

Лабораторная работа

Модель сортировочной горки (транспортной системы)

Цель работы: закрепить навыки работы со средой Any Logic и развить навыки применения Железнодорожной библиотеки AnyLogic для создания модели сортировочной горки.

Постановка задачи.

Сортировочная горка – это вид сортировочной станции, использующий для перемещения вагонов земное тяготение, то есть скатывание вагонов и групп вагонов с уклона.

Прибывающий товарный поезд содержит 5 разных типов вагонов, которые отправляются на 5 путей назначения. Как только на пути накапливается 8 вагонов одного типа, из них формируется новый состав, который покидает этот путь.

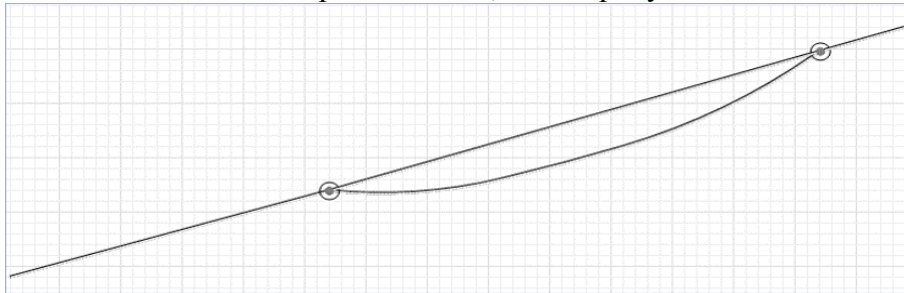
Содержание работы.

1. Создание железнодорожного узла
2. Задание логики процесса
3. Создание типов вагонов
4. Моделирование сортировки вагонов

Шаг 1. Создание железнодорожного узла

С помощью модели воспроизведем отправку вагонов на путь, который называют **стрелочной горловиной**. Оттуда вагоны отправляются через серию стрелок, называемую **стрелочной улицей**, на пути сортировки.

Начнем с топологии железнодорожной сети, как на рисунке ниже:




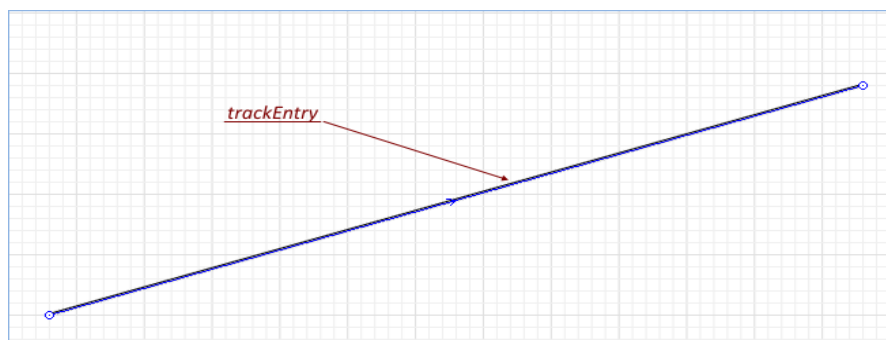
Создайте новую модель. Назовите ее *Hump Yard* и укажите *минуты* в качестве единиц модельного времени.

Задайте топологию путей. Сначала нарисуйте ж/д пути.

Начните с прямого пути. На этом пути будут появляться поезда, поэтому мы назовем его *trackEntry*. Будем давать осмысленные имена только тем путям, на которые будут ссылаться блоки диаграммы процесса. Остальные пути могут остаться названными по умолчанию.

Нарисуйте прямой ж/д путь


1. Сначала выделите двойным щелчком элемент **Ж/д путь**  в секции **Разметка пространства** палитры **Железнодорожная библиотека**.
2. Щелкните в любой точке графического редактора, чтобы начать рисовать путь.
3. Чтобы нарисовать прямой сегмент пути, добавьте щелчком мыши конечную точку сегмента.
4. Завершите рисование двойным щелчком.
5. Назовите этот ж/д путь *trackEntry*.

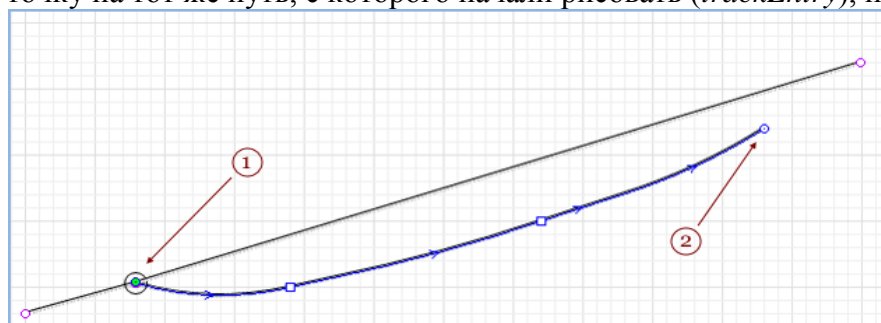


Если это первый ж/д путь в модели, вы увидите сообщение с предложением сменить масштаб модели на: 2,0 пикселя в 1 метре.

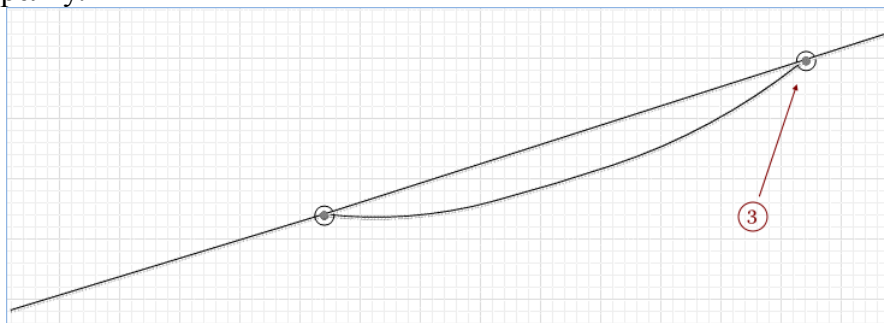
После этого вы увидите нарисованный прямой ж/д путь, по которому будет двигаться состав. Теперь давайте нарисуем обьездной путь, по которому локомотив будет подъезжать к составу сзади для его перемещения на сортировочную горку. Этот путь будет не прямой, а дуговой формы, и сценарий его рисования будет несколько отличаться.

Нарисуйте ж/д путь дуговой формы

1. Совершите двойной щелчок по элементу **Ж/д путь**  в палитре **Железнодорожная библиотека**. Затем щелкните по ранее нарисованному пути (см. рисунок ниже, пункт 1), чтобы начать рисовать новый ж/д путь. Круг, обозначающий стрелку, появится автоматически.
2. Последовательно щелкая мышью, нарисуйте несколько сегментов пути, чтобы получился путь, как на рисунке ниже. Чтобы добавить дуговой элемент, не отпускайте кнопку мыши после щелчка, а переместите курсор, удерживая кнопку мыши. При этом вы увидите, как меняется радиус дуги. Отпустите кнопку мыши, когда посчитаете, что сегмент приобрел нужную форму. Завершите рисование ж/д пути двойным щелчком рядом с фигурой первого пути (см. пункт 2 на рис. ниже). Сразу поставить конечную точку на тот же путь, с которого начали рисовать (*trackEntry*), не получится.



