Выполним первую тренировочную задачу:

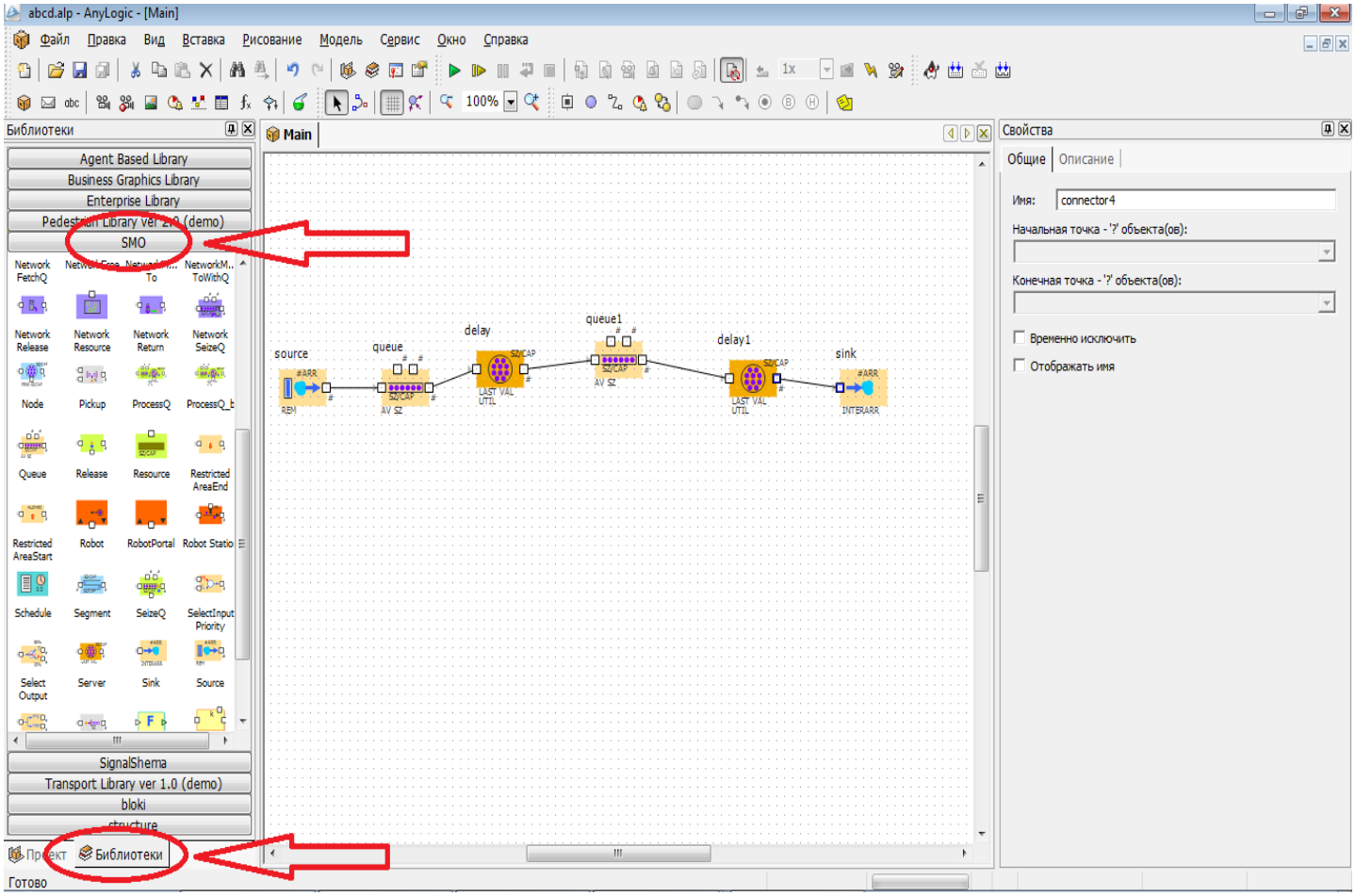
Поток заявок распределен по равномерному закону со средним 100 и отклонением от среднего 20. Заявки обслуживаются последовательно двумя приборами. Время обслуживания распределено по равномерному закону: первый прибор 150,30; второй - 100,50 (единица измерения времени 1 с). Промоделируйте процесс обслуживания 500 заявок. Ответьте на следующие вопросы:

- Какую часть времени были заняты приборы?
- Какое количество заявок обслужено каждым прибором?
- Сколько заявок поступило за это время?
- Возникали ли очереди перед каждым из приборов?
- Измените параметры генератора заявок и приборов так, чтобы исчезла очередь перед первым прибором и появилась перед вторым; исчезли очереди вообще?

- Промоделируйте задачу в течение 3 часов.

Для этого нам понадобится файл описания библиотеки СМО, который называется SMO_min. И ещё пустой проект в среде Anylogic. Напоминаю, что сам проект и любые объекты внутри него называем только английскими буквами.

