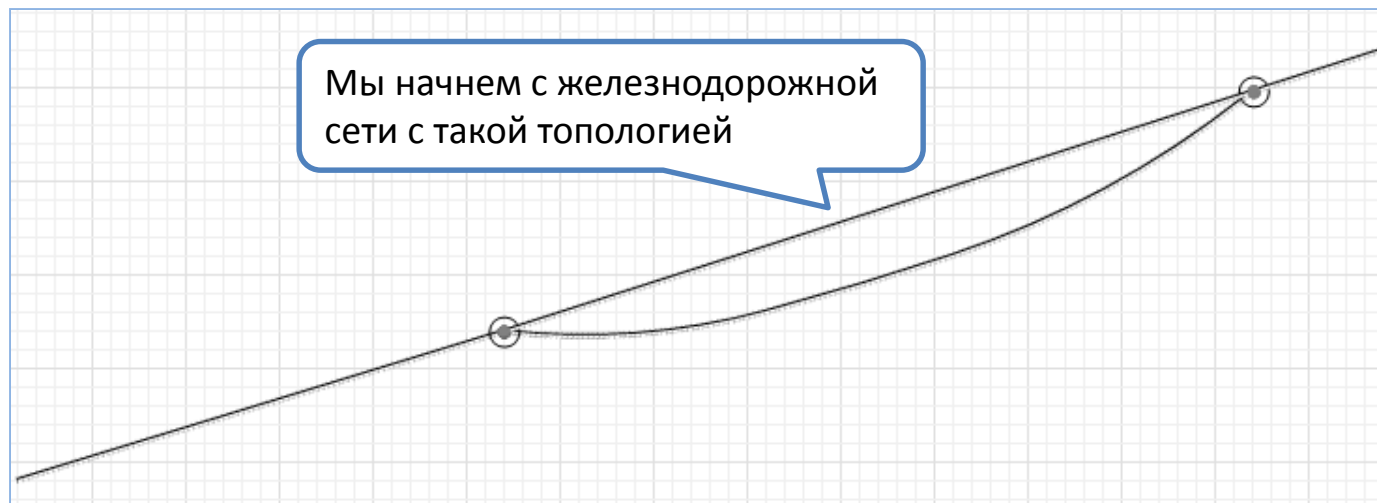
Мы построим модель с помощью *Железнодорожной библиотеки AnyLogic*.

Железнодорожная библиотека

- *Железнодорожная библиотека* позволяет эффективно моделировать и визуализировать функционирование железнодорожных узлов и железнодорожных транспортных систем любого уровня сложности и масштаба. Сортировочные станции, пути погрузки/разгрузки больших предприятий, железнодорожные станции и вокзалы, станции метрополитена, шаттлы аэропортов, пути на контейнерных терминалах, движение трамваев и даже рельсовая транспортировка в угольных шахтах - все эти задачи могут быть легко и точно промоделированы с помощью *Железнодорожной библиотеки*.
- *Железнодорожная библиотека* интегрирована с другими библиотеками AnyLogic – *Библиотекой моделирования процессов* и *Пешеходной библиотекой*, что позволяет соединять железнодорожные модели с моделями, более детально рассматривающими движение грузовиков, кранов, кораблей, моделями пассажиропотоков, производственных и бизнес-процессов и т.д.



Сортировочная горка. Фаза 1. Шаг 1



Создайте новую модель. Назовите ее *Hump Yard*. Задайте топологию путей. Сначала нарисуйте ж/д пути.

Начните с прямого пути. Поезда будут появляться на этом пути, поэтому мы назовем его *trackEntry*. Мы будем давать «говорящие» имена только тем путям, на которые будут ссылаться блоки диаграммы процесса. Остальные пути будут называться по умолчанию.

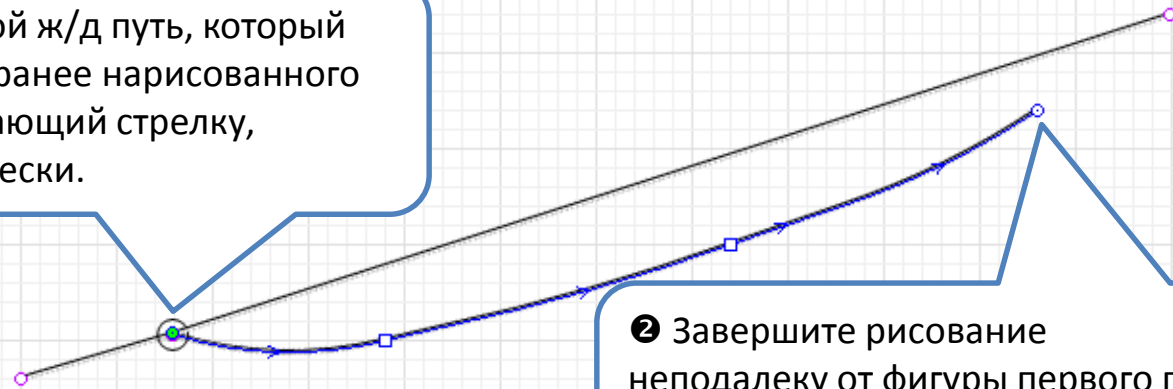
Рисование прямых ж/д путей

- Ж/д пути рисуются так же, как и пешеходные пути. Сначала выделите двойным щелчком элемент **Ж/д путь** в секции *Разметка пространства* палитры *Железнодорожной библиотеки*.
- Щелкните в любой точке графического редактора, чтобы начать рисовать путь.
- Чтобы нарисовать прямой сегмент пути, добавьте щелчком мыши конечную точку сегмента.
- Завершите рисование двойным щелчком.

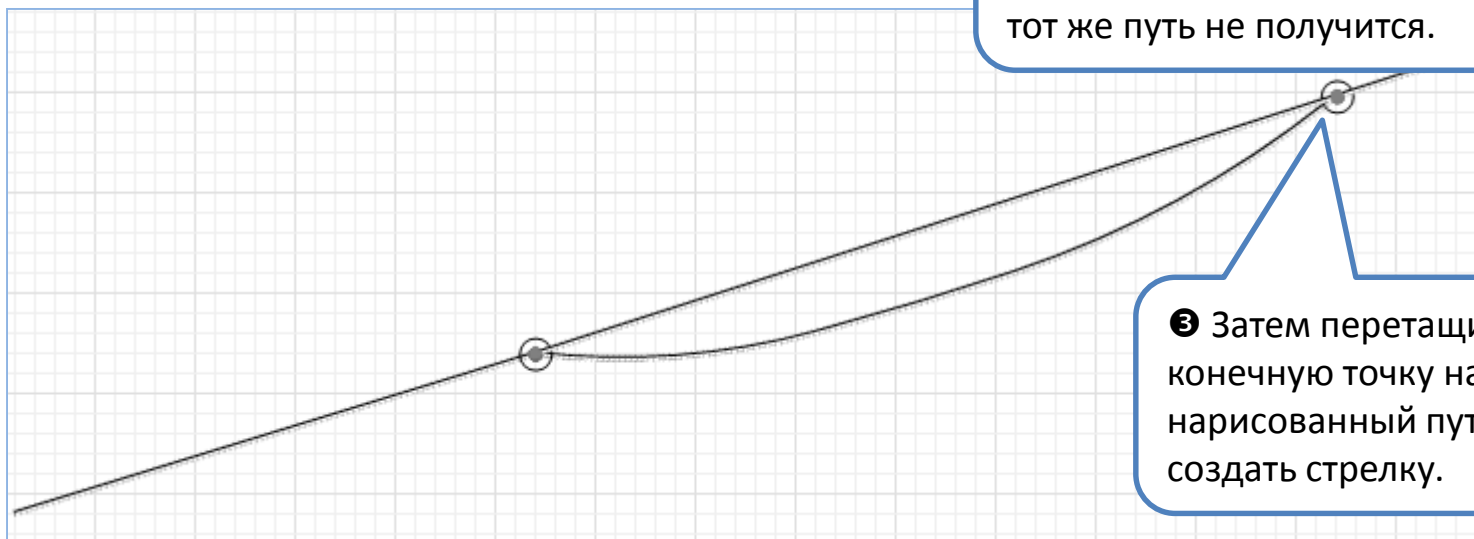


Сортировочная горка. Фаза 1. Шаг 2

❶ Нарисуйте дуговой ж/д путь, который начинается в точке ранее нарисованного пути. Круг, обозначающий стрелку, появится автоматически.



❷ Завершите рисование неподалеку от фигуры первого пути. Сразу поставить конечную точку на тот же путь не получится.



❸ Затем перетащите конечную точку на ранее нарисованный путь, чтобы создать стрелку.