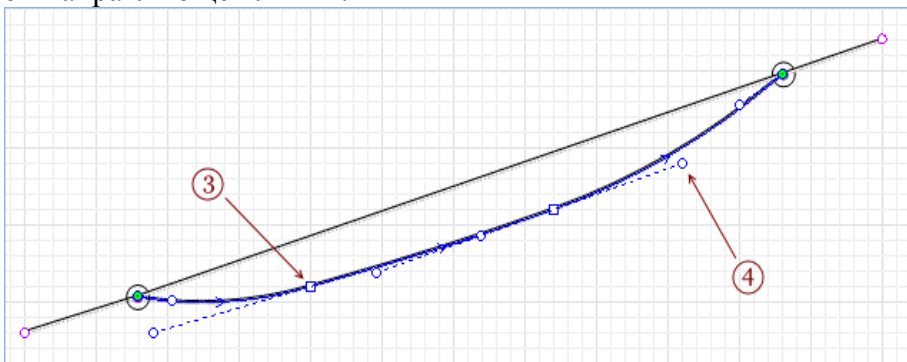
3. Перетащите конечную точку (2) на ранее нарисованный путь (*trackEntry*), чтобы создать стрелку.



Отредактируйте ж/д путь дуговой формы

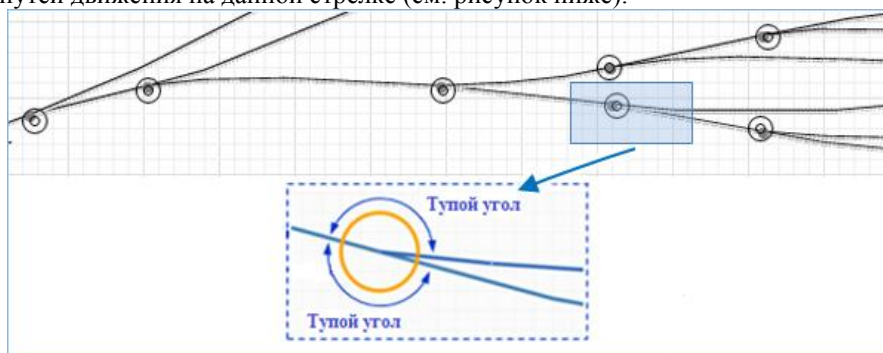
1. Выделите путь, который хотите отредактировать, щелкнув по нему.
2. Щелкните правой кнопкой мыши по выделенному пути и выберите **Редактировать направляющие** из контекстного меню.
3. Чтобы переместить крайнюю точку сегмента, перемещайте квадратную метку-манипулятор.

4. Чтобы изменить радиус кривизны, перемещайте круглую метку-манипулятор на конце пунктирной направляющей линии.



Железнодорожные стрелки

- Железнодорожная сеть состоит из путей и стрелок. Железнодорожные стрелки появляются автоматически в виде круга в точках соединения ж/д веток с существующими путями. В существующую точку соединения нельзя добавить дополнительные ж/д пути.
- Два из трех образованных стрелкой углов должны быть тупыми. Это необходимо для определения возможных путей движения на данной стрелке (см. рисунок ниже).

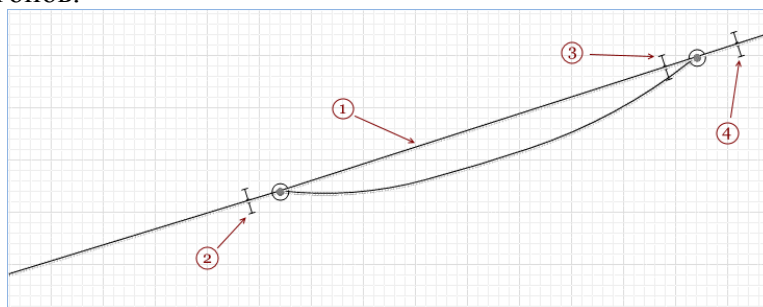


Модифицируйте железнодорожный узел

Каждый раз при создании железнодорожных стрелок, ж/д путь разбивается на отдельные пути. В нашем случае мы создали две железнодорожные стрелки, соединив обьездной путь с существующим путем в двух точках. В результате существующий путь разделился на три пути, каждый со своим именем и свойствами.

Сейчас мы дадим осмысленное имя сегменту **ж/д пути** между железнодорожными стрелками и добавим элемент **Точка ж/д пути** в наш железнодорожный узел.

- Назовите этот путь *trackArrival*, так как он обозначает место прибытия поезда.
- Перетащите элемент **Точка ж/д пути** из палитры **Железнодорожная библиотека** и расположите его, как показано на рисунке ниже. Назовите его *stopLineEntry*. Он будет определять место появления поездов.
- Добавьте элемент **Точка ж/д пути: stopLineArrival**. В этой точке поезд остановится, чтобы отцепить все вагоны, что позволит локомотиву продолжить путь уже без вагонов.
- Добавьте элемент **Точка ж/д пути: stopLineHump**. Здесь будет производиться сортировка вагонов.



Точка ж/д пути

Точка ж/д пути является элементом разметки пространства, который используется, 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.

В этой части учебного пособия мы создали железнодорожный узел, который будет далее использоваться в нашей модели.

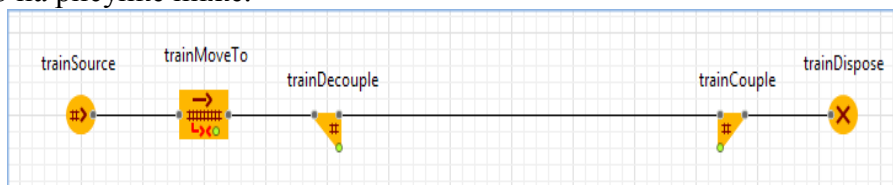
Обратите внимание, для каждого шага моделирования имеются контрольные модели. Вы можете использовать их, чтобы сравнить свою модель с аналогичной моделью, созданной нами. Далее приведена ссылка на контрольную модель для этого шага.

Шаг 2. Задаем логику

Задайте логику процесса с помощью блоков *Железнодорожной библиотеки*

1. Добавьте следующие блоки на диаграмму процесса: [TrainSource](#), [TrainMoveTo](#), [TrainDecouple](#), [TrainCouple](#), [TrainDispose](#). С помощью этих блоков мы зададим логику, согласно которой будет передвигаться наш поезд.

Начните диаграмму с блока **TrainSource**, который создает поезда, затем добавьте блок **TrainMoveTo**, расположив его рядом с первым блоком таким образом, чтобы они могли автоматически соединиться. Добавьте оставшиеся блоки на диаграмму процесса, как показано на рисунке ниже.



В свойствах блоков **TrainSource** и **trainMoveTo** укажите следующее:

Сначала мы укажем, чтобы поезда появлялись у ж/д точки *stopLineEntry* каждые 15 минут.

TrainSource:

Время между прибытиями: 15 минут

Кол-во вагонов (включая локомотив): 11

Точка ж/д пути: *stopLineEntry*

После создания, каждый поезд будет направляться к ж/д точке *stopLineArrival*, у которой он будет замедляться, после чего окончательно остановится.

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*

2. Мы также добавили блок **TrainDecouple**, задача которого отцеплять заданное количество вагонов от поступающего поезда. В свойствах блока можно увидеть, что по умолчанию он отцепляет один вагон от поезда, а именно первый (в нашем случае - это локомотив). Локомотив будет покидать блок через нижний порт *outDecoupled* (мы добавим это соединение на следующем шаге).

