

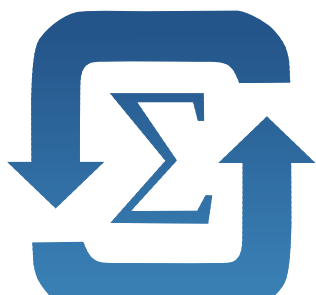
Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Кафедра теоретической и прикладной информатики

Лабораторная работа № 4
по дисциплине «Компьютерное моделирование»



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИ-61
СТУДЕНТЫ:	Ершов П.К., Мамонова Е.В., Цыденов З.Б.
ПРЕПОДАВАТЕЛЬ:	Черникова О. С. Карманов В. С.

Новосибирск

2020

1. Цель работы

Научиться строить модели системы управления цепями поставок.

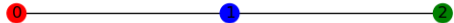
2. Ход работы

Модель цепи поставок основана на известной деловой игре «BeerGame» разработанной группой профессоров Массачусетского Технологического Института (MIT). Задача цепи поставок состоит в том, чтобы произвести и поставить конечному потребителю пиво: фабрика производит, а другие три звена цепи поставок продвигают пиво, пока оно не достигает конечного потребителя в конце системы поставок.

У каждого участника цепи есть метод оценки величины заказа, история заказов, текущий объём товара, товар, находящийся в пути (поставки). Сделанный кем-либо из участников заказ поступает на его склад через 1 период (в разработанной модели 1 неделя). Для упрощения задачи производитель изготавливает свою продукцию с такой же задержкой. Спрос на рынке является случайным (рассмотреть разные распределения).

За хранение товара на складе каждый участник цепи платит арендную плату. Расходы, связанные с содержанием сбытовых запасов – это упущенные доходы, которые могли быть получены, если бы деньги, вложенные в запасы, были использованы в других целях. Поскольку объём складских помещений предполагается постоянным, то расходы на хранение также не будут меняться. Кроме того, к потерям относятся излишки сбытовых запасов, имеющиеся сверх их минимально необходимого количества. Запасы, превышающие минимально необходимый уровень, не используются, что означает наличие упущенной выгоды, которая равна сумме прибыли, которая могла быть получена в случае размещения данных средств, например, на депозите в банке. Если на момент заказа у поставщика недостаточно товара, то он закупает его у конкурента по завышенной цене. Различие между собственной ценой и ценой конкурента является параметром задачи.

Вариант:

Номер варианта	Алгоритмы заказа	Тип структуры сети поставок
2	1 2, $h=3$	 Линейная, 3 участника

Алгоритм № 1.

Размер текущей закупки определяется заказом, который поступил последним. Это самая простая («наивная») стратегия, которая оптимальна при стабильном рынке:

$$o_i^P = d_i^P - s_i^P.$$

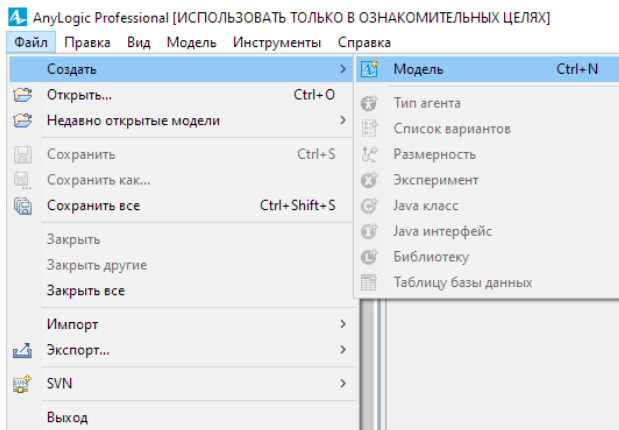
Алгоритм № 2.