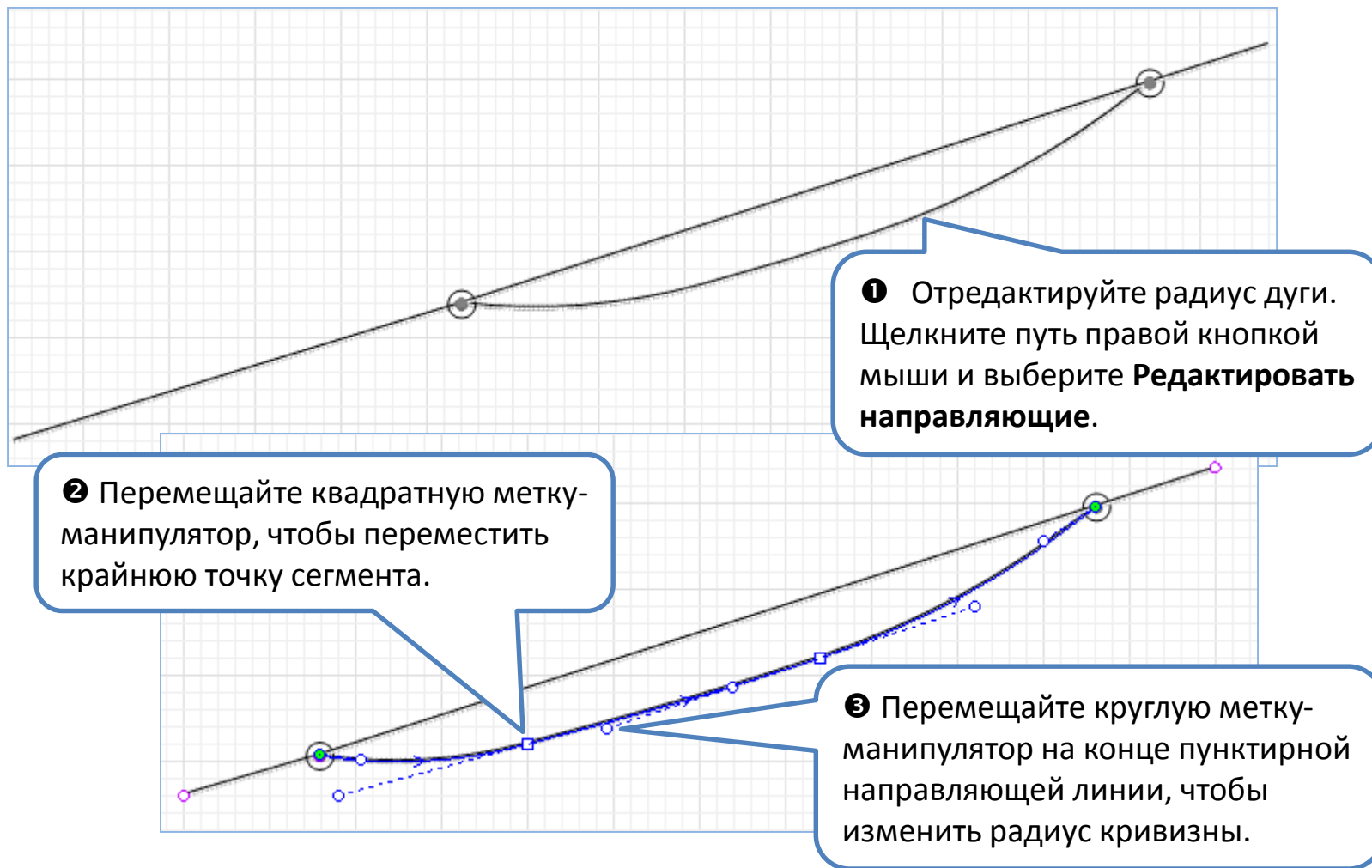
На предыдущем шаге мы нарисовали прямой путь. Но пути могут также быть дуговой формы. Давайте теперь нарисуем дуговой путь – *стрелочную горловину*.

Рисование дуговых ж/д путей

- Выделите двойным щелчком элемент **Ж/д путь** в палитре *Железнодорожной библиотеки*. Затем щелкните в любой точке графического редактора, чтобы начать рисовать путь.
- Чтобы добавить дуговой элемент, не отпускайте кнопку мыши после щелчка и двигайте курсором. Вы увидите, как изменяется радиус дуги. Отпустите кнопку мыши, когда этот сегмент готов и можно добавить следующий.
- Завершите рисование двойным щелчком.

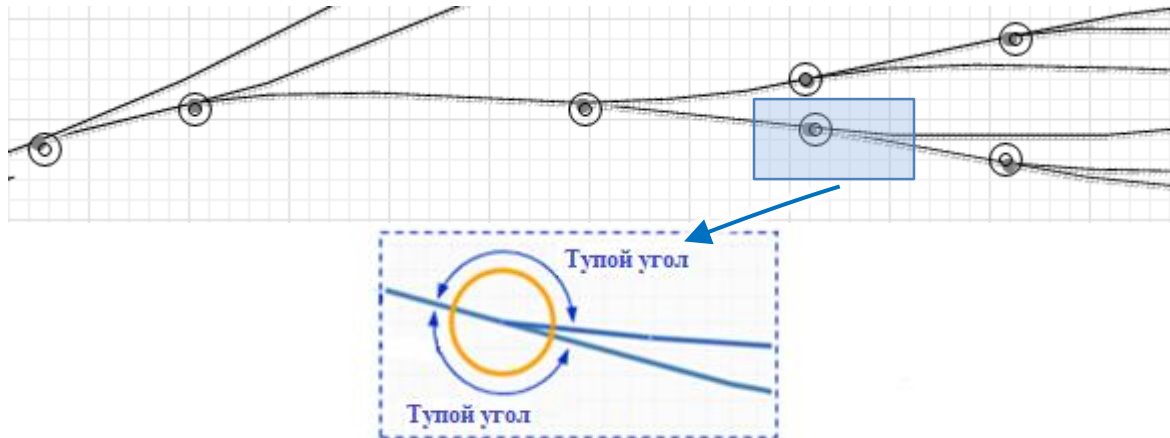


Сортировочная горка. Фаза 1. Шаг 3

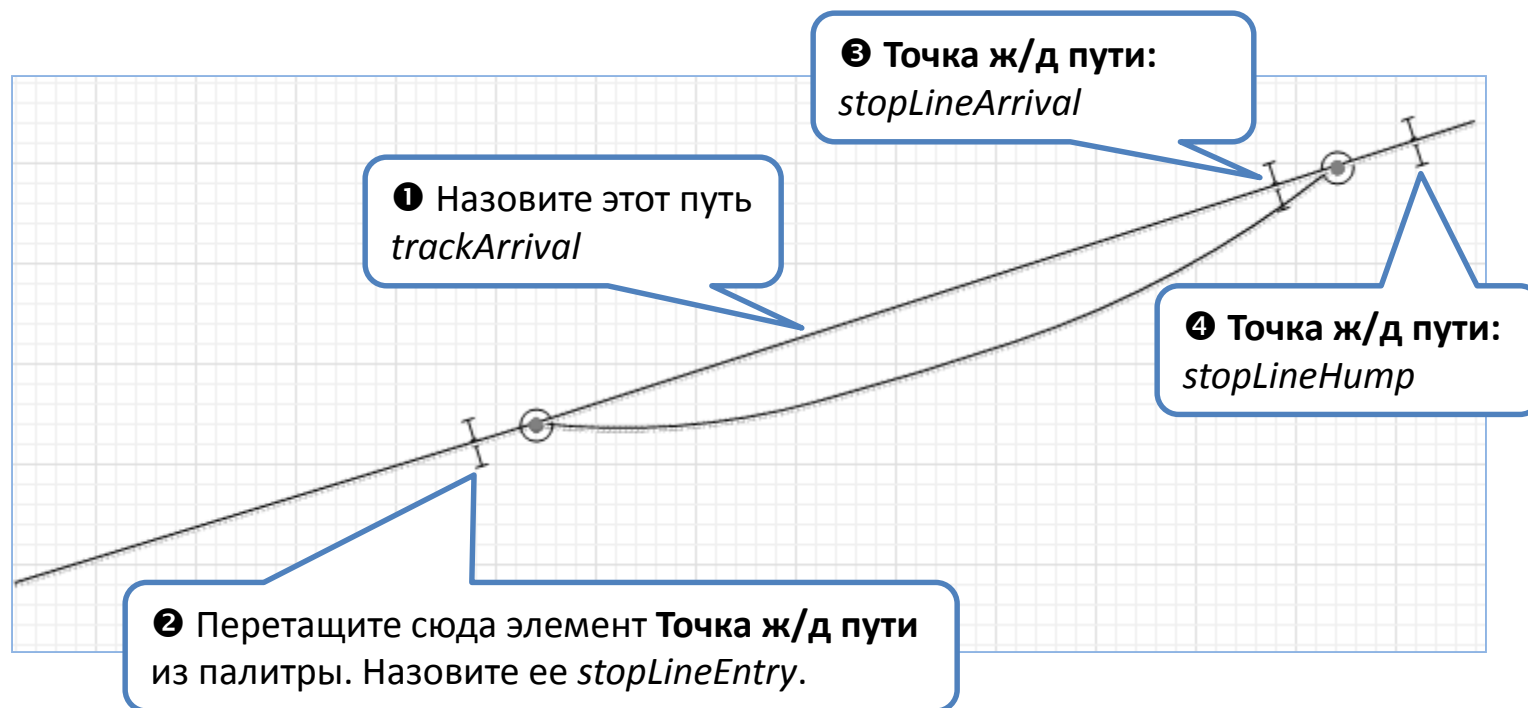


Железнодорожные стрелки

- Железнодорожная сеть состоит из путей и стрелок. Рисуя пути, Вы можете заметить, как стрелки (круги) появляются в точках соединения путей.
- Существуют правила, объясняющие появление стрелки:
 - В каждой стрелке должны сходиться ровно три конечные точки путей.
 - При каждой стрелке должно быть два тупых угла из трех между путями. Стрелка определяет маршрут по этим углам.