3. Блок **TrainCouple** нужен для сцепления локомотива и вагонов в один состав.

Ниже приведены описания блоков *Железнодорожной библиотеки*, которые мы используем.

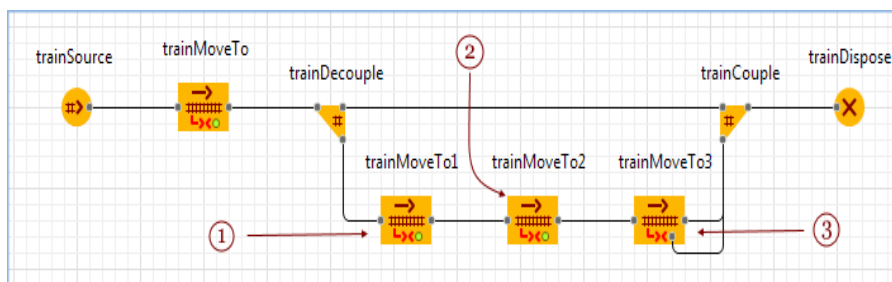
Блок Железнодорожной Библиотеки	Описание
Train Source	Создает поезда, выполняет начальную настройку и помещает их в железнодорожную сеть. Начинает любую ж/д диаграмму процесса. Поддерживает несколько типов расписаний прибытия.
Train Dispose	Удаляет поезда из модели.
Train Move To	Моделирует движение поездов. Может рассчитывать маршрут и положение стрелок по ходу движения поезда по маршруту. Поддерживает функции ускорения и торможения.

Train Couple	Сцепляет два поезда, которые $i_c/2$ касаются $i_c/2$ 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.
RailSettings	Задаёт специфические настройки для железнодорожной сети.

Продолжим рисование диаграммы процесса

Добавьте три блока **TrainMoveTo** и соедините их, как показано на рисунке, следуя описанным ниже шагам. С помощью этих блоков мы зададим поведение поезда на железнодорожном узле, которое будет включать в себя: отцепление, ход назад, сцепление с вагонами с последующим выталкиванием их на горку.

Начните с присоединения блока **TrainMoveTo** к порту *outDecoupled* (который находится внизу) блока **TrainDecouple**, так как мы будем задавать логику для отсоединенного локомотива.



1. Добавьте еще один блок **TrainMoveTo** на диаграмму процесса (*trainMoveTo1*). Задайте параметры нового блока, согласно которым локомотив направится к *stopLineHump* и остановится там.

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*

Создайте еще два блока **TrainMoveTo** (*trainMoveTo2*, *trainMoveTo3*) путем Ctrl+перетаскивания (Mac OS: Cmd+перетаскивания) блока, который вы только что редактировали.

2. Измените свойства блока *trainMoveTo2*. В данный момент локомотив отцеплен и нам нужно, чтобы он затолкал вагоны на горку, т.е. ему нужно вернуться к *stopLineEntry* и произвести сцепку с концом поезда. Так как мы не можем его развернуть на рельсах, локомотив поедет назад мимо пути *trackArrival*, на котором стоят вагоны:

Направление движения: *Назад*

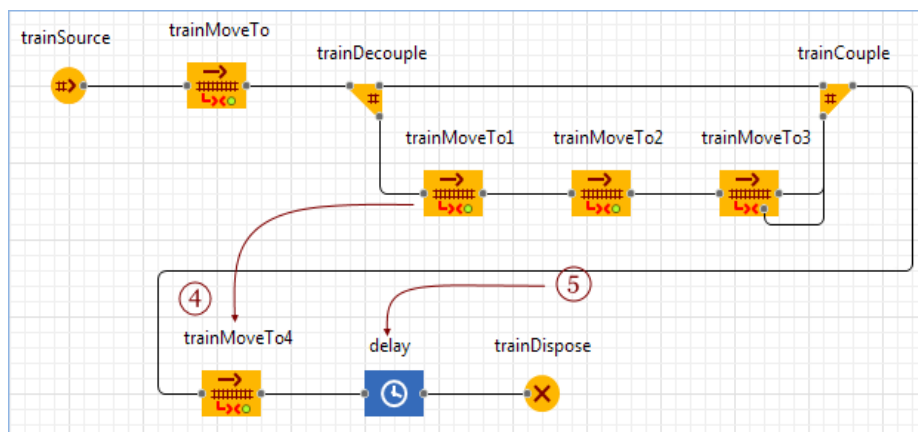
Маршрут не должен содержать: *trackArrival*

Точка ж/д пути: *stopLineEntry*

3. Обратите внимание на количество соединений, ведущих от блока *trainMoveTo3* к блоку *trainCouple*. Нижний порт блока **TrainMoveTo**, в отличие от порта *out*, который находится справа, используется поездами, которые сталкиваются с другими поездами. В нашем случае, столкновение является сцеплением локомотива, прибывающего к точке ж/д пути *stopLineArrival*, с вагонами, которые мы предварительно отцепили от этого локомотива.

Теперь внесите изменения в свойства блока *trainMoveTo3*, чтобы этот блок направлял поезд из его текущего местоположения к точке *stopLineArrival*:

Точка ж/д пути: *stopLineArrival*



Теперь, когда мы смоделировали передвижение локомотивов к концу состава, нужно добавить еще один блок **TrainMoveTo** путем Ctrl+перетаскивания (Mac OS: Cmd+перетаскивания) блока *trainMoveTo1*. Он будет моделировать процесс толкания вагонов локомотивом к горке, представленной в нашей модели элементом **Точка ж/д** пути *stopLineHump*.

4. Измените свойства блока *trainMoveTo4*:

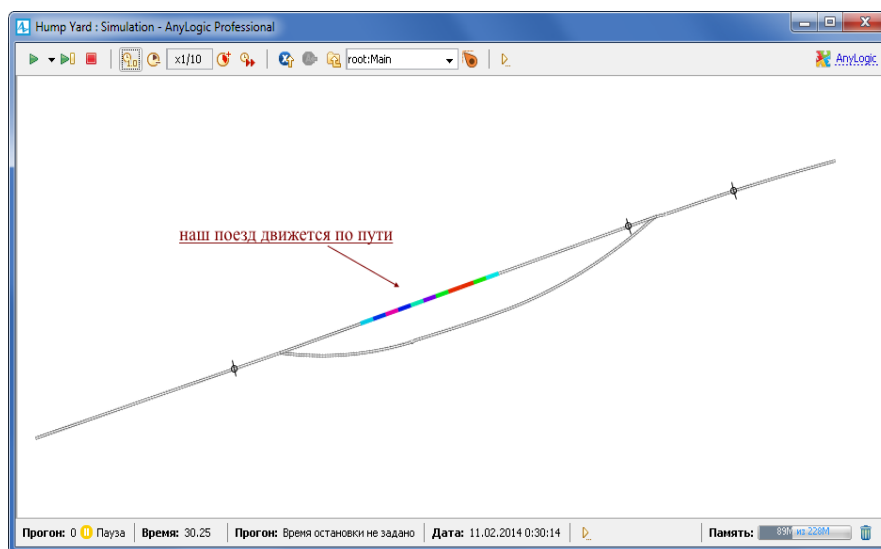
Точка ж/д пути: *stopLineHump*

5. Добавьте блок **Delay** из Библиотеки моделирования процессов, чтобы смоделировать задержку. Измените свойства блока, задав время этой задержки:

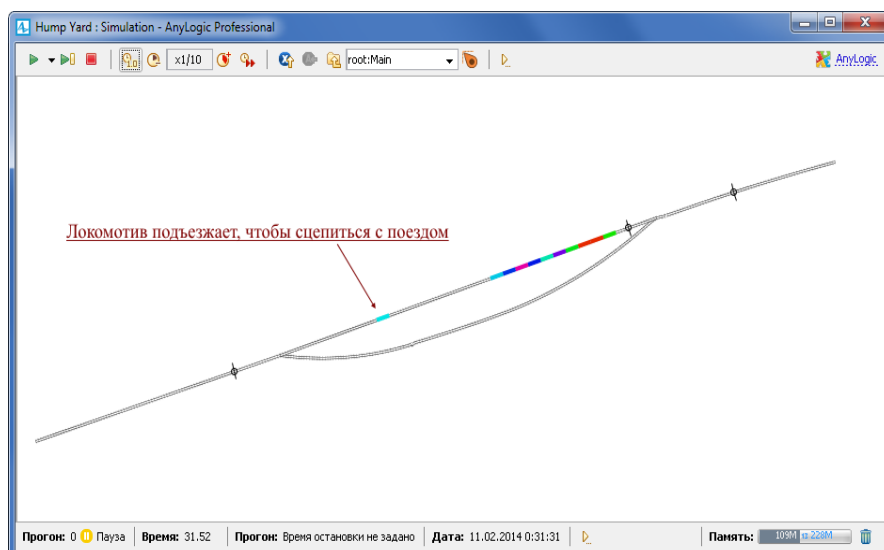
Время задержки: 15 секунд

Мы завершили Шаг 2. Можете запустить модель и посмотреть, как по ж/д путям передвигаются товарные составы.

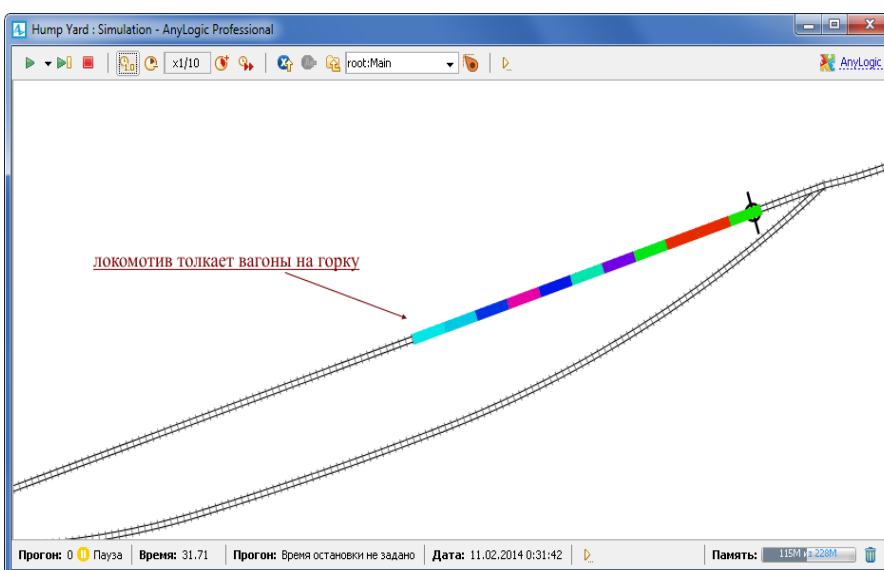
Если вы увидите ошибку "The car being created must fully be on one track", значит на пути недостаточно места для помещения поезда перед элементом **Точка ж/д пути** *stopLineEntry*. В этом случае вам нужно будет увеличить длину пути *trackEntry* и отодвинуть дальше элемент **Точка ж/д пути** *stopLineEntry* от начальной точки ж/д пути.



Локомотив подъезжает к составу, состоящему из вагонов, чтобы сцепиться с ним.



Мы также видим, как локомотив толкает вагоны на горку, после чего останавливается, вагоны отцепляются и сортируются.



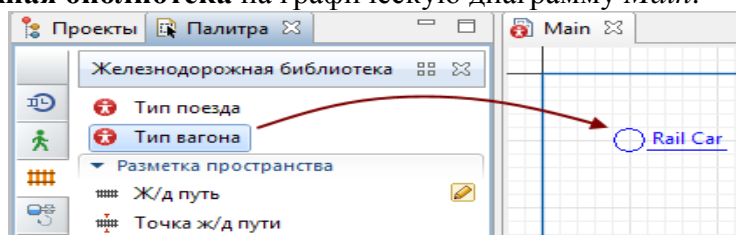
Шаг 3. Создаем типы вагонов

Как вы, должно быть, уже видели в окне работающей модели, созданной на предыдущем шаге, вагоны в нашей модели отличались исключительно цветом. Шаг 3 будет посвящен созданию различных типов вагонов.

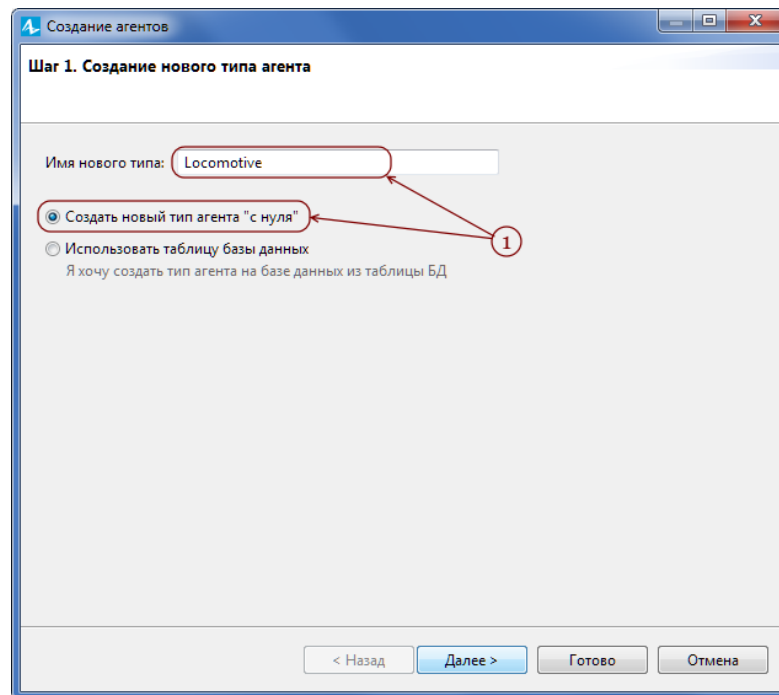
AnyLogic предоставляет удобный Мастер создания типа вагона. Просто укажите имя нового типа вагона и выберите фигуру анимации из списка готовых 3D объектов.