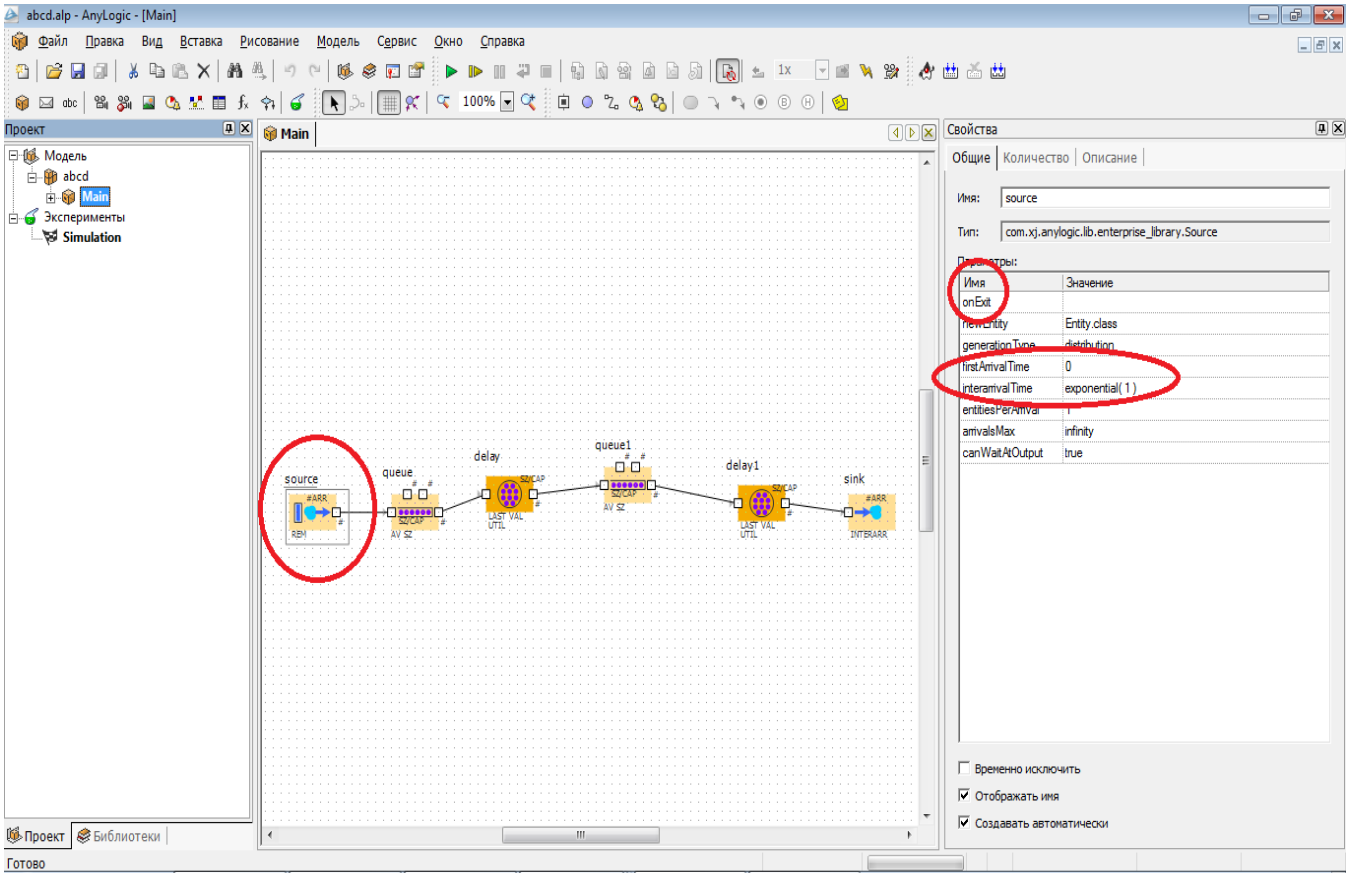
В пустом проекте находим в нижнем левом углу вкладку «Библиотеки» и выбираем «SMO». Перетаскиваем нужные нам блоки на рабочую область и соединяем эти блоки как показано на рисунке. Блок source является генератором заявок, который создаёт заявки для нашей системы массового обслуживания. Блок sink – удаляет заявки из системы после того как они будут обслужены. Блок queue – моделирует очередь. Блок delay задерживает заявку на заданное время. Таким образом, после создания заявка идёт на первый прибор, и если к нему нет очереди, то сразу обрабатывается. В противном случае встаёт в очередь и ждёт освобождения прибора. После обработки на первом приборе заявка аналогично взаимодействует со вторым, далее она удаляется из системы блоком sink.



Теперь подробнее про каждый блок.

Стоит сказать, что переход из одного блока в другой происходит мгновенно и на это времени не тратится.