Сортировочная горка. Фаза 1. Шаг 4

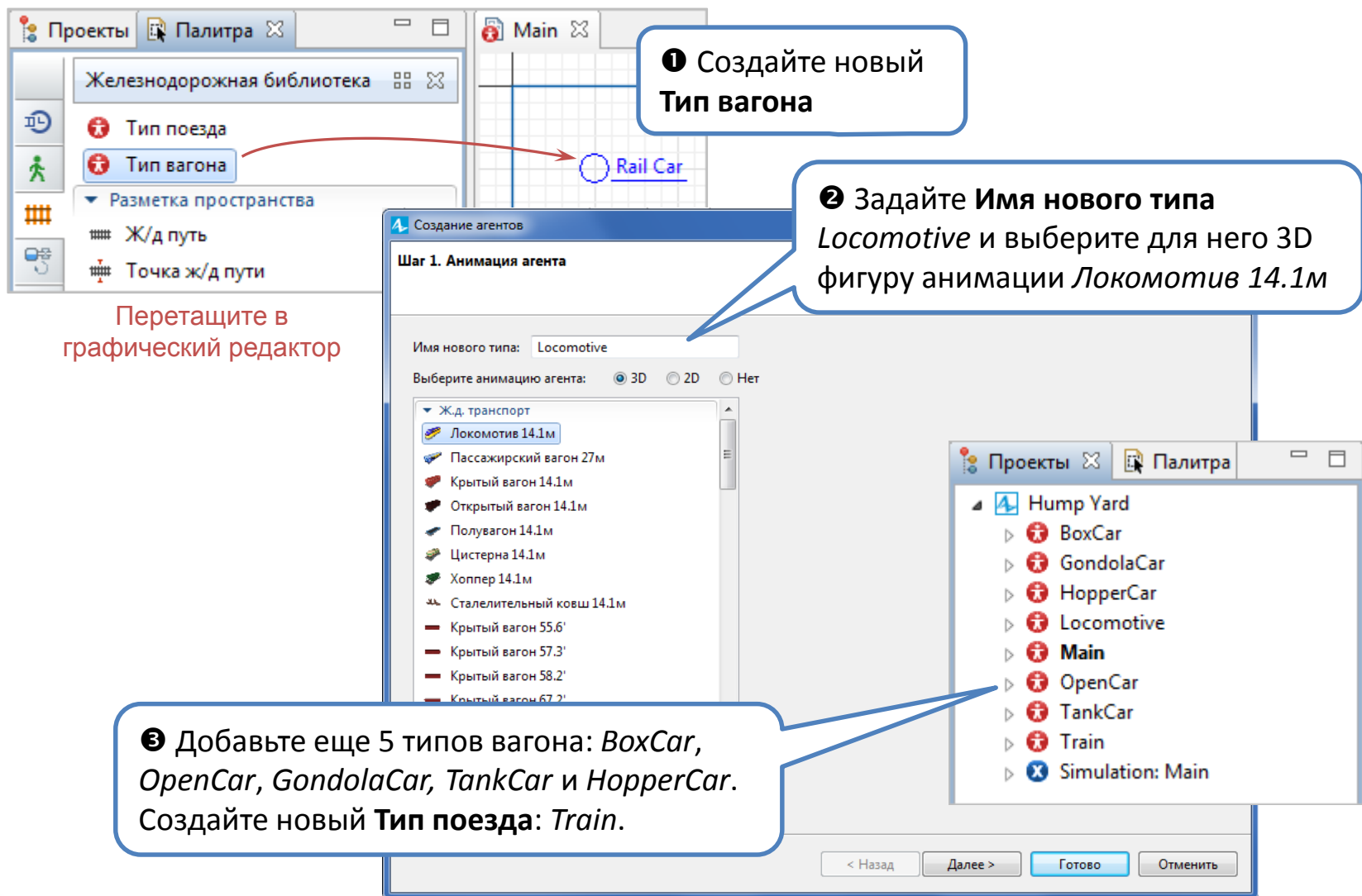


Точка ж/д пути

- **Точка ж/д пути** является элементом разметки пространства, используемом, чтобы задать точную позицию на ж/д пути. Это может понадобиться, когда Вы задаете:
 - *Позицию на пути, где появляется поезд*
 - *Позицию на пути, где поезд должен остановиться*



Сортировочная горка. Фаза 1. Шаг 5



1 Создайте новый
Тип вагона

2 Задайте **Имя нового типа**
Locomotive и выберите для него 3D
фигуру анимации *Локомотив 14.1м*


3 Добавьте еще 5 типов вагона: *BoxCar*,
OpenCar, *GondolaCar*, *TankCar* и *HopperCar*.
Создайте новый **Тип поезда**: *Train*.


Перетащите в
графический редактор



Создайте 6 типов вагона:

Locomotive -  Локомотив 14.1м

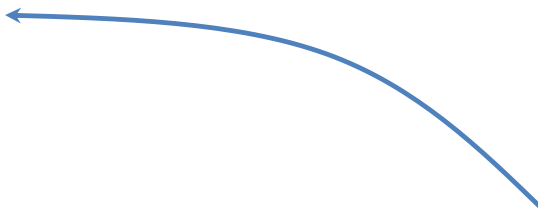
BoxCar -  Крытый вагон 14.1м

OpenCar -  Открытый вагон 14.1м

GondolaCar -  Полувагон 14.1м

TankCar -  Цистерна 14.1м

HopperCar -  Хоппер 14.1м



При создании новых типов вагона выберите соответствующую 3D фигуру анимации из списка.

Таким же образом создайте тип поезда (перетащив элемент **Тип поезда** из палитры). Назовите его *Train*. Типу поезда в нашей модели не нужна фигура анимации: поезд будет состоять из вагонов.

Создание нового типа вагона

- Мы предоставляем Вам удобный Мастер создания типа вагона. Просто укажите имя нового типа вагона и выберите фигуру анимации из списка готовых 3D объектов.



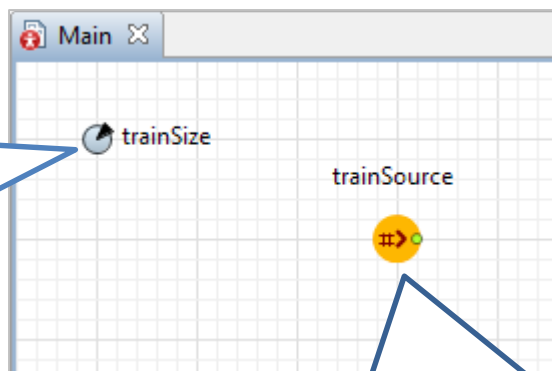
Сортировочная горка. Фаза 1. Шаг 6

❶ Добавьте параметр:

Имя: *trainSize*

Тип: *int*

Значение по умолчанию: 9



❷ *TrainSource*

Время между прибытиями: 15 минут

Кол-во вагонов (включая локомотив): *trainSize*

Точка входа задается как: Смещение на пути

Путь: *trackEntry*

Смещение: от начала пути

Смещение первого вагона: $trainSize * 15 + 15$ м

▼ **Поезд и вагоны**

Новый поезд: *Train*



Задайте логику процесса с помощью блоков *Железнодорожной библиотеки AnyLogic*.

- ② Начните диаграмму с блока ***TrainSource***, который создает поезда. Поезда будут появляться на ж/д пути *trackEntry*.

Блок Ж/д библиотеки	Описание
Train Source	Создает поезда, выполняет начальную настройку и помещает их в железнодорожную сеть. Начинает любую ж/д диаграмму процесса. Поддерживает несколько типов расписаний прибытия.
Train Dispose	Удаляет поезда из модели.
Train Move To	Контролирует движение поездов. Может рассчитывать маршрут и положение стрелок по ходу движения поезда по маршруту. Поддерживает функции ускорения и торможения.
Train Couple	Сцепляет два поезда, которые «касаются» друг друга, в один.
Train Decouple	Расцепляет вагоны поступающего в блок поезда и создает из них новый поезд.
Train Enter	Помещает поступающего в блок агента-поезд на заданный путь ж/д сети.
Train Exit	Извлекает поступающий в объект поезд из ж/д сети и передает агента-поезд далее в обычную диаграмму процесса.
Rail Settings	Задаёт специфические настройки для железнодорожной сети.



Сортировочная горка. Фаза 1. Шаг 7

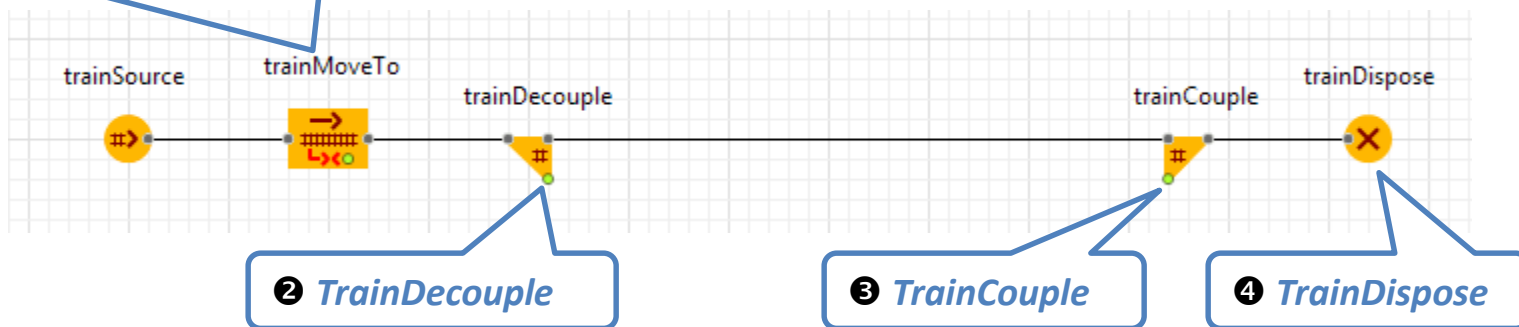
❶ *TrainMoveTo*

Маршрут: Вычисляется автоматически от текущего пути до пути назначения

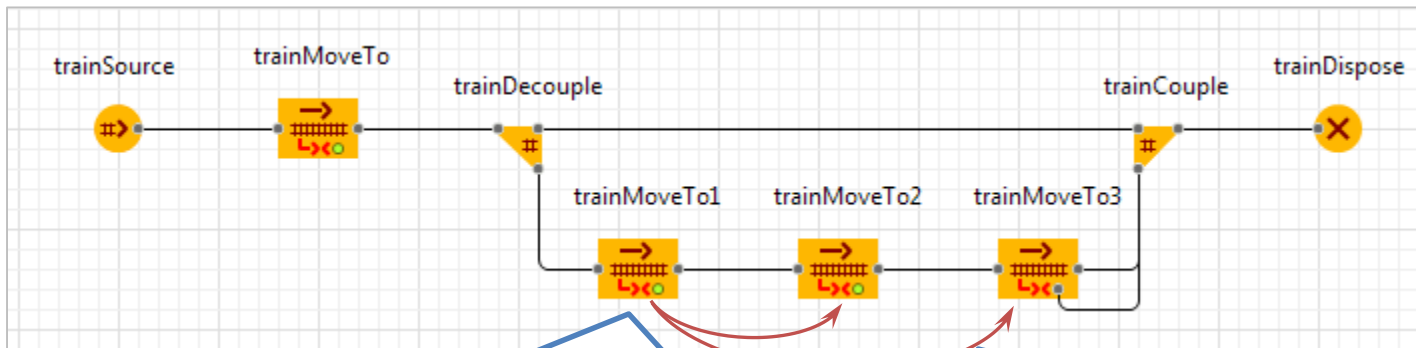
Цель движения: Заданная точка пути

Точка ж/д пути: *stopLineArrival*

При окончании движения: Затормозить и остановиться



Сортировочная горка. Фаза 1. Шаг 8



❶ *TrainMoveTo*

Маршрут: Вычисляется автоматически от текущего пути до пути назначения

Цель движения: Заданная точка пути

Точка ж/д пути: *stopLineHump*

При окончании движения: Затормозить и остановиться

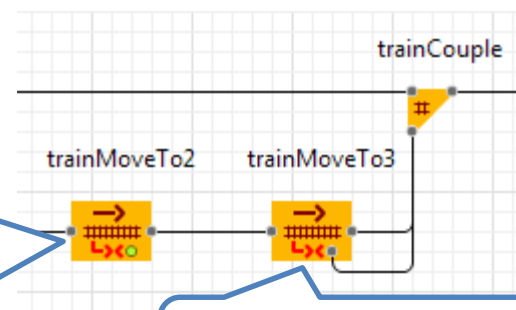
❷ Ctrl+тащите (Mac OS: Cmd+тащите) блок *TrainMoveTo*, чтобы создать еще два таких же блока