Задайте типы вагонов

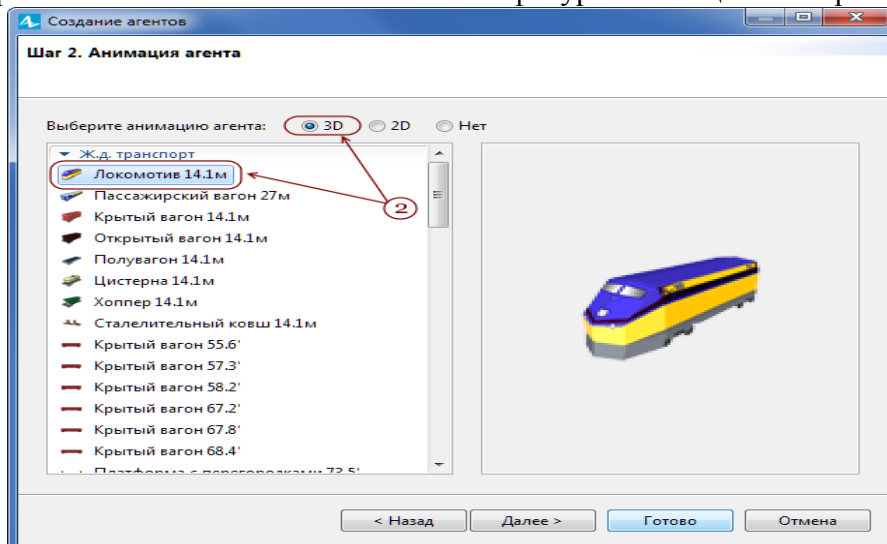
1. Создайте новый **Тип вагона**, перетащив элемент **Тип вагона** из палитры **Железнодорожная библиотека** на графическую диаграмму **Main**.



2. Откроется окно **Мастера Создание агентов**. Укажите **Имя нового типа** как *Locomotive*, оставьте опцию **Создать новый тип агента "с нуля"** выбранной. Нажмите **Далее**.



3. Выберите *Locomotive 14.1m* в качестве его фигуры анимации в 3D режиме.

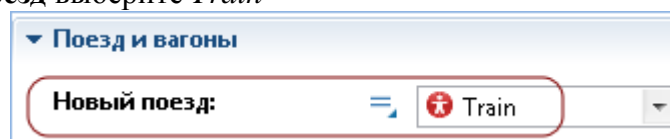


3. Добавьте еще 5 типов вагонов таким же образом. Назовите их *BoxCar*, *OpenCar*, *GondolaCar*, *TankCar* и *HopperCar*. Выберите соответствующие 3D фигуры анимации для каждого типа вагона: *Крытый вагон*, *Открытый вагон*, *Полувагон*, *Цистерна*, *Хоппер*.
4. Таким же образом создайте тип поезда (перетащив элемент **Тип поезда** из палитры **Железнодорожной библиотеки**). Назовите его *Train*.



Измените свойства блока *TrainSource*

Давайте укажем блоку *trainSource*, чтобы он создавал поезда нашего типа *Train*:

1. В поле **Новый поезд** выберите *Train*



2. В Поле **Новый вагон** введите `carSetup(carindex)`

Переключитесь к полю для введения кода, щелкнув по иконке  которая сменится на , позволив написать Java выражение. Введите следующее выражение:

Новый вагон:



```
carindex == 0 ? new Locomotive() :  
randomlyCreate( OpenCar.class,  
BoxCar.class, GondolaCar.class,  
HopperCar.class, TankCar.class )
```

Здесь мы используем условный оператор Java для того, чтобы создавать вагоны различных типов. Когда блок **TrainSource** создает новый поезд, он последовательно вызывает выражение, заданное в поле **Новый вагон**, для каждого создаваемого вагона поезда. Локальная переменная `carindex` хранит индекс только что созданного вагона. Наше выражение проверяет выполнение условия `carindex==0`. Если индекс равен нулю (то есть, мы имеем дело с первым вагоном состава), то будет выполнено выражение, написанное после знака ?

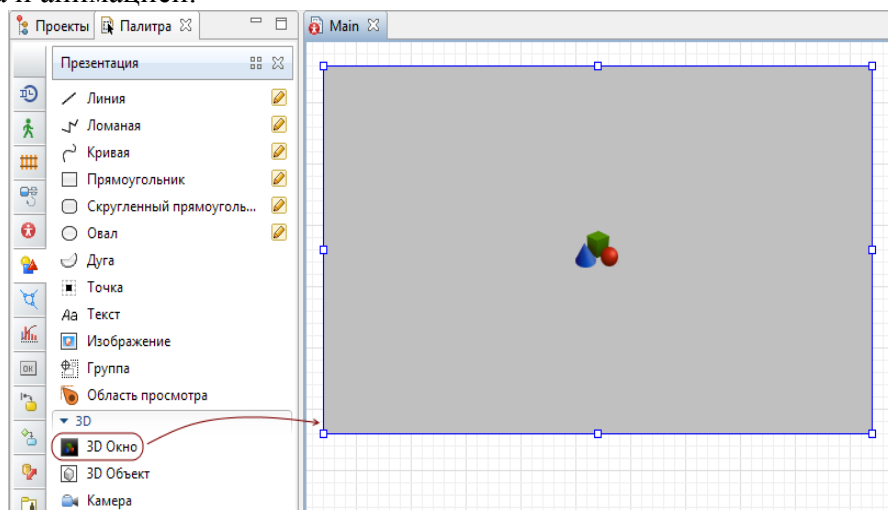
Это выражение `new Locomotive()` создаст вагон типа `Locomotive`.


Если заданное логическое условие вернет `false`, это будет значить, что сейчас создается не первый вагон состава. В этом случае мы воспользуемся функцией `randomlyCreate`, чтобы случайным образом выбрать один из типов грузовых вагонов, заданных нами на прошлом шаге. Список типов вагонов мы передаем функции в качестве ее аргументов.

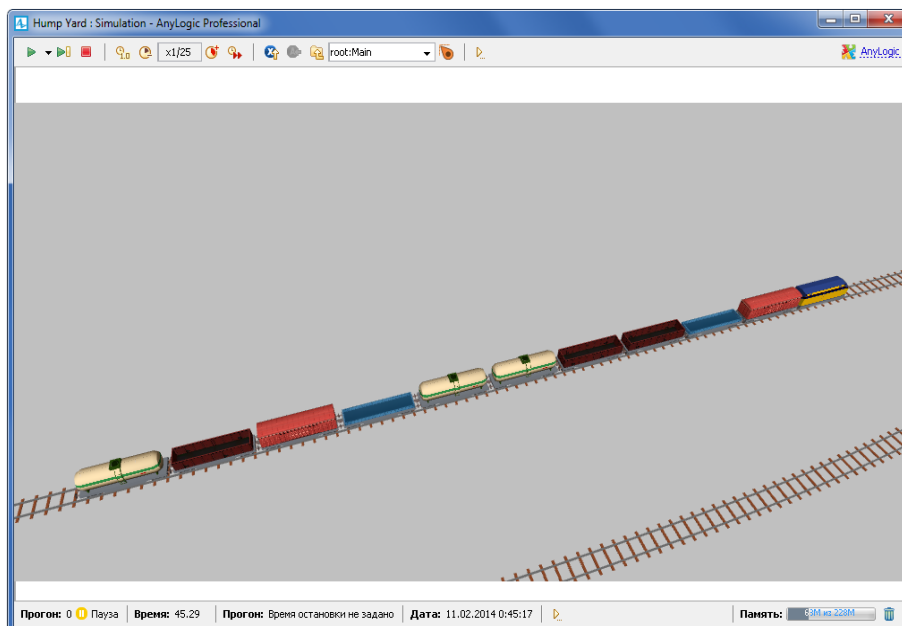
В результате, у нас будет поезд с локомотивом в качестве первого вагона, за которым следуют десять вагонов созданных нами типов.