Выберем блок source и посмотрим на его свойства и характеристики в правой части окна. Нас интересует несколько полей этого блока. Поле onExit нужно чтобы выполнялись какие то действия после того как заявка будет покидать блок. Туда вставляется программный код на языке java. В поле firstArrivalTime записывается время генерации первой заявки, она обычно остаётся равным нулю, так как обычно первая заявка генерируется в момент запуска системы массового обслуживания. Если нужно чтобы первая заявка в систему попала не в момент запуска, тогда измените значение в этом поле с 0 на то количество секунд, которое потребуется для генерации первой заявки. Следующее поле interArrivalTime – сюда ставится количество секунд между генерациями заявок. В нашей задаче сказано - Поток заявок распределен по равномерному закону со средним 100 и отклонением от среднего 20. Это означает, что время генерации составит от 80 до 120 секунд. Запишем в это поле – ***uniform(80,120)***, это функция случайно выбирает число из промежутка в скобках.



Теперь блок queue. Он имитирует очередь и в нем можно выбрать тип очереди:

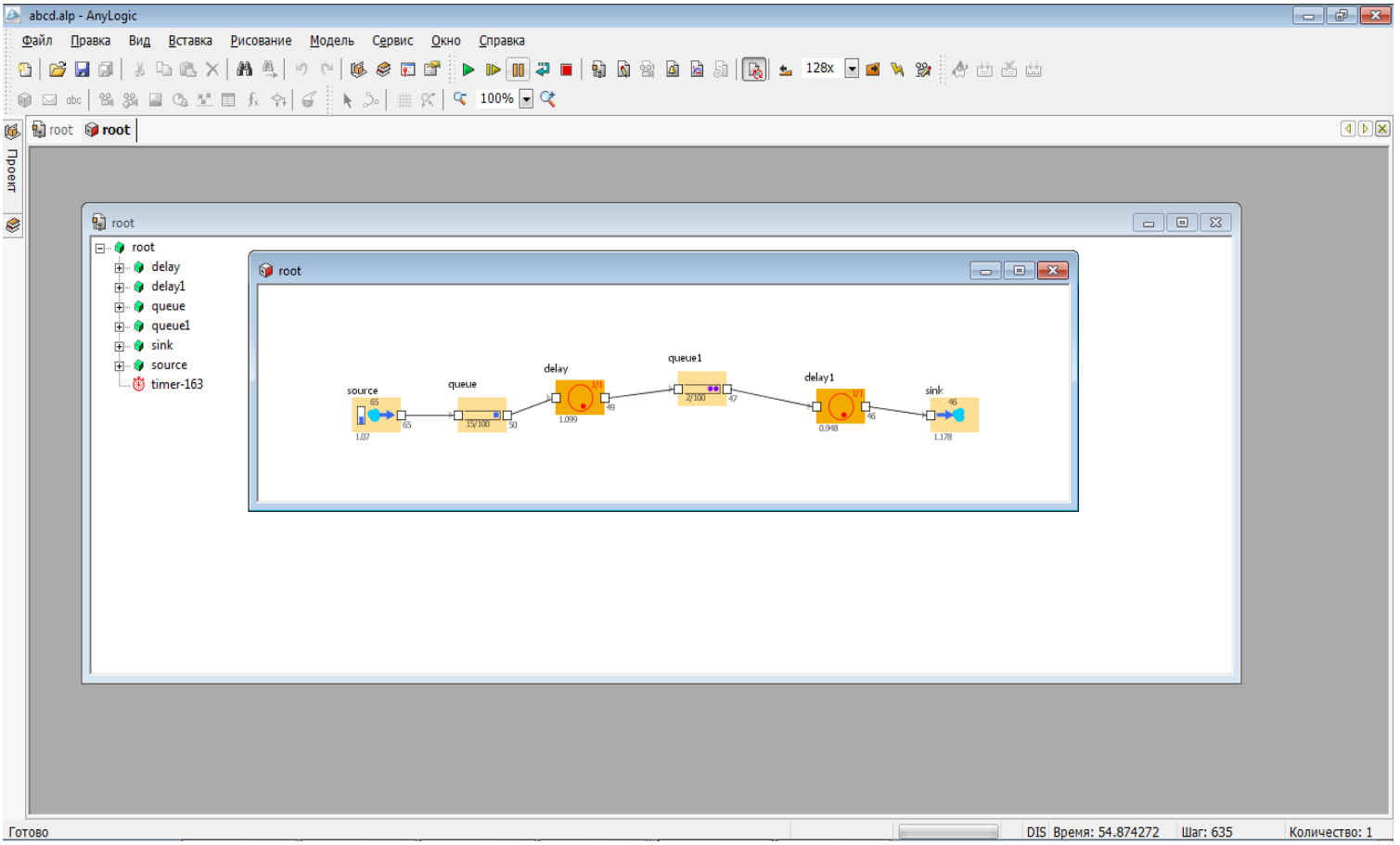
1. FIFO – первым пришёл, первым ушел (обычная очередь)
2. LIFO – последним пришёл, первым ушел
3. RANDOM – в случайном порядке
4. PRIORITY – очередь по приоритетам

У нас обычная очередь. Поэтому оставляем FIFO. Поле capacity – вместительность, разворачиваем список и выбираем ***infinity*** (бесконечная очередь), так как мы не знаем, сколько у нас будет заявок в системе, а значит, не знаем, сколько могут из них стоять в очереди. Так как у блока queue есть и вход и выход, то в полях onEnter и onExit мы можем прописать программные инструкции для выполнения определённых действий, но об этом позже. Аналогично для очереди перед вторым прибором.