❸

Направление движения: Назад

**Маршрут не должен содержать {...}:
*trackArrival***

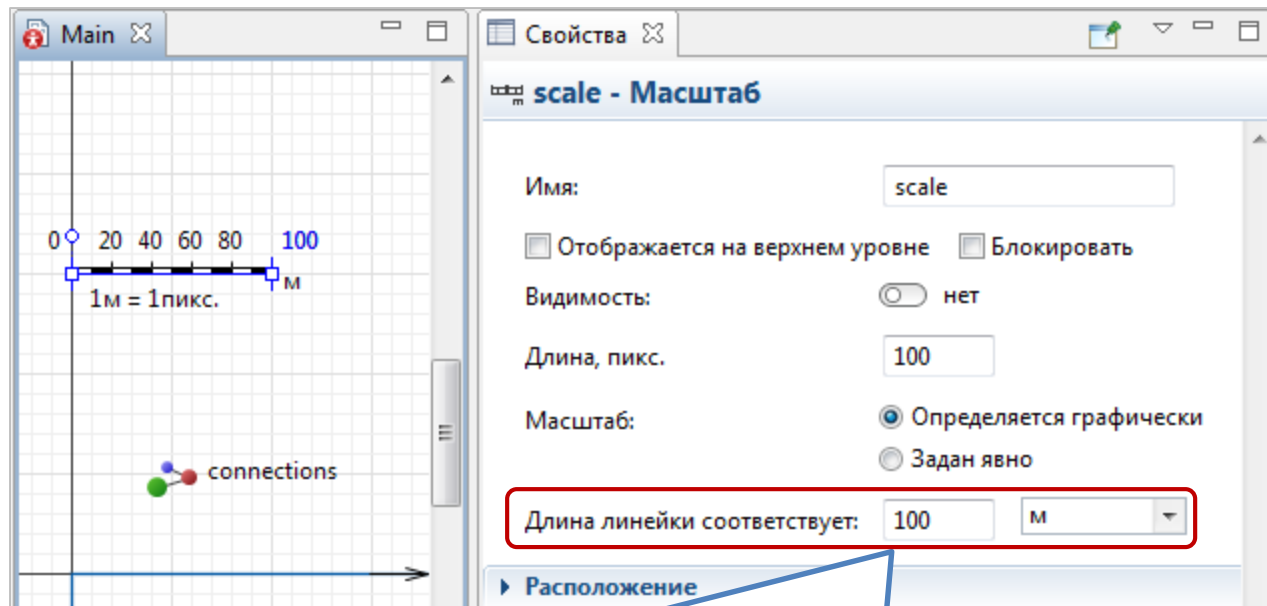
Точка ж/д пути: *stopLineEntry*



❹ **Точка ж/д пути:**
stopLineArrival



Сортировочная горка. Фаза 1. Шаг 9



❶ Измените масштаб диаграммы *Main*. Элемент **Масштаб** находится на диаграмме над осью X. Вагоны в нашей модели имеют длину 14 метров, поэтому мы увеличим масштаб ж/д путей до *1 метра в 1 пикселе*, чтобы они помещались на путь. Все созданные нами типы вагонов имеют такой же масштаб на своей диаграмме



Сортировочная горка. Фаза 1. Шаг 10

1 Запустите модель. Вы увидите, как поезд появляется и движется по нашему пути.

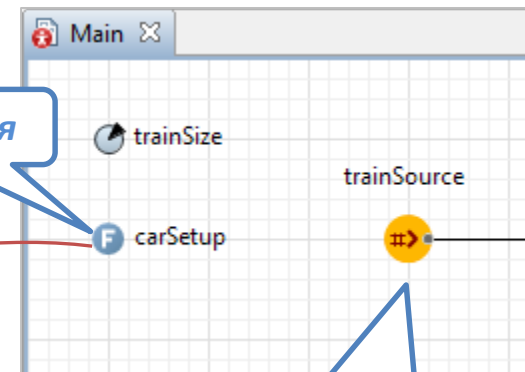
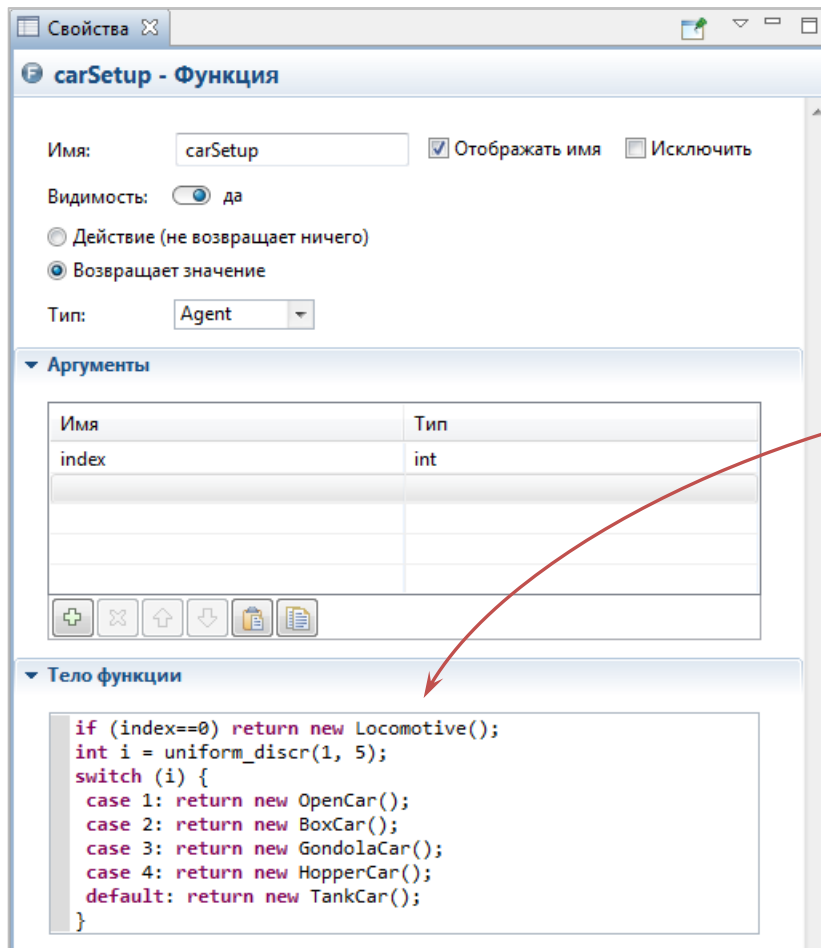
2 Поезд останавливается в точке остановки и локомотив использует стрелочную горловину, чтобы присоединиться к концу поезда.

Прогон: 0 Пауза | Время: 15.89 | Прогон: Время остановки не задано | Дата: 28.10.2014

Прогон: 0 Пауза | Время: 126.70 | Прогон: Время остановки не задано | Дата: 28.10.2014 14:51:41 | Память: 132M из 2048M



Сортировочная горка. Фаза 1. Шаг 11



❶ Функция

❷ Новый вагон:
carSetup(carindex)



- ❶ Нам нужно разнообразить вагоны поезда. Функция *carSetup()* устанавливает *Locomotive* как тип первого вагона в поезде и затем случайным образом выбирает типы остальных вагонов в поезде.

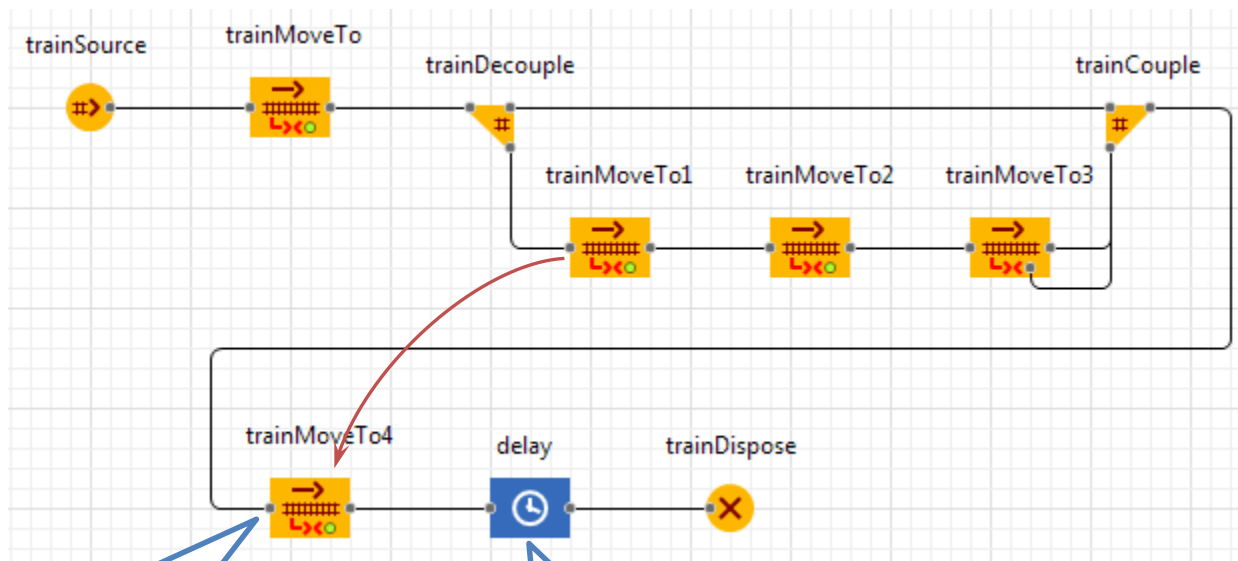
Оператор *switch*

- Оператор *switch* позволяет выбирать, какую из произвольного количества секций кода нужно выполнить, исходя из того, какому значению равно заданное выражение. В нашем примере оператор *return* используется вместо *break*, потому что мы находимся в теле функции.