Наблюдайте за моделью в 3D режиме

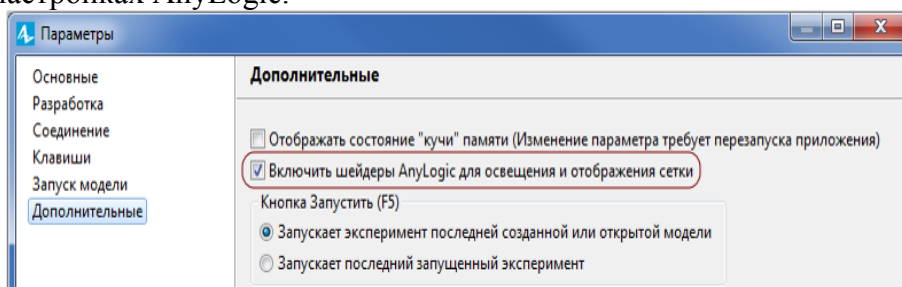
1. Добавьте **3D окно** из палитры **Презентация** и поместите его под диаграммой процесса и анимацией.



2. Запустите модель и нажмите на кнопку  **Показать область**. Выберите из выпадающего списка пункт **[window3d]**, чтобы переключиться в режим 3D просмотра. Теперь вы видите поезд, который начинается с локомотива, за которым следуют десять вагонов других типов.



Если у вас возникнут проблемы с отображением текстур 3D объектов, выключите шейдеры в настройках AnyLogic.

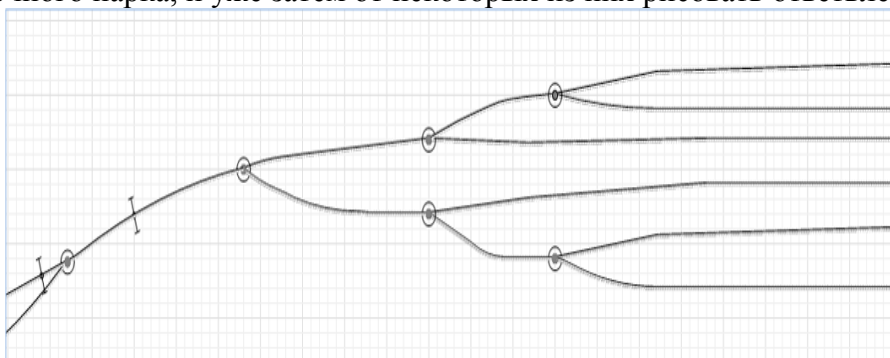


Шаг 4. Моделируем сортировку вагонов

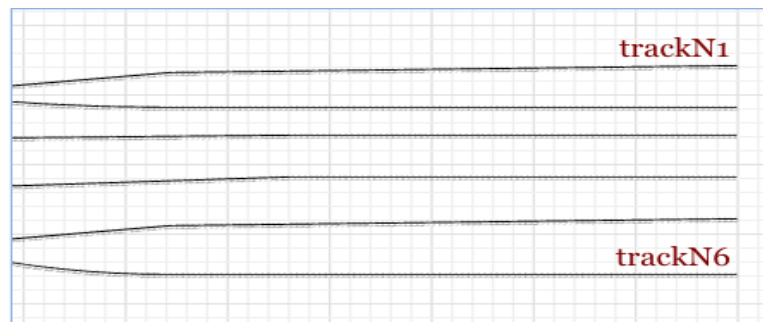
Теперь давайте промоделируем процесс сортировки вагонов и их распределения на разные пути согласно их типу.

Нарисуйте дополнительные ж/д пути

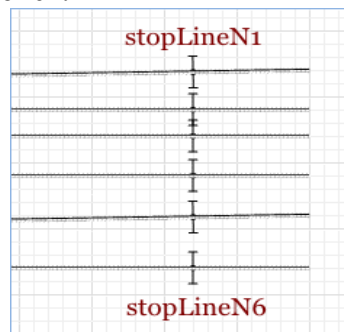
1. Нарисуйте дополнительно несколько ж/д путей, как показано на рисунке ниже. На этих путях будет происходить накопление вагонов, на каждом пути будут собираться вагоны определенного типа. Чтобы нарисовать подобную топологию ж/д путей, необходимо вначале нарисовать несколько длинных путей, ведущих до самого конца сортировочного парка, и уже затем от некоторых из них рисовать ответвления.



2. Присвойте путям осмысленные имена: *trackN1* ... *trackN6*.



- Добавьте элемент **Точка ж/д пути** на каждый путь назначения (всего шесть штук). Назовите их *stopLineN1*, ... *stopLineN6*. Таким образом, мы задаем линию остановки для каждого пути. Вагоны, которые катятся вниз под действием силы земного притяжения, будут останавливаться у этих точек.



Задайте логику диаграммы процесса

Поезд, прибывающий на сортировочную станцию, может содержать несколько вагонов одного и того же типа, расположенных друг за другом. Такие вагоны могут быть отправлены на свой путь все вместе.

- Добавьте **Функцию**, которая будет подсчитывать количество следующих одним за другим вагонов одного типа в поезде, который находится на горке. Назовите ее *carsOfSameType* и измените ее свойства следующим образом:

Свойства

carsOfSameType - Функция

Имя: carsOfSameType
Отображать имя
Исключить

Видимость:
да
Действие (не возвращает ничего)
Возвращает значение

Тип: int

Аргументы

Имя	Тип
train	Train

Тело функции

```

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

```

- Создайте еще одну **Функцию** и назовите ее *departurePointOnTrack*. Она будет определять путь назначения для вагонов в сортировочном парке (представляющем собой 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.

Внесите в свойства следующие изменения:

Свойства

departurePointOnTrack - Функция

Имя: ☒ Отображать имя

☐ Исключить

Видимость: ☒ да

☐ Действие (не возвращает ничего)

☒ Возвращает значение

Тип:

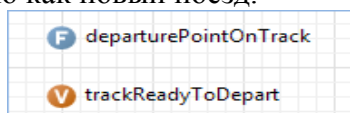
Аргументы

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;
```

3. Перетащите элемент **Переменная** в графическую диаграмму из палитры **Агент**. Эта переменная будет содержать ссылку на путь, где требуемое количество вагонов одного типа готово покинуть станцию как новый поезд.

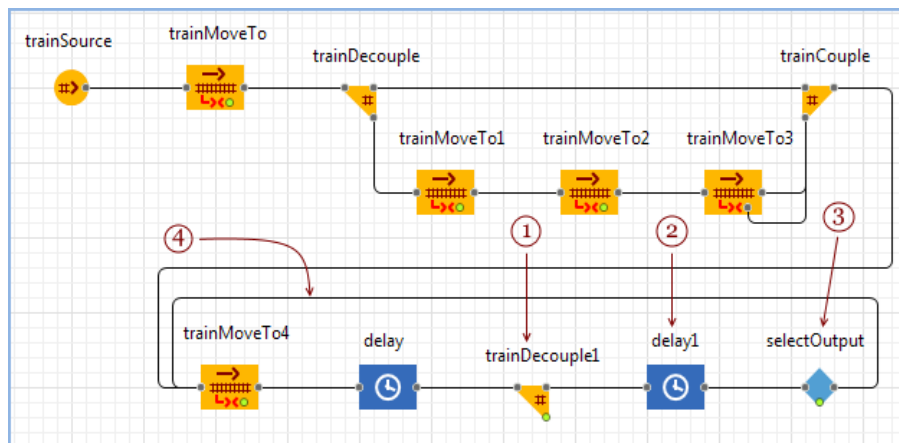


Настройте **Переменную**:

Назовите ее *trackReadyToDepart*.

Выберите *Другой* в выпадающем списке **Тип**. В появившемся справа поле выберите *RailwayTrack*.

Добавьте блоки в диаграмму процесса



1. Блок **TrainDecouple** расцепляет вагоны, когда поезд находится на горке. Поезд может содержать несколько вагонов одного типа, следующих друг за другом. Такие вагоны откатываются на свой путь назначения сразу же. (Мы определяем, сколько вагонов нужно откатить, функцией *carsOfSameType*, созданной на предыдущем шаге). Путь назначения задается созданной нами функцией *departurePointOnTrack*.

Укажите следующее в свойствах блока **TrainDecouple**:
Количество вагонов для отцепления: *carsOfSameType(train)*
Новый поезд: *Train*

2. Имеется короткая задержка перед тем, как поезд продолжит расцепляться.

Укажите следующее в свойствах блока **Delay**:

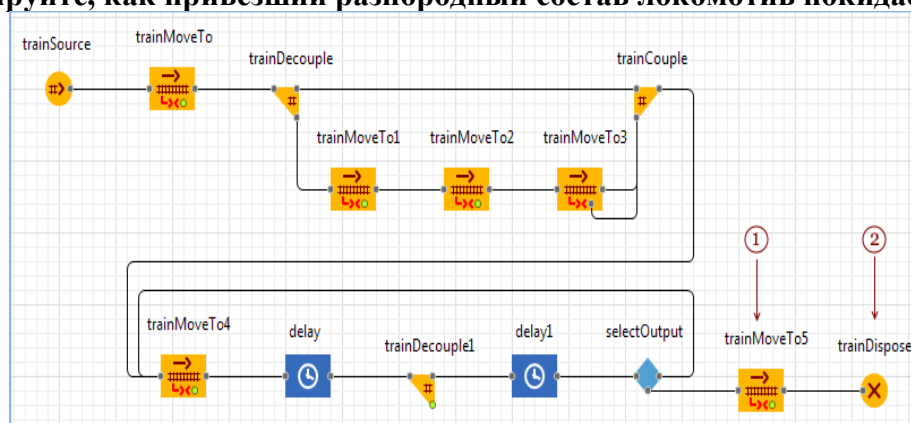
Время задержки: *5 секунд*

3. 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**. Цикл повторяется до тех пор, пока поезд не содержит только локомотив, готовый покинуть станцию.

Укажите следующее в свойствах блока **SelectOutput**:
Выход true выбирается: *при выполнении условия*
Условие: *agent.size() > 1*

4. Не забудьте присоединить правый порт блока **SelectOutput** к входному порту блока **trainMoveTo4**, чтобы вернуть поезд на горку.

Промоделируйте, как привезший разнородный состав локомотив покидает станцию



1. Мы снова будем использовать блок **TrainMoveTo**, чтобы промоделировать движение локомотива. В качестве точки назначения мы укажем нижний путь (*trackN6*).

Укажите следующее в свойствах блока **TrainMoveTo**:

Маршрут: *Вычисляется автоматически...*

Цель движения: *Заданное смещение на пути*

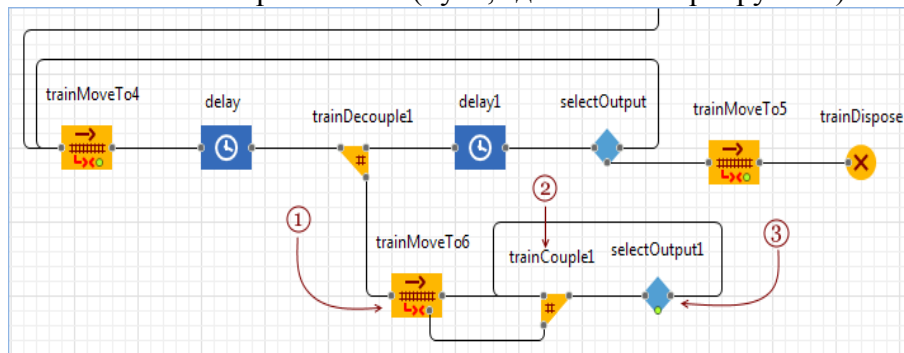
Путь: *trackN6*

Крейсерская скорость: *5 м/с*

2. В конце концов, локомотив удаляется из модели блоком **TrainDispose**.

Задайте логику скатывания вагонов на пути сортировочного парка

Промоделируйте скатывание вагонов на свои пути назначения в сортировочном парке под действием силы земного притяжения (пути, где вагоны сортируются).



1. Движение, как обычно, моделируется блоком **TrainMoveTo**. Поезд содержит отдельные вагоны или сцепленные вагоны одного типа. Назначение задается функцией *departurePointOnTrack*, которую мы указали ранее.

Укажите следующее в свойствах блока **TrainMoveTo**:

Маршрут: *Вычисляется автоматически...*

Цель движения: *Заданная точка пути\Точка ж/д пути:* `departurePointOnTrack(train)`

Крейсерская скорость: *5 м/с*

При окончании движения: *Затормозить и остановиться*

2. Если на пути назначения уже есть какие-то вагоны, то они комбинируются вместе с только что прибывшими вагонами в блоке **TrainCouple**.

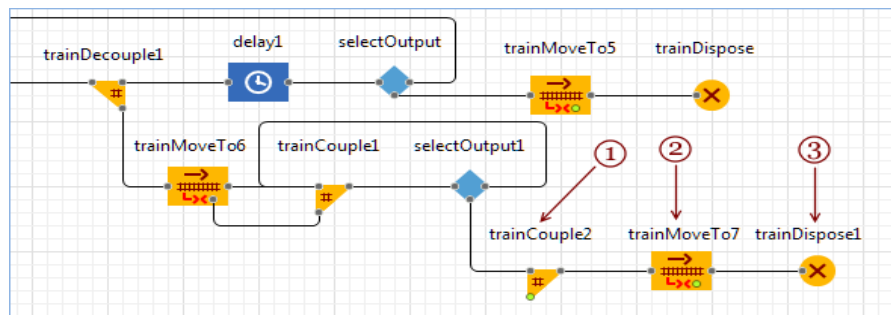