Объём заказа определяется через скользящее арифметическое среднее порядка h :

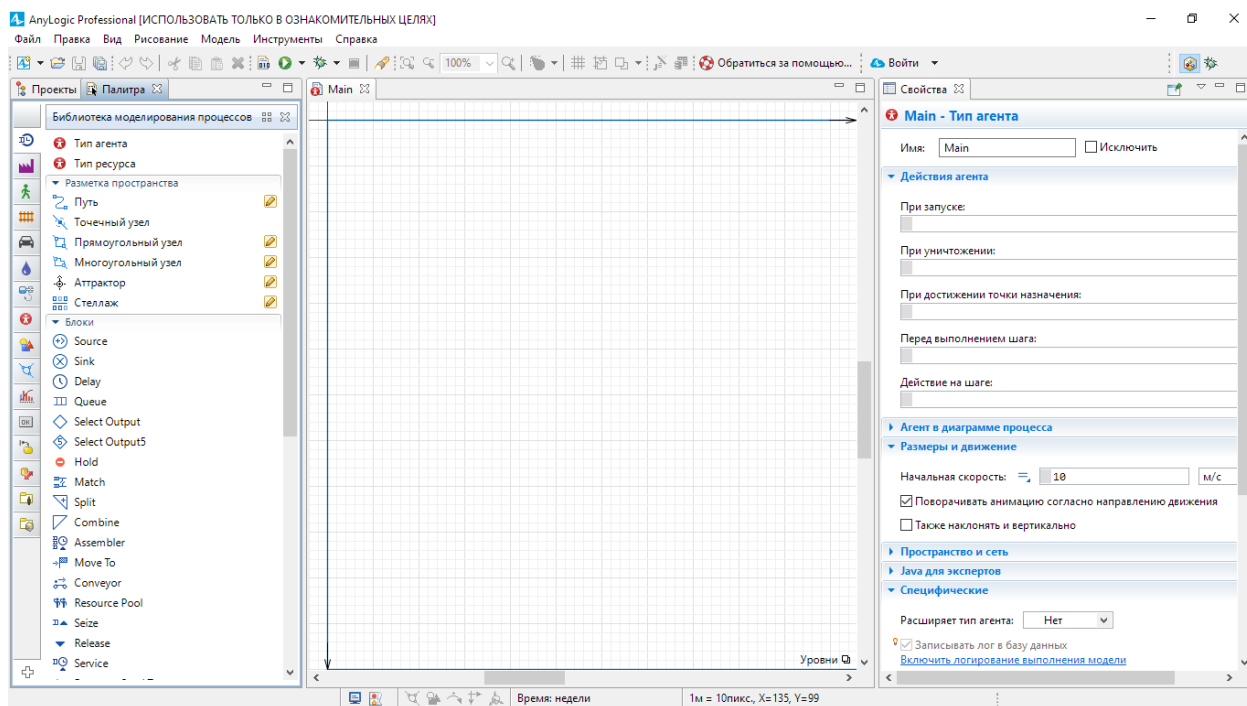
$$o_i^P = \frac{1}{h} \sum_{j=i-h}^i d_j^P - s_i^P.$$

Создание модели

Для выполнения данной лабораторной работы требуется создать модель в инструменте имитационного моделирования AnyLogic. Назовем нашу модель Dead_Game, т. к., хотя за основу взята BeerGame, мы против алкоголя, ведь алкоголь убивает. Назначим единицы модельного времени – недели.



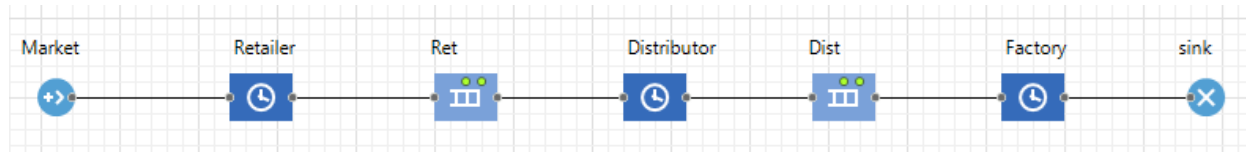
После создания модели откроется панель разработчика или рабочая область:



В левой части области находится панель Палитра, обеспечивающая легкую навигацию по элементам моделей.