Syntax	Example
<pre>switch(<integer expression>) { case <integer constant 1>: <statements to be executed in case 1> break; case <integer constant 2>: <statements to be executed in case 2> break; ... default: <statements to be executed if no cases apply> break; }</pre>	<pre>int month; String season; switch(month) { case 1: season = "winter"; break; case 2: season = "winter"; break; case 3: season = "spring"; break; ... case 11: season = "autumn"; break; case 12: season = "winter"; break; default: season = "invalid value"; break; }</pre>



Сортировочная горка. Фаза 1. Шаг 12



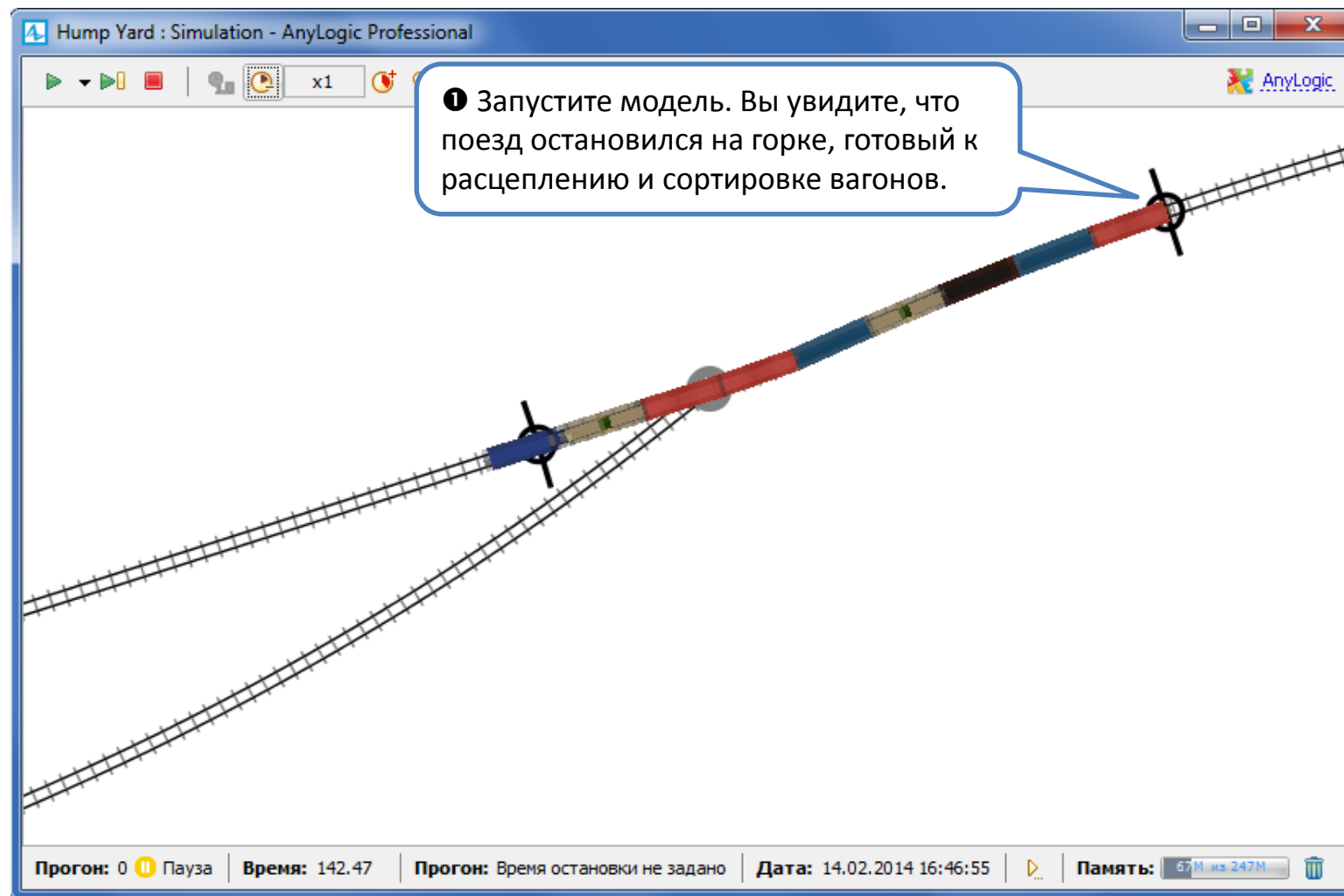
❶ Еще раз Ctrl+перетащите блок *TrainMoveTo*, чтобы создать новый блок. Убедитесь, что Точка ж/д пути: *stopLineHump*

❷ *Delay* (блок Библиотеки моделирования процессов)
Время задержки: 15 секунд

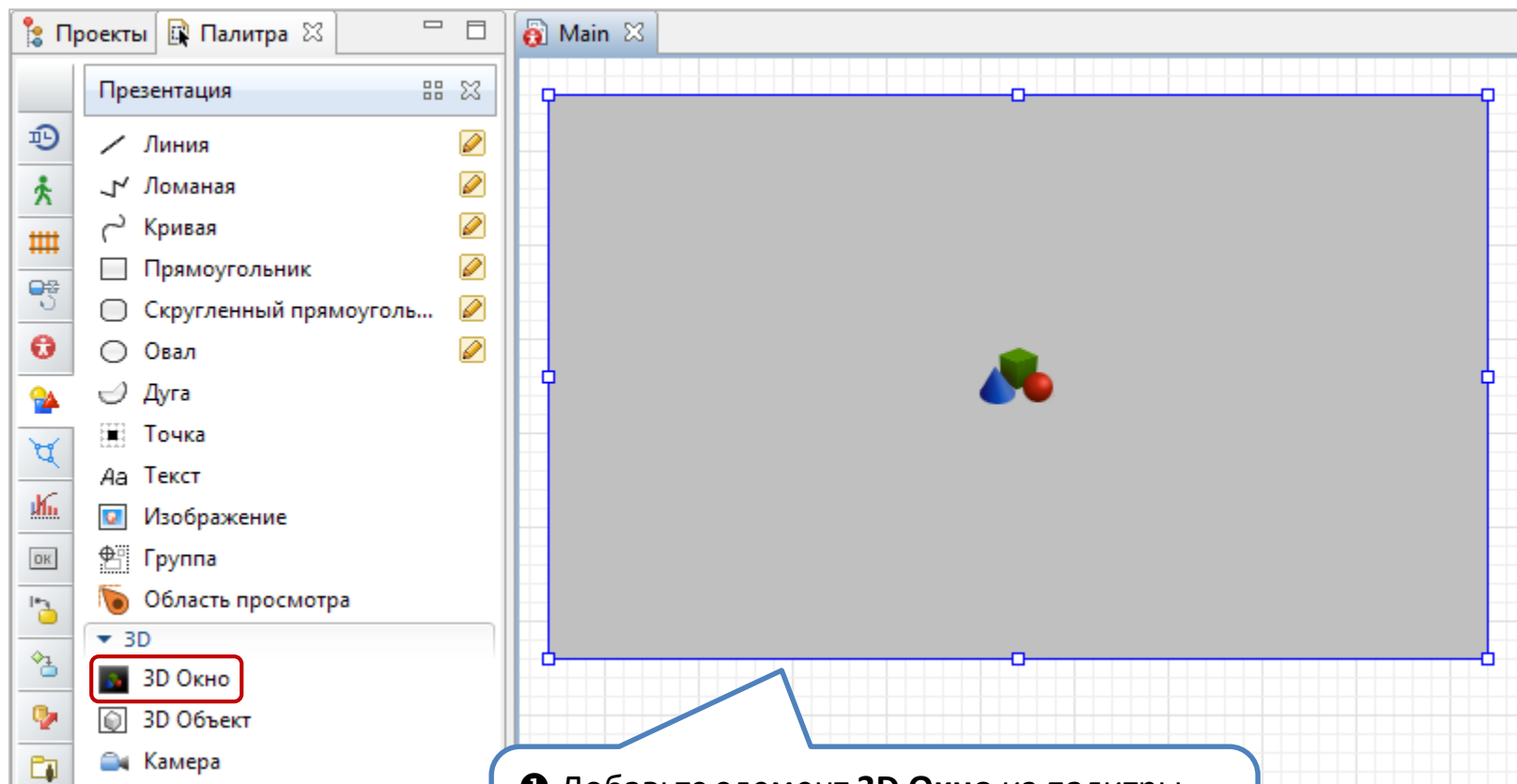




Сортировочная горка. Фаза 1. Шаг 13



Сортировочная горка. Фаза 1. Шаг 14



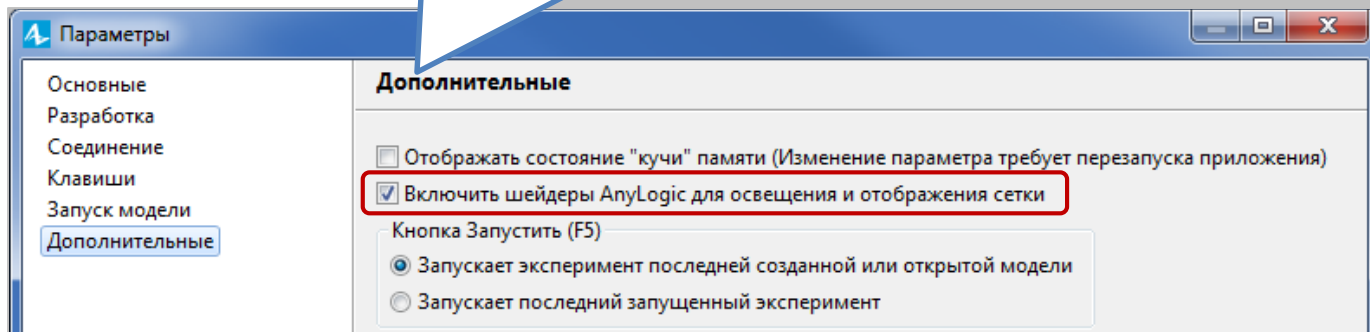
❶ Добавьте элемент **3D Окно** из палитры **Презентация**, расположите его под Вашей диаграммой и фигурами анимации



Сортировочная горка. Фаза 1. Шаг 15

❶ Запустите модель и щелкните
Показать область... [window3d]

❷ Если у Вас имеются проблемы с
визуализацией текстур 3D объектов,
Вы можете выключить шейдеры в
настройках AnyLogic



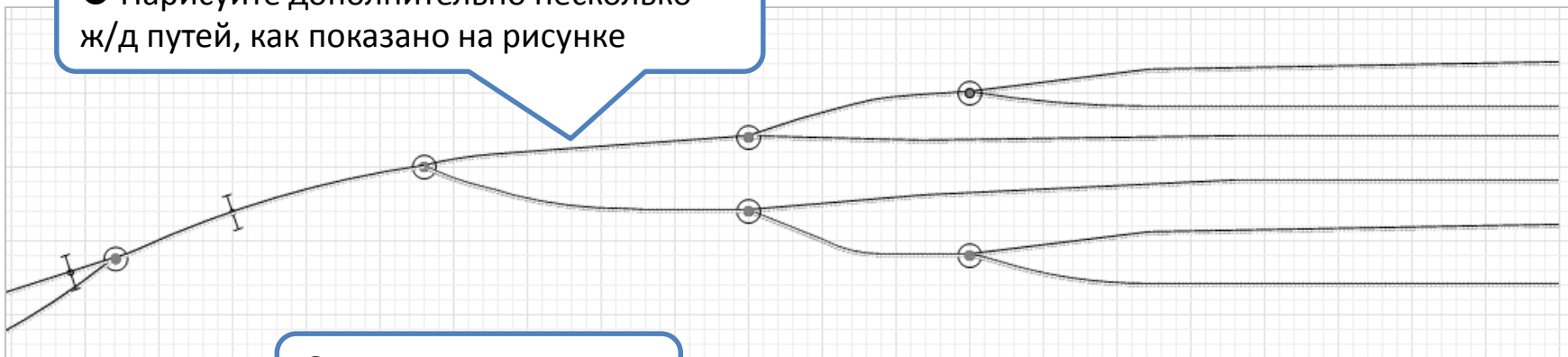
Сортировочная горка. Фаза 2

- Теперь давайте смоделируем процесс сортировки вагонов и их распределения на разные пути согласно их типу.
- Для этого нужно нарисовать больше путей и добавить дополнительные блоки на диаграмму процесса.