В блоке delay нас, прежде всего, интересует параметр delayTime (время задержки заявки). В задаче сказано - первый прибор 150,30; второй - 100,50. Значит, для первого прибора мы запишем в этом поле ***uniform(120,180)***, для второго - ***uniform(50,150)***. Вместительность (поле capacity) оставляем 1, так как в каждый момент времени прибор может обслуживать только одну заявку и не более.

В блоке sink есть только одно поле, которое на данный момент заполнять нам не нужно.

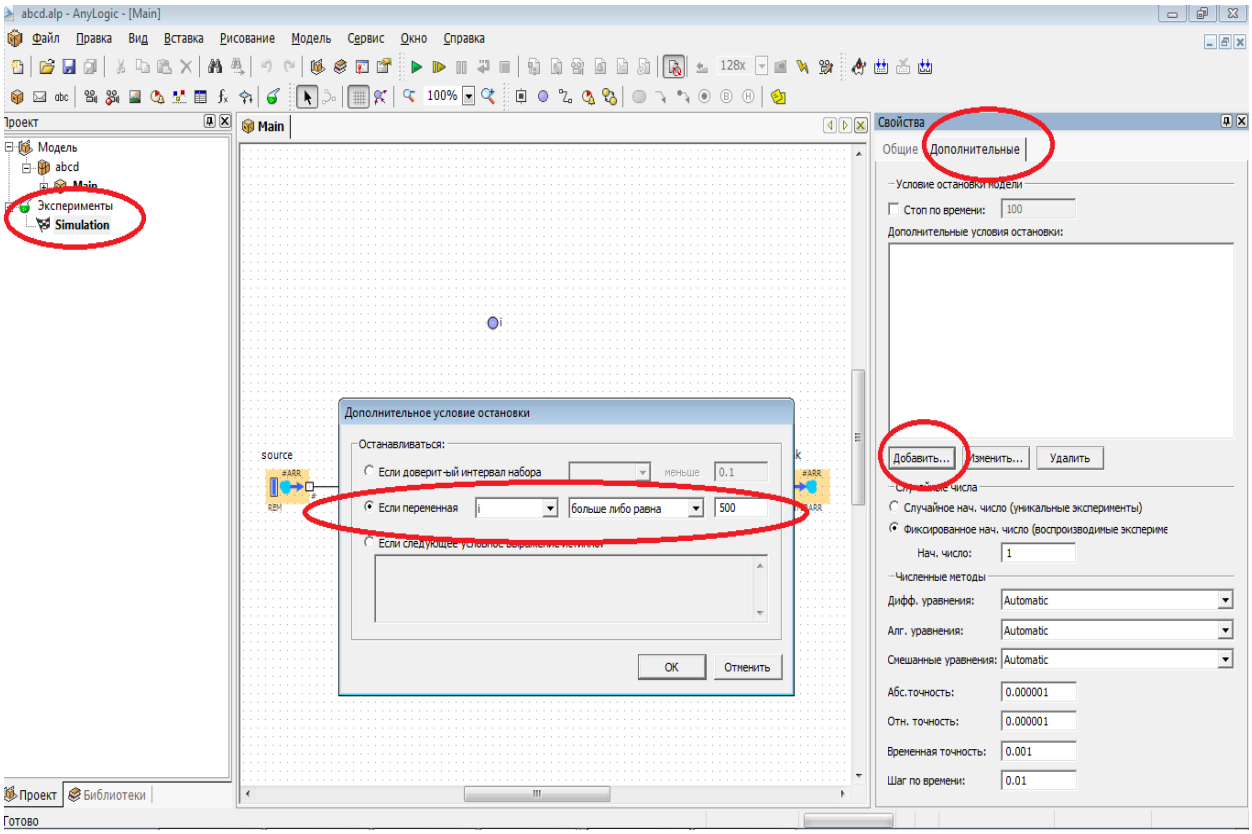
Запускам эксперимент и смотрим на анимацию. Вы можете увидеть статистическую информацию рядом с блоками, внутри них и нажав на них.



Например, сколько вышло заявок из генератора(65) , сколько обслужено каждым прибором(49,46), сколько выведено из системы(46), сколько сейчас в очередях(15,2).

Осталось поставить одно условие – обслужить 500 заявок. Для этого нам понадобится одно из полей onExit или onEnter и новая переменная I (тип real, по умолчанию ничего не ставим), которую мы поставим на рабочей области рядом со СМО. Мы можем точно утверждать что заявка закончила своё «путешествие» по СМО после того как она была обслужена на втором приборе. И поскольку переход между блоками мгновенный, то у нас есть 2 варианта в каком месте СМО мы можем считать обслуженные заявки – это на выходе с блока delay1 или на входе в блок sink. Давайте на блоке sink. Выбираем его, и справа в окне свойств в поле onEnter пишем программную инструкцию, которая будет увеличивать переменную i, которая является своеобразным счётчиком. Итак, в поле запишем ***i=i+1*** или можно проще ***++i***

Таким образом, мы в переменой I будем считать обработанные заявки. Теперь нам нужно настроить эксперимент так, чтобы когда счётчик стал равным 500, он заканчивался. Для этого в дереве объектов слева выбираем объект simulation, смотрим в окно свойств справа, выбираем вкладку Дополнительные, нажимаем кнопку Добавить и в появившемся окне формируем условие – если переменная I больше либо равна 500.