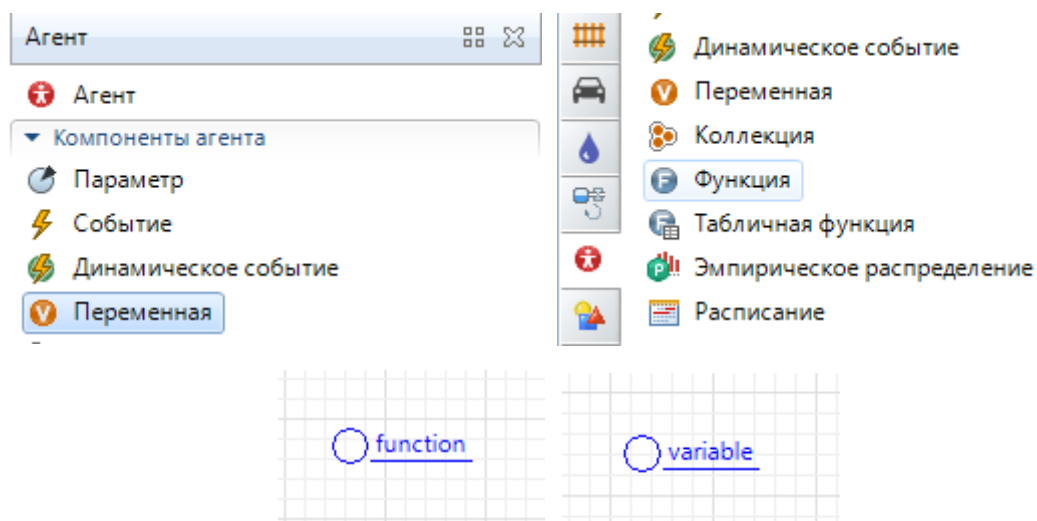
Создание диаграммы процессов

Наша модель представляет собой цепочку следующего вида:



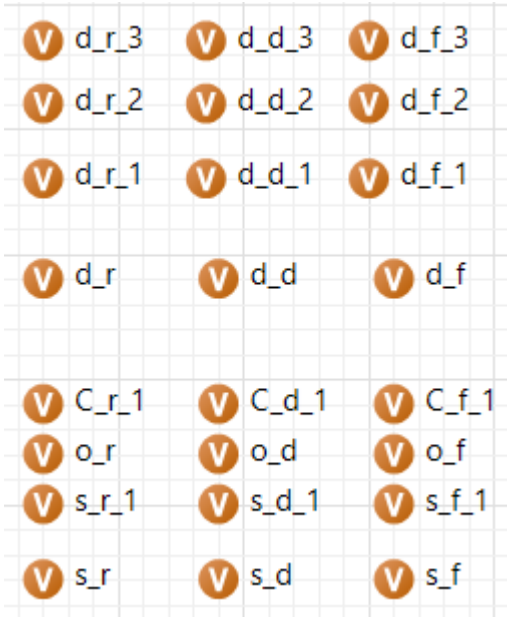
Наименование	Расшифровка
Market	Рынок
Retailer	Розничный торговец
Ret	Промежуточная очередь
Distributor	Дистрибьютор
Dist	Промежуточная очередь
Factory	Фабрика
Sink	Выход

Для реализации данной модели понадобятся функции и переменные, которые находятся в палитре Агент:

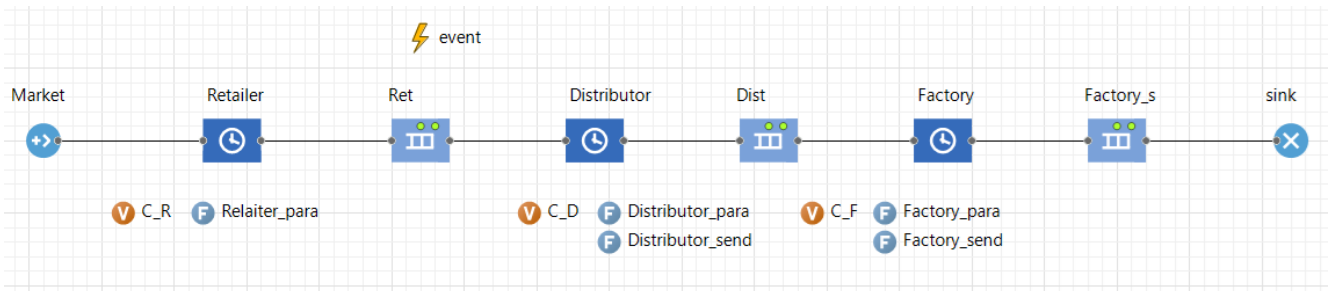


Согласно 2-ому алгоритму, объём заказа определяется через скользящее арифметическое среднее порядка h . Требуется для каждого участника цепи предусмотреть количество переменных, не меньшее $h = 3$ (по варианту – это максимальное значение h):