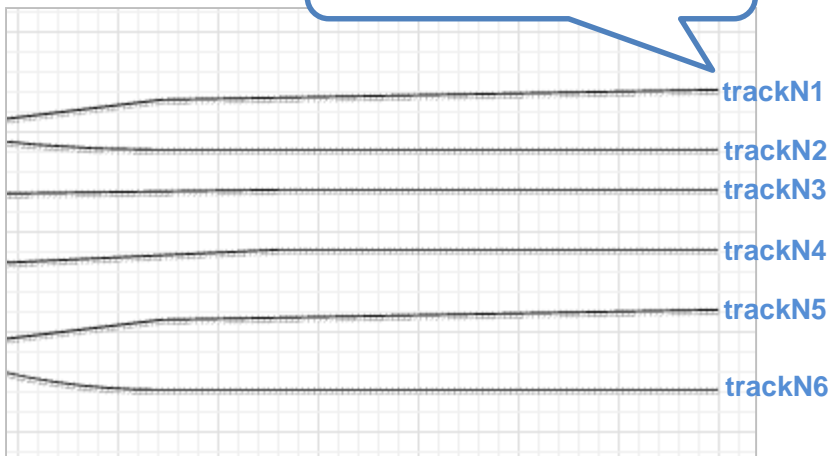


Сортировочная горка. Фаза 2. Шаг 1

❶ Нарисуйте дополнительно несколько ж/д путей, как показано на рисунке



❷ Назовите пути следующим образом:



stopLineN1

stopLineN6

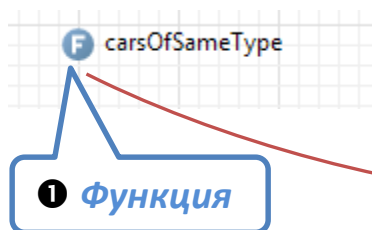
❸ Добавьте шесть элементов **Точка ж/д пути** и назовите их *stopLineN1 .. stopLineN6*



- ❶ Нарисуйте пути назначения.
- ❷ Дайте путям «говорящие» имена.
- ❸ Добавьте элемент **Точка ж/д пути** на каждый путь назначения. Таким образом мы задаем линию остановки для каждого пути. Вагоны, который катятся вниз под действием силы земного притяжения, будут останавливаться на этих линиях.



Сортировочная горка. Фаза 2. Шаг 2



Свойства

carsOfSameType - Функция

Имя: ☒ Отображать имя ☐ Исклчить

Видимость: ☒ да

☐ Действие (не возвращает ничего)

☒ Возвращает значение

Тип:

▼ Аргументы

Имя	Тип
train	Train

▼ Тело функции

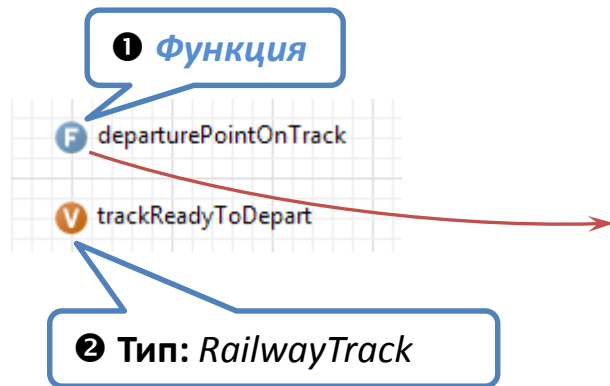
```
Agent firstCar = train.getFirst();
int n = 1;
while( train.getCar(n).getClass() == firstCar.getClass() )
    n++;
return n;
```



- ❶ Поезд, прибывающий на сортировочную станцию может содержать несколько вагонов одного и того же типа подряд. Такие вагоны могут быть отправлены на свой путь все вместе. Эта функция вычисляет количество вагонов одного типа, следующих друг за другом, в поезде, находящемся на горке.



Сортировочная горка. Фаза 2. Шаг 3



Свойства

departurePointOnTrack - Функция

Имя: ☒ Отображать имя

☐ Исключить

Видимость: ☒ да

☐ Действие (не возвращает ничего)

☒ Возвращает значение

Тип:

Аргументы

Имя	Тип
train	Train

Тело функции

```
Agent car = train.getCar(0);
if( car instanceof HopperCar ) return stopLineN1;
if( car instanceof BoxCar ) return stopLineN2;
if( car instanceof OpenCar ) return stopLineN3;
if( car instanceof GondolaCar ) return stopLineN4;
if( car instanceof TankCar ) return stopLineN5;
return null;
```



- ❶ Давайте зададим функцию, которая определяет путь назначения для вагонов в сортировочном парке (пути, где сортируются вагоны определенного типа)

Эта функция использует поезд как аргумент, затем проверяет тип первого вагона поезда (так как обычно поезд и состоит всего лишь из одного вагона).

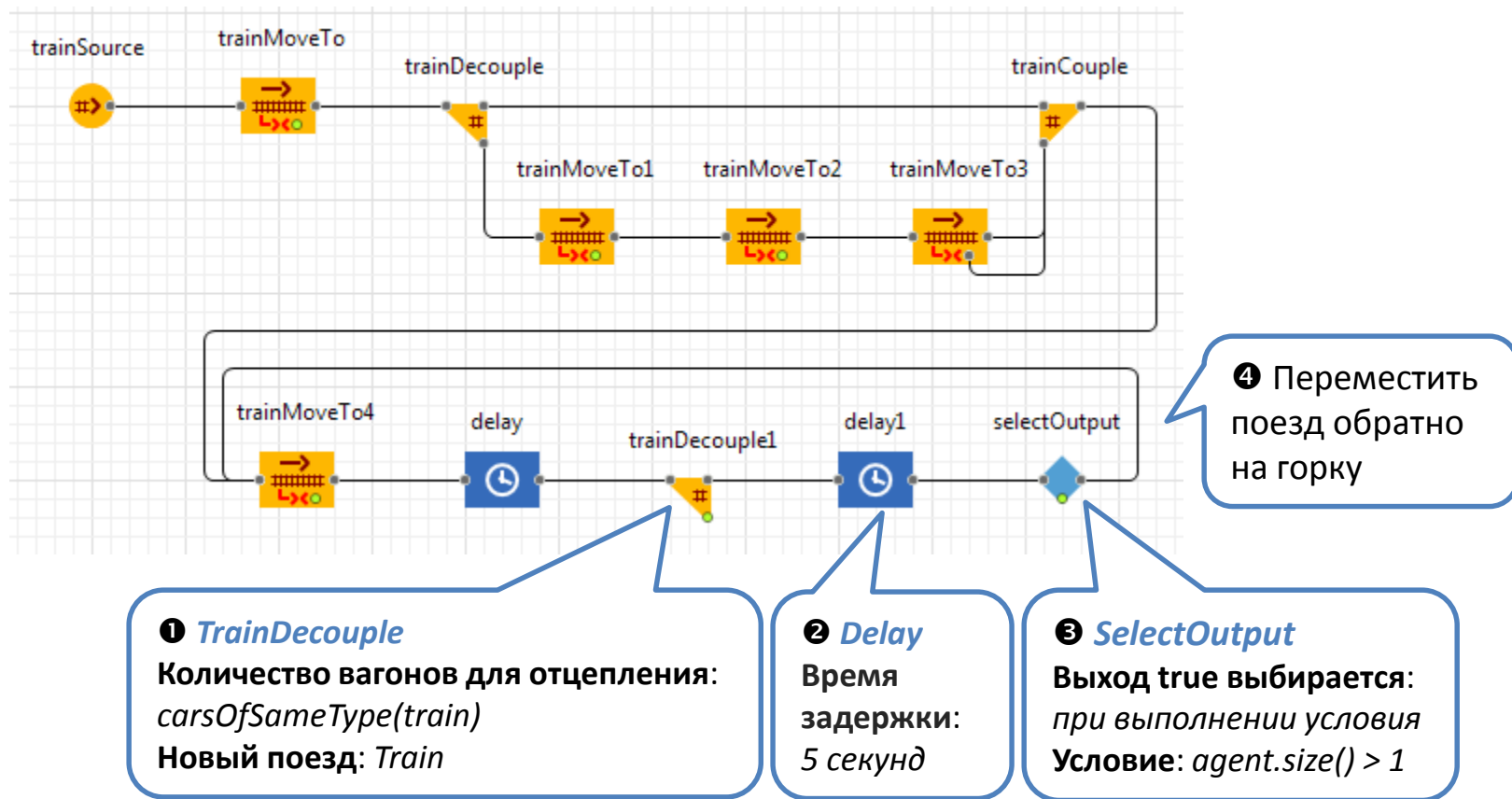
Мы находим тип вагона с помощью оператора Java *instanceof*. Он возвращает *true*, если объект является единицей заданного класса.

В итоге, функция возвращает элемент **Точка ж/д пути**, который зависит от типа вагона.

- ❷ Эта переменная содержит ссылку на путь, где требуемое количество вагонов одного типа готовы покинуть станцию как новый поезд.