Запускаем эксперимент и смотрим, что он остановится в момент, когда рядом с блок sink будет цифра 500.

На этом основная часть задачи закончена, однако, остаются вопросы к этой задаче, на которые мы сможем ответить также с помощью проекта в Anylogic.

Какую часть времени были заняты приборы?

Для этого поставим 2 новые переменные, которые будут рассчитывать время работы приборов – dt1 и dt2 соответственно для первого и второго приборов. Дело осложняется тем, что мы не знаем точно, сколько приборы тратят на каждую заявку, так как время мы задали как случайную величину из промежутка (120,180) и (50,150) соответственно. Для того чтобы узнать это нам нужно обратиться к руководству SMO_min (страница 21, описание блока delay). В описании каждого блока есть таблицы с параметрами и функциями, которые доступны в этом блоке. Нам нужен параметр

delayTimeValue (в одно слово, без пробелов и так как написано здесь со строчными и заглавными буквами, в руководстве могут быть опечатки).

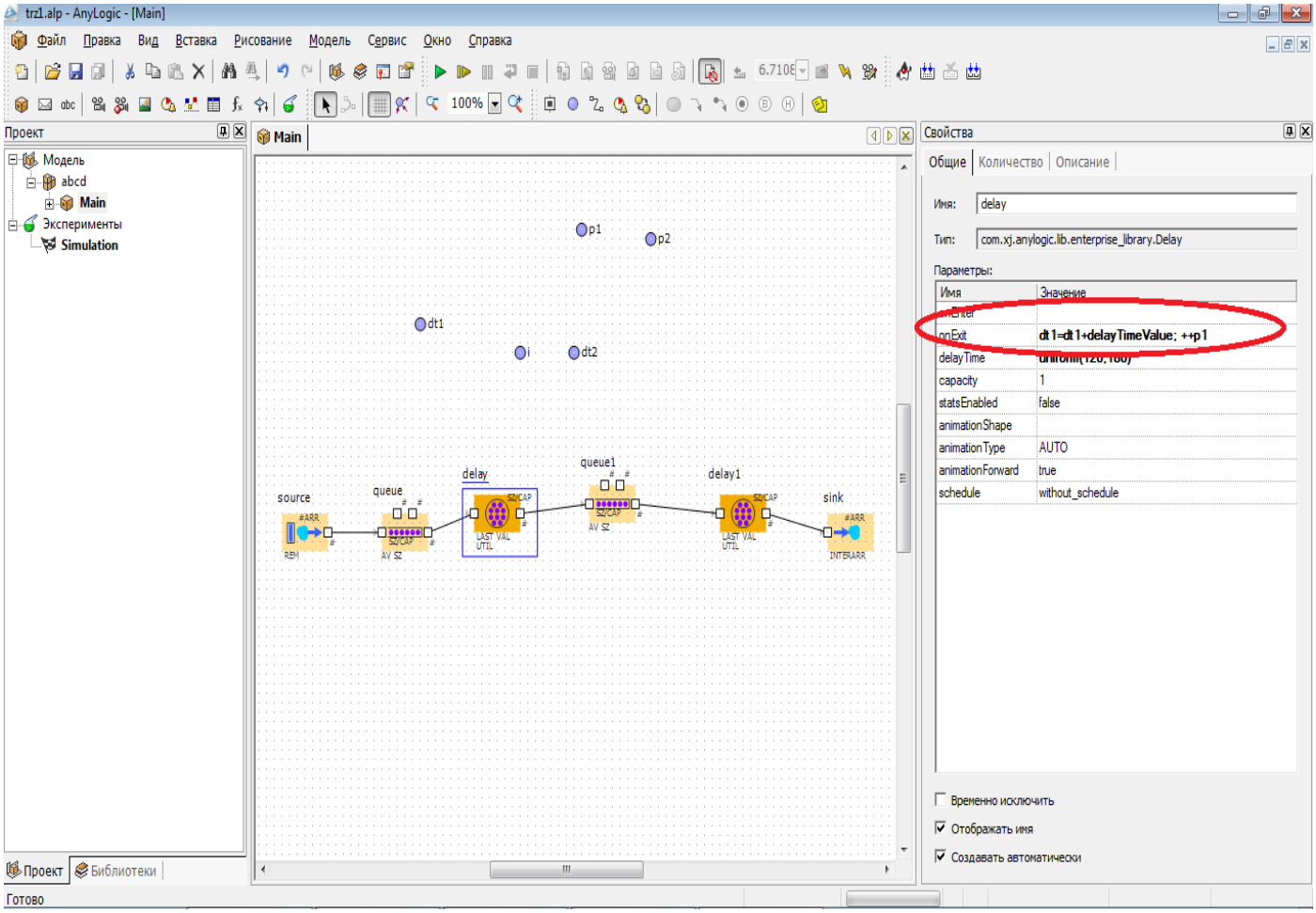
Пересчитывать суммарное время работы прибора нам нужно каждый раз когда заявка уходит из него. Поэтому в блоке delay в поле onExit мы пишем ***dt1=dt1+delayTimeValue***