Все переменные:



Преобразуем модель следующим образом:



Для каждого участника предусмотрим ряд функций, учитывая, что:

Необходимо оценить объём поставок на основе анализа истории закупок и текущего состояния запасов. В этом случае уравнение управления запасами выглядят следующим образом:

$$s_i^p = s_{i-1}^p + P_i^p - d_i^p.$$

Смысл уравнения состоит в следующем: уровень запаса на конец периода i равен сумме уровня запаса на конец предшествующего периода ($i - 1$) и объема поступлений за вычетом спроса за период i [15].

i – номер периода (день, неделя, месяц);

$o_i^p, o_i^d, o_i^w, o_i^r$ – объём заказа производителя, дистрибьютора, оптового поставщика, розничного торговца, [шт];

$d_i^p, d_i^d, d_i^w, d_i^r$ – объём спроса для производителя, дистрибьютора, оптового поставщика, розничного торговца, [шт];

$s_i^p, s_i^d, s_i^w, s_i^r$ – объём остатков на складе у производителя, дистрибьютора, оптового поставщика, розничного торговца, [шт];

$P_i^p, P_i^d, P_i^w, P_i^r$ – объём поставок у производителя, дистрибьютора, оптового поставщика, розничного торговца, [шт];

$C_i^p, C_i^d, C_i^w, C_i^r$ – финансовый результат (прибыль или убыток) у производителя, дистрибьютора, оптового поставщика, розничного торговца, [д.е.];

l – срок поставки, кол-во временных периодов;

sdt_o – среднее квадратичное отклонение объёма заказов;

sdt_s – среднее квадратичное отклонение объёма остатков;

sdt_C – среднее квадратичное отклонение финансового результата;

$P_i^p, P_i^d, P_i^w, P_i^r, P_i^m$ – цена производства, цена продажи у производителя, дистрибьютора, оптового поставщика, розничного торговца, [д.е.];

t – стоимость хранения одной единицы товара на складе, [д.е.];

q – наценка при перекупке, %.

Данные, определяющие значения входных показателей:

h – количество используемых для вычислений заказов;

k – цена хранения одной единицы товара на складе, [д.е.].

Решением задачи будут оптимальные взаимодействия участников цепи поставок. Каждый из них на каждом временном промежутке выполняет следующие действия:

- принимает заказы от клиентов;
- получает товар от поставщиков;
- обновляет значения текущих показателей;
- отправляет товар клиентам дальше по цепи, посылает новый заказ поставщикам.

Выбор объемов заказов $O_i^p, O_i^d, O_i^w, O_i^r$ в каждом раунде является тем решением, которое принимают участники цепи.

Добавим в модель переменные прибыли, учитывая, что C_R, C_D, C_F – прибыль каждого из участников соответственно.

▼ C_R - Переменная

Имя:

☒ Отображать имя ☐ Исключить

Видимость: ☒ да

Тип: ▼

Начальное значение:

▼ C_D - Переменная

Имя:

☒ Отображать имя ☐ Исключить

Видимость: ☒ да

Тип: ▼

Начальное значение:

▼ C_F - Переменная

Имя:

☒ Отображать имя ☐ Исключить

Видимость: ☒ да

Тип: ▼

Начальное значение:

Функция для розничного торговца:

F Relaiter_para - Функция

Имя: ☒ Отображать имя

Видимость: ☒ да

☒ Действие (не возвращает ничего)

☐ Возвращает значение

▸ Аргументы

▾ Тело функции

```
d_r = Retailer.size();
d_r_1 = d_r;
d_r_2 = d_r_1;
d_r_3 = d_r_2;

if(d_r < s_r){
    C_r_1 = C_R;
    C_R += d_r * p_r - s_r * k - o_r * p_m;
}
else{
    C_r_1 = C_R;
    C_R += d_r * p_r - s_r * k - o_r * p_m;
    C_R -= (d_r - s_r_1) * (1 - q / 100) * p_r;
}
```

Данная функция проверяет товар на складе и высчитывает прибыль розничного торговца, согласно формулам, указанным ниже. C_{r_1} – прибыль на 1 итерацию в прошлое.