Сортировочная горка. Фаза 2. Шаг 4

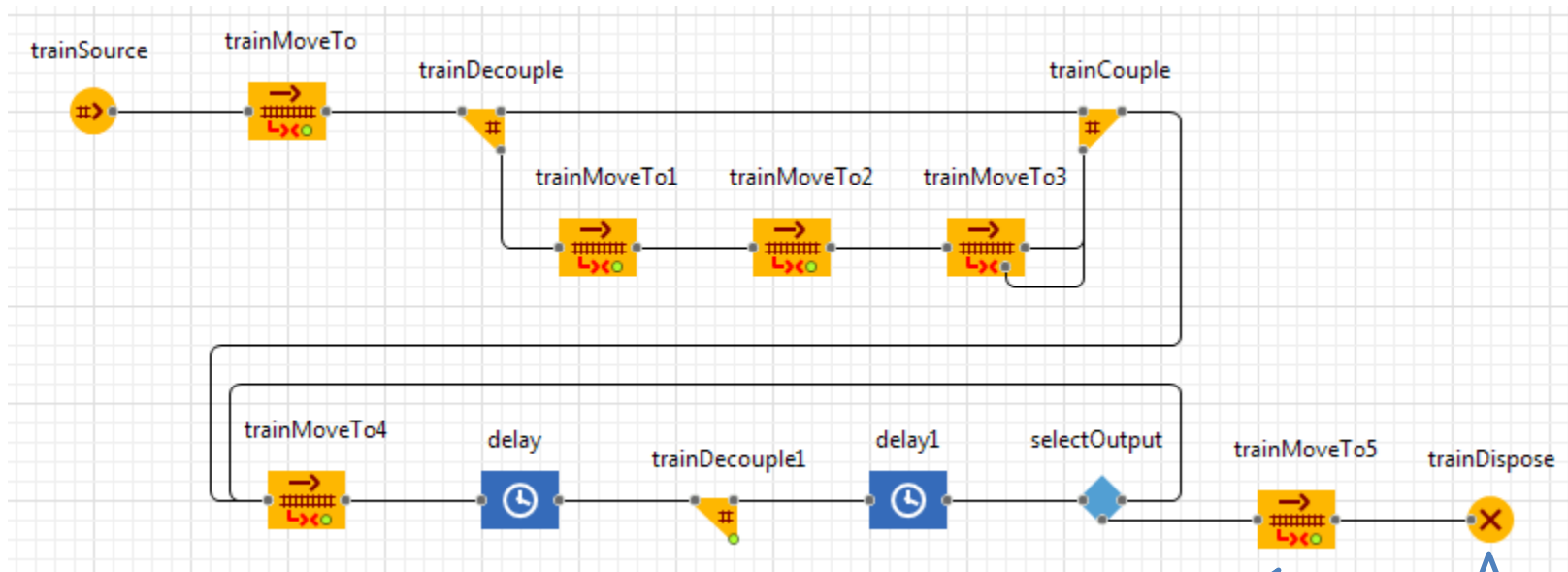


Теперь давайте добавим новую логику процесса на нашу диаграмму.

- ❶ Блок **TrainDecouple** расцепляет вагоны, когда поезд находится на горке. Поезд может содержать несколько вагонов одного типа подряд. Такие вагоны откатываются на свой путь назначения сразу же. (Мы находим количество вагонов, которые нужно откатить, функцией *carsOfSameType*, созданной на предыдущем шаге). Путь назначения задается созданной нами функцией *departurePointOnTrack*.
- ❷ Имеется короткая задержка перед тем, как поезд продолжит расцепляться.
- ❸ Теперь мы должны определить, содержит ли поезд еще вагоны, которые можно отцепить. Этим занимается блок **SelectOutput**. Мы проверяем, содержит ли поезд еще хоть один вагон (помните, что и сам локомотив считается вагоном). Если это так, то поезд снова поступает в блок **TrainDecouple**. И так повторяется до тех пор, пока поезд не содержит только локомотив, готовый покинуть станцию.



Сортировочная горка. Фаза 2. Шаг 5



❶ *TrainMoveTo*

Маршрут: Вычисляется автоматически...

Цель движения: Заданное смещение на пути

Путь: trackN6

Крейсерская скорость: 5 м/с

❷ *TrainDispose*

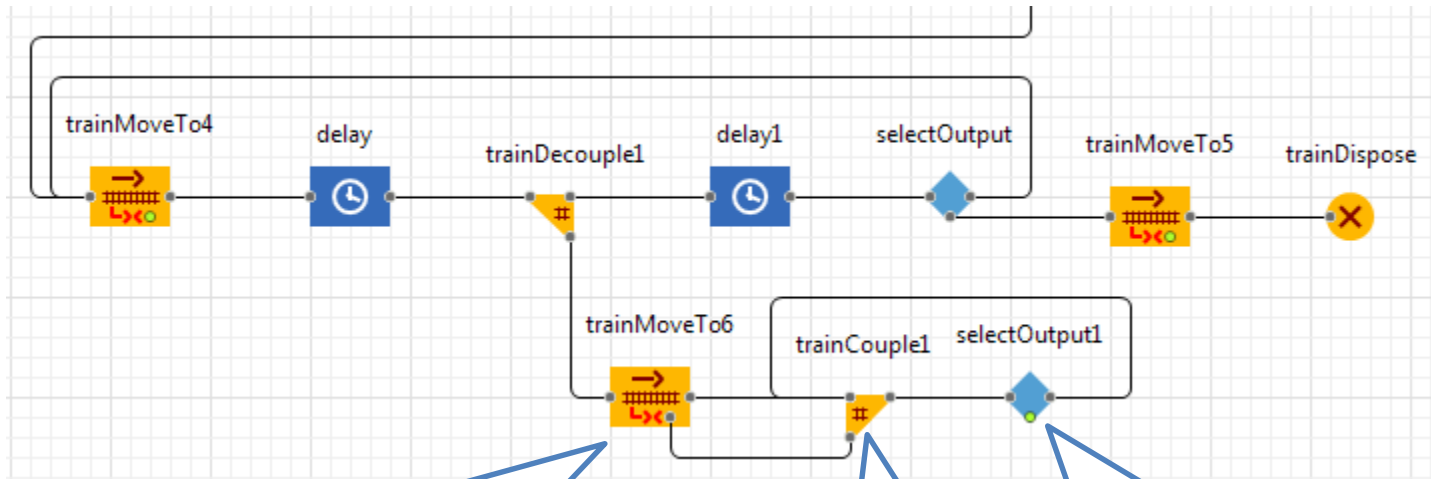


Теперь давайте смоделируем то, как локомотив покидает станцию.

- ① Мы снова используем блок *TrainMoveTo*, чтобы имитировать движение поезда. Как назначение мы указываем имя самого нижнего пути (*trackN6*).
- ② В конце концов, локомотив удаляется из модели блоком *TrainDispose*.



Сортировочная горка. Фаза 2. Шаг 6



❶ *TrainMoveTo*

Маршрут: Вычисляется автоматически...
Цель движения: Заданная точка пути
Точка ж/д пути: `departurePointOnTrack(train)`
Крейсерская скорость: 5 м/с
При окончании движения: Затормозит и остановится

❷ *TrainCouple*

❸ *SelectOutput*

Выход true выбирается:
При выполнении условия
Условие:
 $\text{agent.size()} < \text{trainSize}-1$

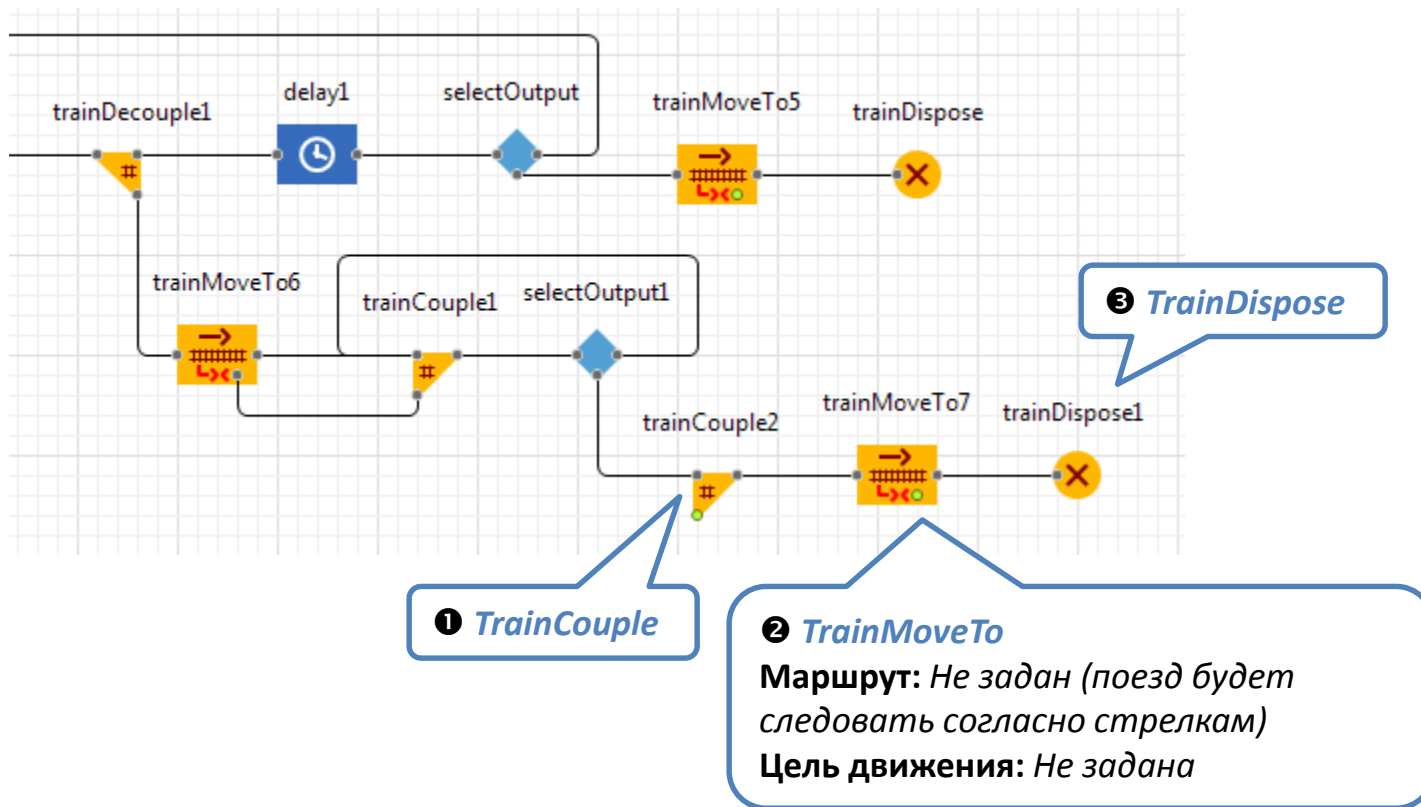


Теперь мы хотим имитировать то, как вагоны откатываются под действием силы земного притяжения на свои пути назначения в сортировочном парке (пути, где вагоны сортируются).