Тоже самое для второго прибора и переменной dt2

После этого, запускаем эксперимент и ставим новую диаграмму, на которую вытаскиваем переменные dt1 и dt2. Мы можем видеть, что поначалу первый прибор работает «много», а второй простаивает, но потом и второй прибор «включается» в работу.

Какое количество заявок обслужено каждым прибором?

Создадим 2 новые переменные – p1 и p2 для того чтобы считать сколько заявок обслужено первым и вторым прибором соответственно. Для подсчёта используем тот же приём что и со счётчиком заявок на блоке sink. В блоках delay и delay1 в поле OnExit поместим инструкции ***++p1*** и ***++p2*** соответственно.



Возникали ли очереди перед каждым из приборов?

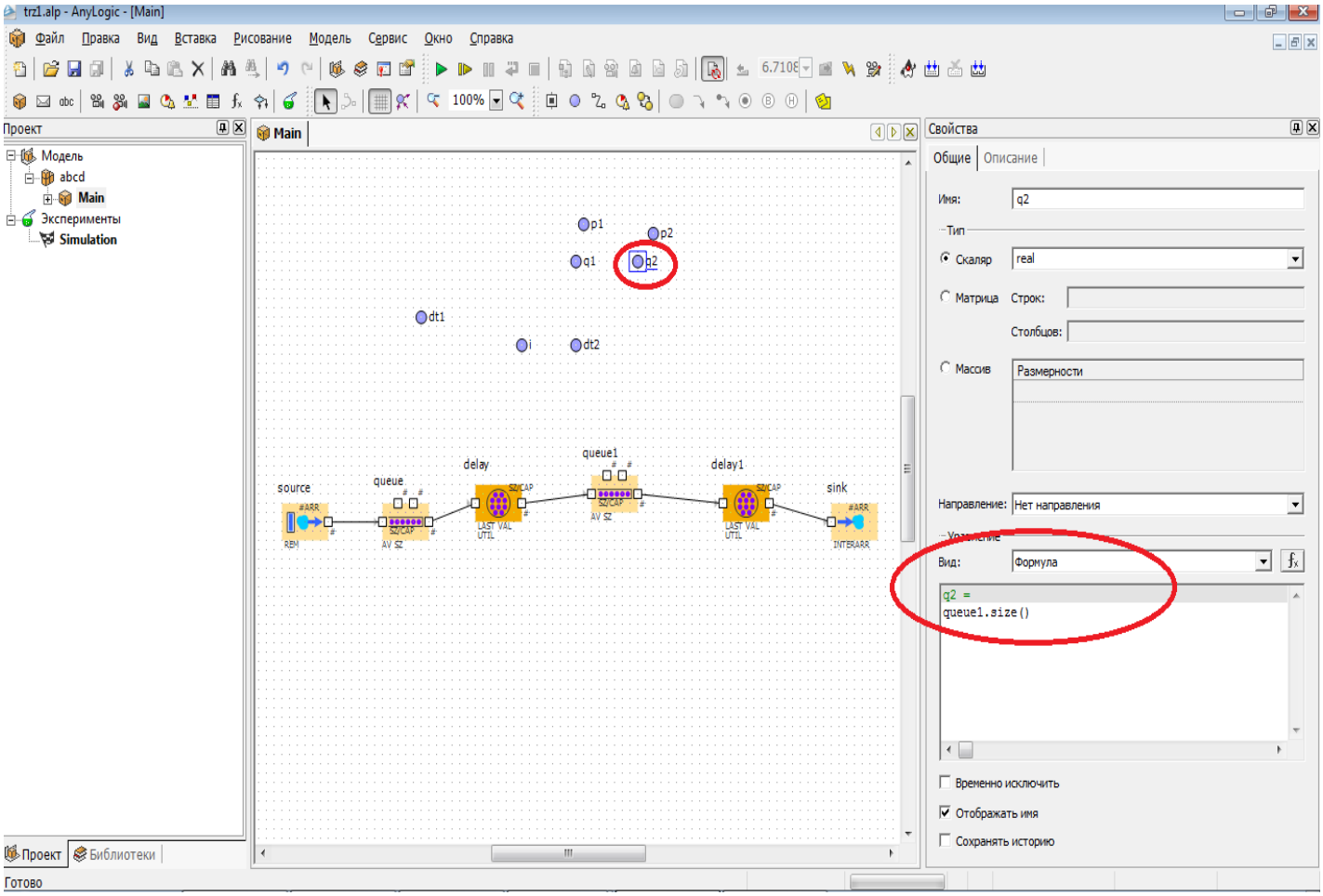
Создадим 2 новые переменные – q1 и q2 для того чтобы считать сколько заявок стоит в очередях перед первым и вторым прибором соответственно.