Каждую единицу времени (в рассматриваемой модели неделя) происходят поставки, продажи, закупки и производится выплата за хранение. Финансовый результат вычисляется по следующей формуле (на примере розничного торговца):

$$C_i^r = C_{i-1}^r + d_i^r \cdot p^m - s_i^r \cdot k - o_i^r \cdot p^r.$$

Если на складе не хватает товара, то штраф за перекупку определяется так:

$$C_i^r = C_i^r - (d_i^r - s_{i-1}^r) \cdot \left(1 + \frac{q}{100}\right) \cdot p^r$$

Функции для дистрибьютора:

Distributor_para - Функция

Имя: ☒ Отображать имя

Видимость: ☒ да

☒ Действие (не возвращает ничего)

☐ Возвращает значение

Аргументы

Тело функции

```
d_d = Distributor.size();
d_d_1 = d_d;
d_d_2 = d_d_1;
d_d_3 = d_d_2;

if(d_d < s_d){
    C_d_1 = C_D;
    C_D += d_d * p_d - s_d * k - o_d * p_m;
}
else{
    C_d_1 = C_D;
    C_D += d_d * p_d - s_d * k - o_d * p_m;
    C_D -= (d_d - s_d_1) * (1 - q / 100) * p_d;
}
```

Данная функция проверяет товар на складе и высчитывает прибыль дистрибьютора, согласно формулам, аналогичным приведенным выше формулам для розничного торговца. C_{d_1} – прибыль на 1 итерацию в прошлое. Она обрабатывается при поступлении заказа дистрибьютору.

Distributor_send - Функция

Имя: ☒ Отображать ил

Видимость: ☒ да

☒ Действие (не возвращает ничего)

☐ Возвращает значение

Аргументы

Тело функции

```
if(mod == 0){
    s_r_1 = s_r;
    s_r -= d_r;
    o_r = d_r - s_r;
    if(o_r < 0){
        o_r = 0;
    }
    s_r += o_r;
}
else{
    s_r -= d_r;
    o_r = ((d_r - s_r) + (d_r_1 - s_r) +
           (d_r_2 - s_r) + (d_r_3 - s_r)) / h;
    if(o_r < 0){
        o_r = 0;
    }
    s_r += o_r;
}
```

Данная функция пополняет склад ретейлера. Выполняется при выходе заказа из дистрибьютора.

Функции для фабрики:

F Factory_para - Функция

Имя: ☒ Отображать имя

Видимость: ☒ да

☒ Действие (не возвращает ничего)

☐ Возвращает значение

▸ Аргументы

▼ Тело функции

```
d_f = Factory.size();
d_f_1 = d_f;
d_f_2 = d_f_1;
d_f_3 = d_f_2;

if(mod == 0){
    s_f_1 = s_f;
    s_f -= d_f;
    o_f = d_f - s_f;
    if(o_f < 0){
        o_f = 0;
    }
    s_f += o_f;
}
else{
    s_f -= d_f;
    o_f = ((d_f - s_f) + (d_f_1 - s_f) +
        (d_f_2 - s_f) + (d_f_3 - s_f)) / h;
    if(o_f < 0){
        o_f = 0;
    }
    s_f += o_f;
}

if(d_f < s_f){
    C_f_1 = C_F;
    C_F += d_f * p_f - s_f * k - o_f * p_m;
}
else{
    C_f_1 = C_F;
    C_F += d_f * p_f - s_f * k - o_f * p_m;
    C_F -= (d_f - s_f_1) * (1 - q / 100) * p_f;
}
```

Данная функция рассчитывает прибыль фабрики, а также поступление на сырьё на склад. Обработывается при входе заказа на фабрику.