- ❶ Движение, как обычно, моделируется блоком *TrainMoveTo*. Поезд содержит отдельные вагоны или сцепленные вагоны одного типа. Назначение задается функцией *departurePointOnTrack*, которую мы указали ранее.
- ❷ Если на пути назначения уже есть какие-то вагоны, то они комбинируются вместе с только что прибывшими вагонами.
- ❸ Вагоны продолжают соединяться, пока не будет достигнуто определенное количество вагонов на пути (тогда локомотив приезжает туда и забирает вагоны со станции). Блок *SelectOutput* проверяет, когда сбор вагонов можно прекратить.



Сортировочная горка. Фаза 2. Шаг 7

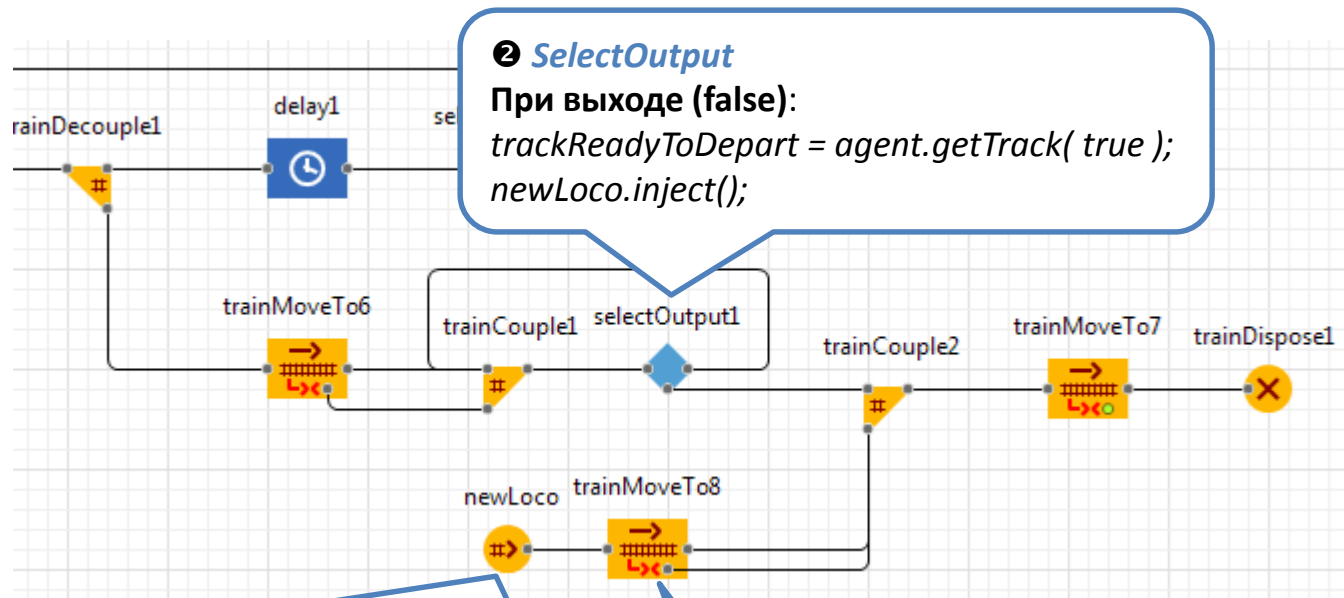


Теперь давайте добавим ответвление диаграммы, моделирующее то, как готовый поезд, собранный из вагонов одного типа, покидает станцию.

- ❶ Сначала мы сцепляем вагоны с локомотивом (логику поведения локомотива мы зададим на следующем шаге).
- ❷ Затем поезд покидает станцию (это имитируется блоком *TrainMoveTo*).
- ❸ В конце концов мы удаляем поезд из модели блоком *TrainDispose*.



Сортировочная горка. Фаза 2. Шаг 8



② *SelectOutput*

При выходе (false):

```
trackReadyToDepart = agent.getTrack( true );  
newLoco.inject();
```

① *TrainSource*

Имя: *newLoco*

Поезда прибывают согласно: Вызовом функции *inject()*

Кол-во вагонов (включая локомотив): 1

Точка входа задается как: Смещение на пути

Путь: *trackReadyToDepart*

Смещение: от начала пути

Смещение первого вагона: *tracklength - 15 м*

Новый поезд: *Train*

③ *TrainMoveTo*

Направление движения: Назад

Маршрут: Не задан (поезд будет следовать согласно стрелкам)

Цель движения: Не задана

Крейсерская скорость: 10 м/с

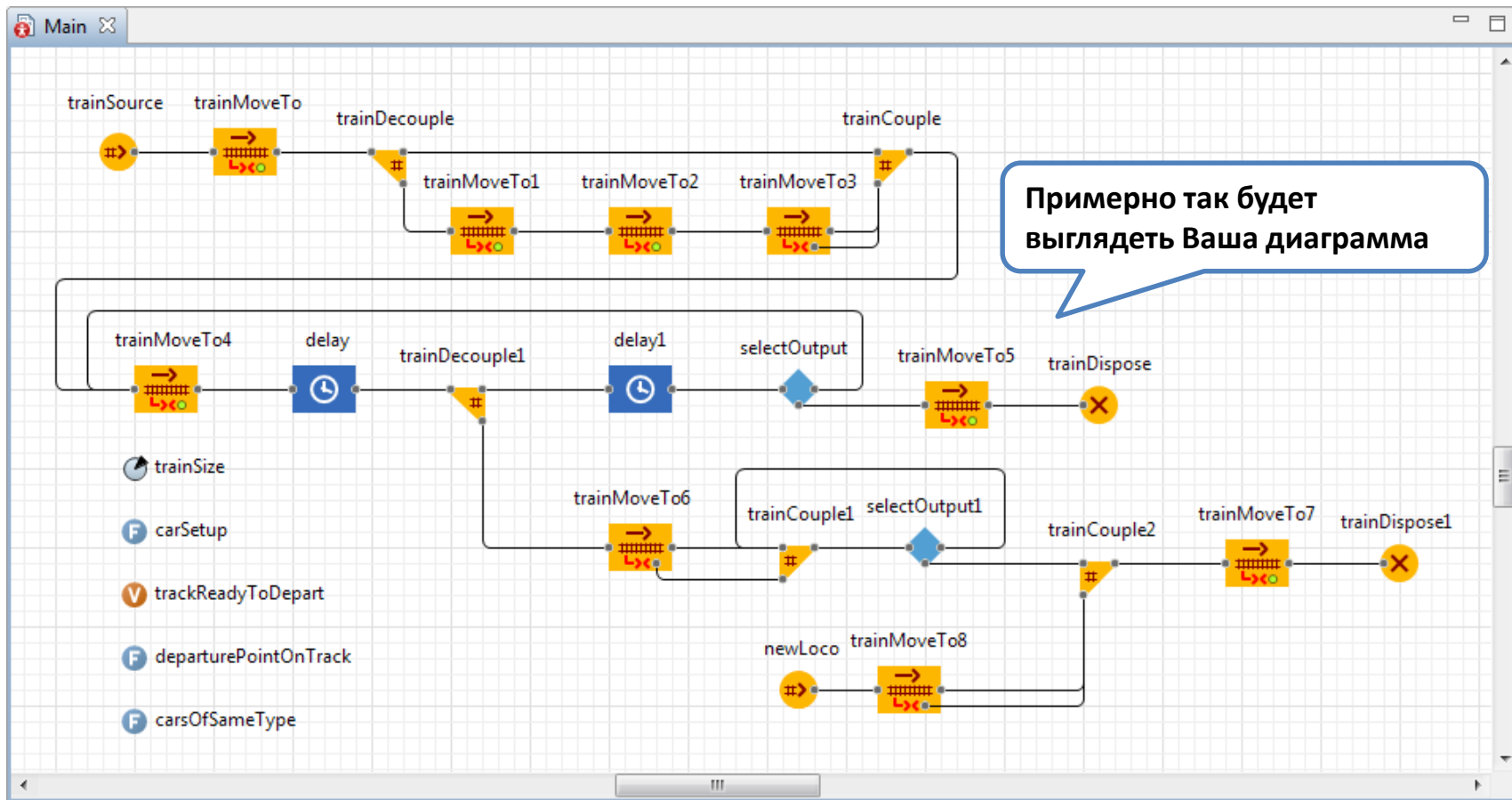


Теперь мы хотим смоделировать то, как локомотивы забирают со станции готовые поезда, составленные из вагонов одного типа.

- ❶ Сначала давайте добавим блок **TrainSource** на модель прибытия локомотива. Поезд состоит только из одного вагона (сам локомотив). Выберите ручной режим создания поездов – вызовами функции **inject()**, так как этот блок будет создавать поезда, только когда новый поезд готов отправиться со станции. Локомотив должен появиться на пути, заданном переменной **trackReadyToDepart** (мы вычисляем значение переменной на следующем шаге).
- ❷ Этот блок проверяет, имеется ли достаточное количество вагонов одного типа на пути или еще нет. Задайте действие **При выходе (false)**. Оно выполняется, когда появляется требуемое количество вагонов.
Линия **trackReadyToDepart = agent.getTrack(true);** помещает текущий путь в переменную **trackReadyToDepart**.
Линия **newLoco.inject();** генерирует новый локомотив. Когда локомотив создан, он следует к блоку, который мы создаем далее.
- ❸ Блок **TrainMoveTo** моделирует то, как локомотив движется к вагонам, готовым к отправке. Пожалуйста, обратите внимание, что его направление движения – **Назад** (локомотив движется в направлении, обратном направлению пути, справа налево). Затем локомотив сцепляется с вагонами и получившийся поезд проходит ту часть диаграммы процесса, которая моделирует отправление поезда с сортировочной горки.



Сортировочная горка. Фаза 2. Шаг 9



Сортировочная горка. Фаза 2. Шаг 10