Нам понадобится функция блока queue, а именно функция ***size().*** Список всех параметров и функций блока queue мы можем посмотреть в к руководстве SMO_min (со страницы 17 и далее)

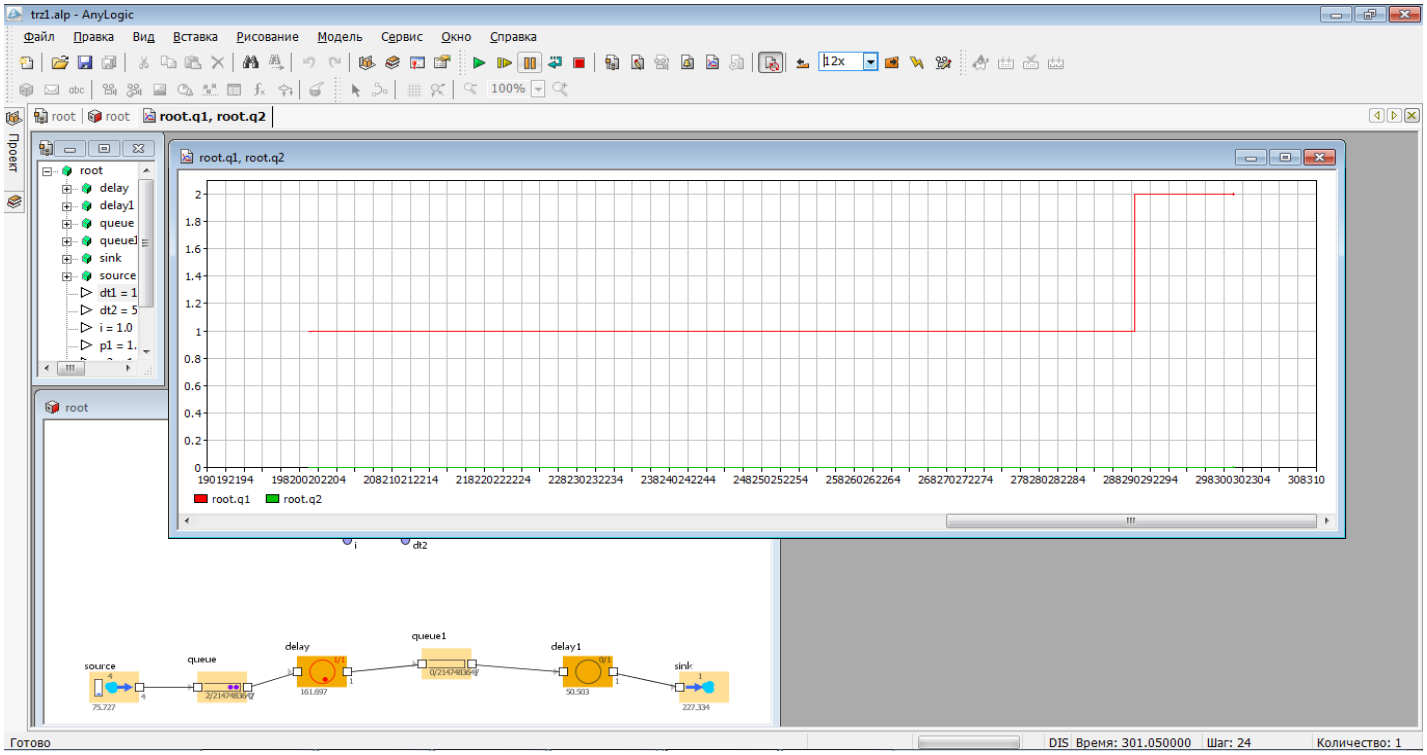
Выбираем переменную q1 и делаем её вид – Формула. Записываем после знака равно выражение ***queue.size()***

Для переменной q2 делаем то же самое, только выражение будет ***queue1.size()***

Таким образом, мы видим, что принципы объектно-ориентированного программирования действуют в создаваемых проектах. Мы выбрали объект (сначала блок queue, а после queue1) и через точку выбрали метод для этого объекта (функции. size).



Запускаем эксперимент и выносим на новую диаграмму переменные q1 и q2, чтобы видеть, сколько заявок было в очередях в каждый момент времени. После определённого времени видим, что перед первым прибором очередь составляет 2 заявки, а перед вторым в очереди заявок нет.