F Factory_send - Функция

Имя: ☒ Отображать им

Видимость: ☒ да

☒ Действие (не возвращает ничего)

☐ Возвращает значение

Аргументы

Тело функции

```
if(mod == 0){
    s_d_1 = s_d;
    s_d -= d_d;
    o_d = d_d - s_d;
    if(o_d < 0){
        o_d = 0;
    }
    s_d += o_d;
}
else{
    s_d -= d_d;
    o_d = ((d_d - s_d) + (d_d_1 - s_d) +
           (d_d_2 - s_d) + (d_d_3 - s_d)) / h;
    if(o_d < 0){
        o_d = 0;
    }
    s_d += o_d;
}
```

Данная функция пополняет склад дистрибьютора. Обработывается на выходе заказ из фабрики.

Обработка функций пополнения складов, т. е. отправки заказа заказчику выставлена на выход следующего звена в цепи поставок для того, чтобы имитировать время, необходимое на поставку заказа.

Mod – переменная режима, от которой зависит используемый алгоритм расчёта заказа (0 – первый; 1 – второй с $h = 3$).

Алгоритм расчёта заказа



mod

Первый

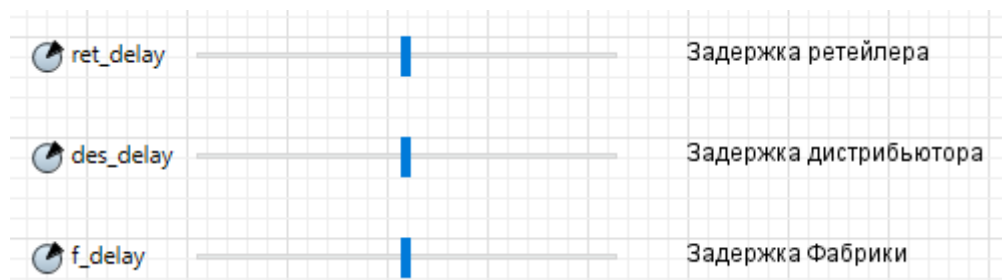


Второй с $h = 3$

У заказа есть стоимость, которую тоже нужно регулировать. Установим соответствующие параметры для всех участников:



Также понадобятся параметры задержки:



Событие event для Market (рынка) о прибытии заказа изменяется согласно стандартному дискретному распределению:

⚡ event - Событие

Имя: ☒ Отображать имя

☐ Исключить

Видимости: ☒ да

Тип события:

Режим:

☒ Использовать модельное время ☐ Использовать календарные даты

Время первого срабатывания (абс.):

Время срабатывания

Период:

☒ Записывать лог в базу данных

[Включить логирование выполнения модели](#)

▼ Действие

```
Market.inject(uniform_discr(0, 200));
```

Свяжем параметры и участников.

Рынок:

⚡ Market - Source

Имя: ☒ Отображать имя

☐ Исключить

Прибывают согласно:

Местоположение прибытия:

Розничный торговец:

▼ Действия

При входе:

При подходе к выходу:

При выходе:

При извлечении:

⌚ Retailer - Delay

Имя: ☒ Отображать имя

☐ Исключить

Тип задержки: ☐ ● Определенное время
☐ ○ До вызова функции stopDelay()

Время задержки:

Максимальная вместимость: ☐

Место агентов:  

▼ Специфические

Выталкивать агентов: ☐

Вернуть агента в исходную точку: ☒

Включить сбор статистики: ☒

Дистрибьютор:

⌚ Distributor - Delay


Имя: ☒ Отображать имя

☐ Исключить

Тип задержки: ☐ ● Определенное время
☐ ○ До вызова функции stopDelay()

Время задержки:

Максимальная вместимость: ☒

Место агентов:  