3. Вагоны продолжают соединяться, пока не будет достигнуто определенное количество вагонов на пути (тогда локомотив приезжает туда и забирает вагоны со станции). Блок **SelectOutput** проверяет, когда сбор вагонов можно прекратить. Мы указываем, что количество собранных вагонов должно быть меньше 8.

Укажите следующее в свойствах блока **SelectOutput**:

Выход true выбирается: *при выполнении условия*

Условие: `agent.size() < 8`

Добавьте логику отправления готового поезда, собранного из вагонов одного типа, со станции



1. Сначала мы сцепляем вагоны с локомотивом при помощи блока **TrainCouple** (логику поведения локомотива мы зададим на следующем шаге).

2. Затем поезд покидает станцию (это моделируется блоком **TrainMoveTo**).

Укажите следующее в свойствах блока **TrainMoveTo**:

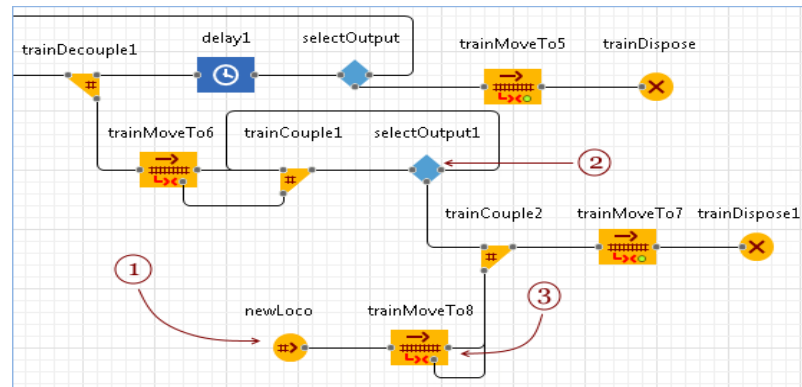
Маршрут: *Не задан (поезд будет следовать согласно стрелкам)*

Цель движения: *Не задана*

3. В конце концов мы удаляем поезд из модели блоком **TrainDispose**.

Добавьте логику локомотива, забирающего готовые к отправке вагоны

Сейчас мы смоделируем, как локомотивы забирают со станции готовые поезда, составленные из вагонов одного типа.



1. Сначала давайте добавим блок **TrainSource**, моделирующий прибытие 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 определяем путь - значение этой переменной - на следующем шаге).

Укажите следующее в свойствах блока **TrainSource**:

Имя: *newLoco*

Поезда прибывают согласно: *Вызовом метода 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 метров*

Новый поезд: *Train*

Новый вагон: *Locomotive*

2. Этот блок проверяет, имеется ли достаточное количество вагонов одного типа на пути или еще нет.

Укажите следующее в свойстве **При выходе (false)** секции **Действия** блока **SelectOutput**:

trackReadyToDepart = agent.getTrack(true);

newLoco.inject();

Действие **При выходе (false)** выполняется, когда появляется требуемое количество вагонов.

Строка *trackReadyToDepart = agent.getTrack(true);* получает текущий путь, на котором скопились готовые к отправке вагоны, и записывает его в локальную переменную *trackReadyToDepart*.

Строка *newLoco.inject();* генерирует новый локомотив в блоке *newLoco*. Когда локомотив создан, он следует к блоку, который мы создаем далее.

3. Блок **TrainMoveTo** моделирует движение локомотива к готовым к отправке вагонам. Обратите внимание на направление его 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.

Укажите следующее в свойствах блока **TrainMoveTo**:

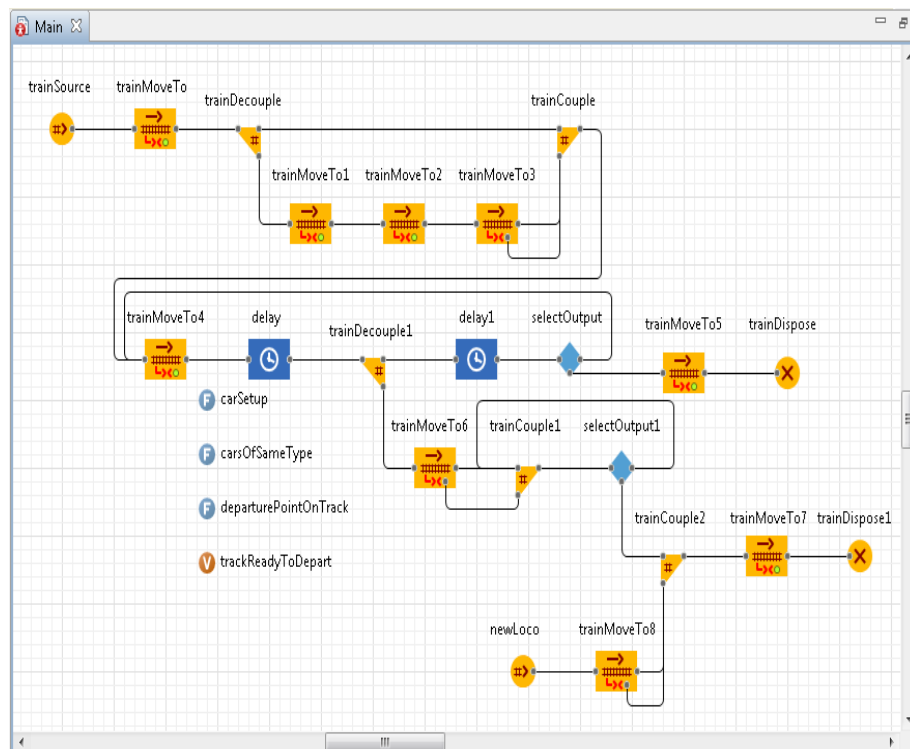
Направление движения: *Назад*

Маршрут: *Не задан (поезд будет следовать согласно стрелкам)*

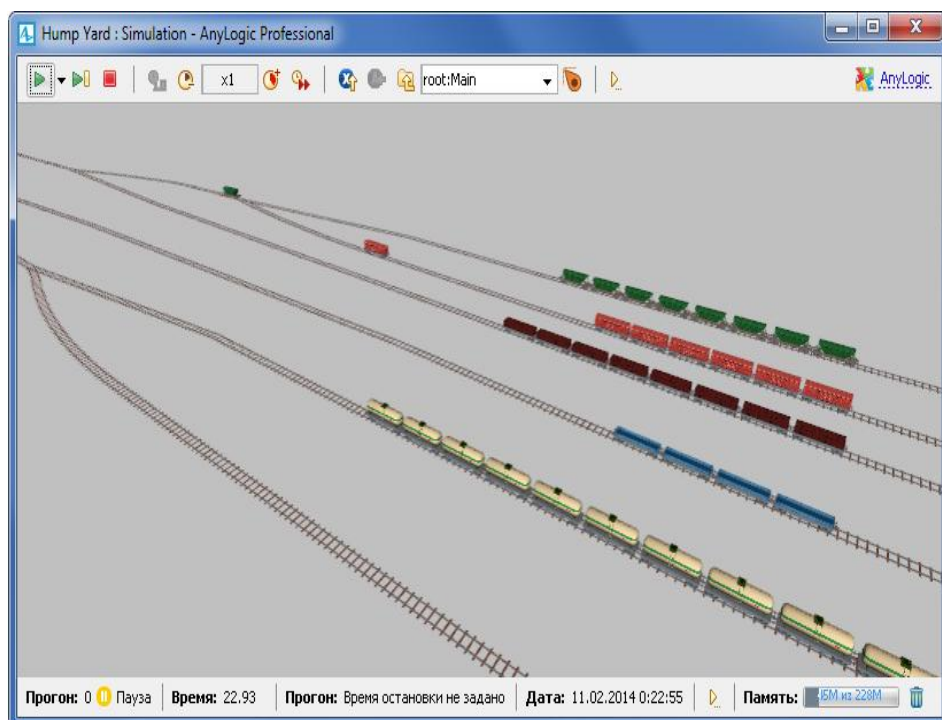
Цель движения: *Не задана*

Крейсерская скорость: *10 м/с*

В итоге, ваша диаграмма должна выглядеть так:



Запустите модель. Вы увидите, как вагоны сортируются по типу. Когда набирается достаточное количество вагонов одного типа, локомотив забирает их с сортировочной станции.



Мы завершили решение небольшой задачи, типовой для железнодорожной библиотеки AnyLogic. Чаще эта библиотека используется не для моделирования железнодорожных сетей целых регионов, а моделирует именно определенные участки - грузовые или пассажирские ж/д узлы.