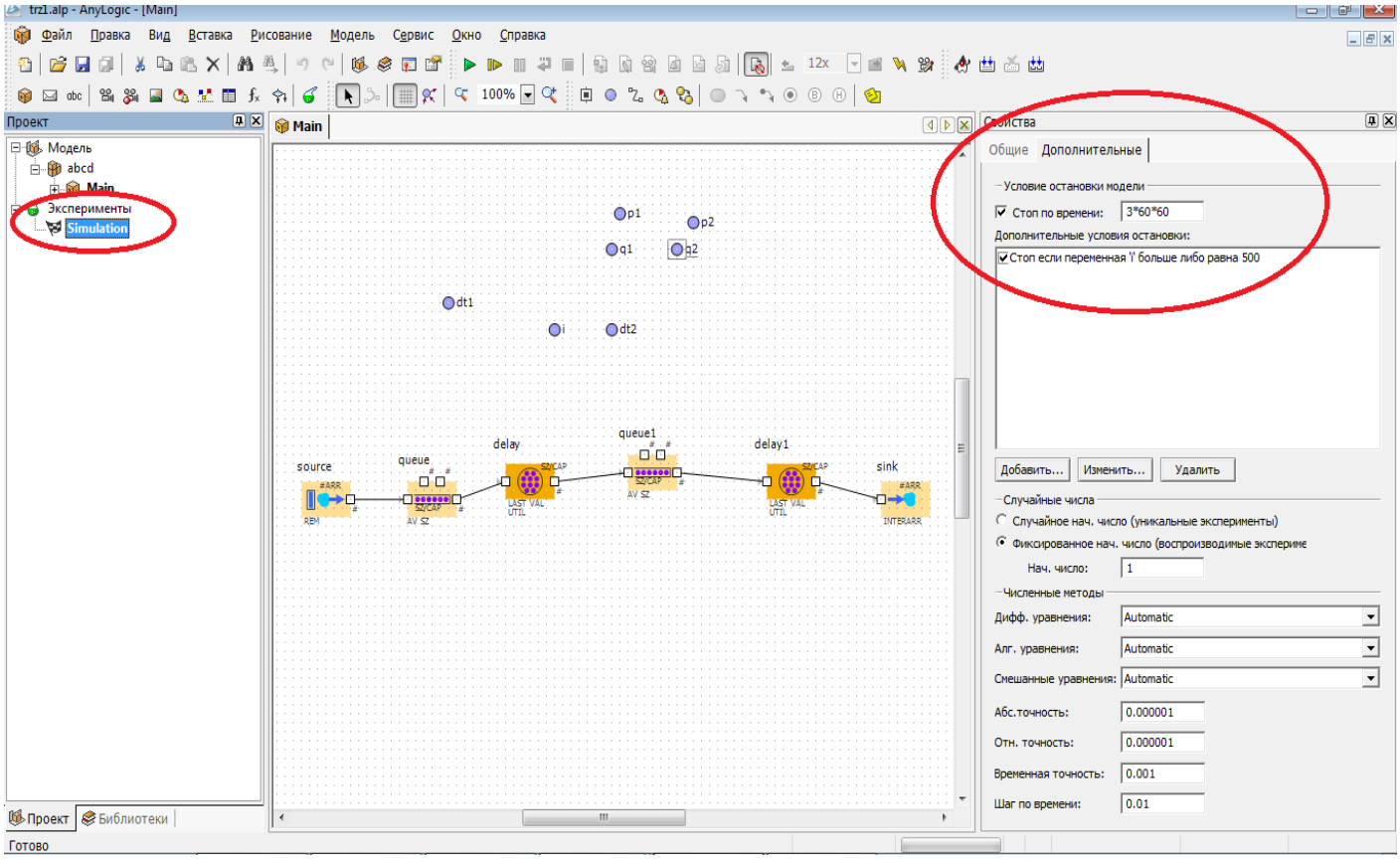


Измените параметры генератора заявок и приборов так, чтобы исчезла очередь перед первым прибором и появилась перед вторым; исчезли очереди вообще?

Для этого надо поправить поле interArrivalTime и delayTime , а конкретнее в генераторе заявок можно увеличить время создания заявки, а в блоке delay наоборот время обработки заявки уменьшить.

Промоделируйте задачу в течение 3 часов

Для этого нужно зайти в Дополнительные параметры эксперимента ***Simulation*** и поставить галочку в поле ***Стоп по времени***, а после задать 3 часа в секундах (10800 или можно написать выражением 3*60*60).


trz1.alp - AnyLogic - [Main]
Файл Правка Вид Вставка Рисование Модель Сервис Окно Справка
Проект
Модель
abcd
Main
Эксперименты
Simulation
Main
p1
p2
q1
q2
dt1
i
dt2
source
queue
delay
queue1
delay1
sink
Свойства
Общие Дополнительные
Условие остановки модели
Стоп по времени: 3*60*60
Дополнительные условия остановки:
Стоп если переменная 'i' больше либо равна 500
Добавить... Изменить... Удалить
Случайные числа
Случайное нач. число (уникальные эксперименты)
Фиксированное нач. число (воспроизводимые эксперименте
Нач. число: 1
Численные методы
Дифф. уравнения: Automatic
Алг. уравнения: Automatic
Смешанные уравнения: Automatic
Абс.точность: 0.000001
Отн. точность: 0.000001
Временная точность: 0.001
Шаг по времени: 0.01
Проект Библиотеки
Готово