▼ Специфические

Выталкивать агентов: ☐

Вернуть агента в исходную точку: ☒

Включить сбор статистики: ☒

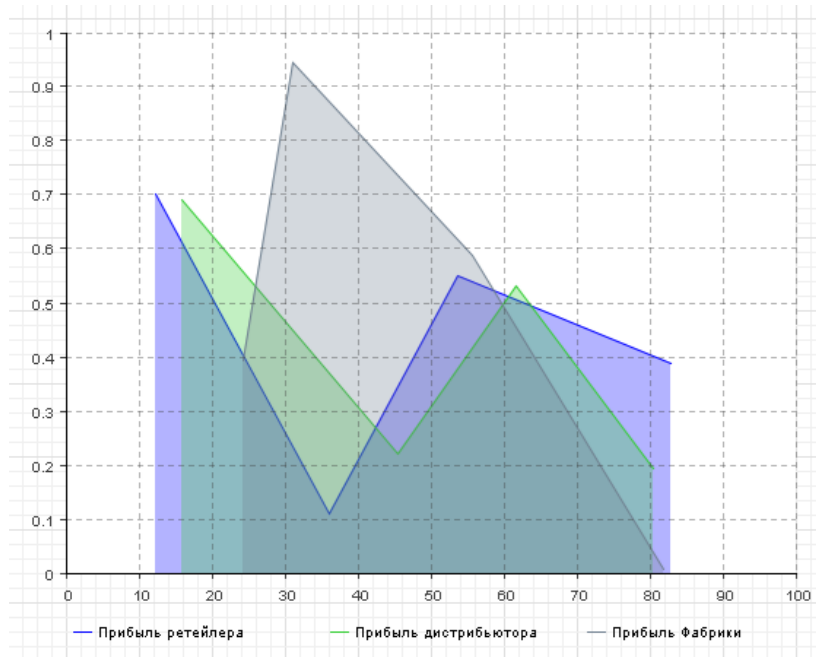
▼ Действия	
При входе:	<code>Distributor_para();</code>
При подходе к выходу:	
При выходе:	<code>Distributor_send();</code>
При извлечении:	

Фабрика:

🕒 Factory - Delay	
Имя:	<input type="text" value="Factory"/> <input checked="" type="checkbox"/> Отображать имя
<input type="checkbox"/> Исключить	
Тип задержки:	<input checked="" type="radio"/> Определенное время <input type="radio"/> До вызова функции stopDelay()
Время задержки:	<input type="text" value="f_delay"/> <input type="text" value="недели"/>
Максимальная вместимость:	<input checked="" type="checkbox"/>
Место агентов:	<input type="text" value="path2"/> <input type="button" value="🗑️"/> <input type="button" value="📄"/>
▼ Специфические	
Выталкивать агентов:	<input type="checkbox"/>
Вернуть агента в исходную точку:	<input checked="" type="checkbox"/>
Включить сбор статистики:	<input type="checkbox"/>
▼ Действия	
При входе:	<code>Factory_para();</code>
При подходе к выходу:	
При выходе:	<code>Factory_send();</code>
При извлечении:	

График прибыли

Добавим график для отображения прибыли ретейлера, дистрибьютора и фабрики:



Данные

☒ Значение ☐ Набор данных

Заголовок: Прибыль ретейлера

Значение: C_R

Стиль маркера: —

Толщина линии: — 1 pt

Цвет: blue

☒ Значение ☐ Набор данных

Заголовок: Прибыль дистрибьютора

Значение: C_D

Стиль маркера: —

Толщина линии: — 1 pt

Цвет: limeGreen

☒ Значение ☐ Набор данных

Заголовок: Прибыль Фабрики

Значение: C_F

Стиль маркера: —

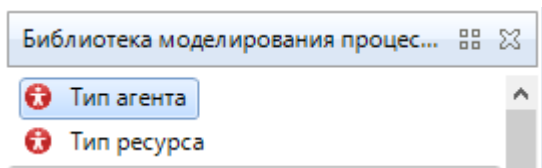
Толщина линии: — 1 pt

Цвет: slateGray



Анимация

Для более понятного отображения запросов выберем новый тип анимации через создание своего типа агента:

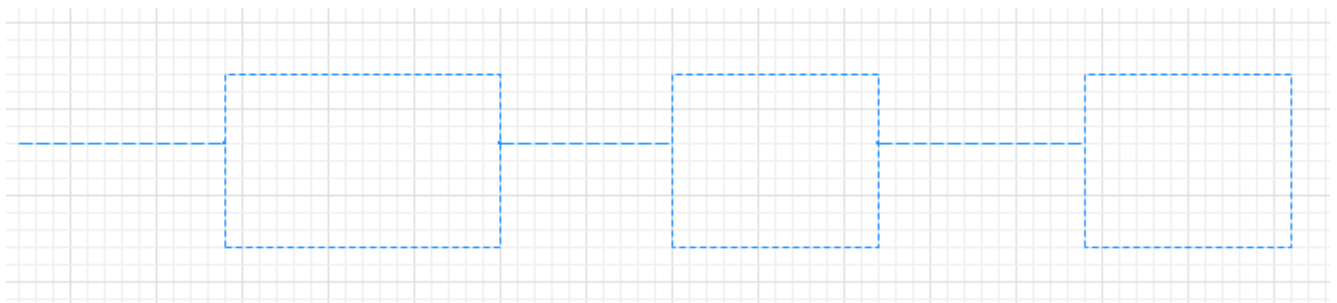


Отообразим товар в виде коробки:

Создадим области отображения анимации:



Расположим элементы следующим образом:



Свяжем участников и пути:

⌚ Retailer - Delay

Имя:

☒ Отображать имя ☐ Исключить

Тип задержки: ☒ Определенное время
☐ До вызова функции stopDelay()

Время задержки:

Максимальная вместимость: ☒

Место агентов:

Distributor - Delay

Имя:

Distributor

☒ Отображать имя

☐ Исключить

Тип задержки:

☒ Определенное время

☐ До вызова функции stopDelay()

Время задержки:

des_delay

недели

Максимальная вместимость:

☒

Место агентов:

path1

Factory - Delay

Имя:

Factory

☒ Отображать имя

☐ Исключить

Тип задержки:

☒ Определенное время

☐ До вызова функции stopDelay()

Время задержки:

f_delay

недели

Максимальная вместимость:

☒

Место агентов:

path2

В source (рынке) укажем нового агента Order, чтобы на вход подавались запросы товара:

Market - Source

Имя:

Market

☐ Исключить

Прибывают согласно:

Вызовам функл

Местоположение прибытия:

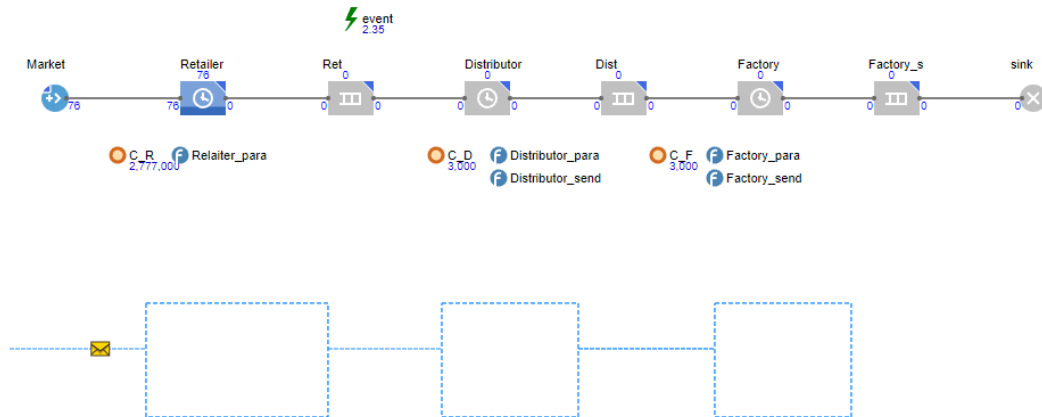
Не задано

Агент

Новый агент:

Order

Проверка работоспособности:



Проведём исследование прибыли в зависимости от различных параметров системы:

На начало исследования параметры следующие:

Стоимость производства: 250 у.е.;

Стоимость продажи у ретейлера: 1000 у.е.;

Стоимость продажи у дистрибьютора: 750 у.е.;

Стоимость продажи у фабрики: 500 у.е.;

Цена при перекупке товара: 1200 у.е. (важно отметить, что этот параметр общий для всех и на начало мы используем цену, которая превышает цену продажи у ретейлера, в дальнейшем мы изменим этот параметр);

Стоимость хранения на складе: 200 у.е.;

Задержки:

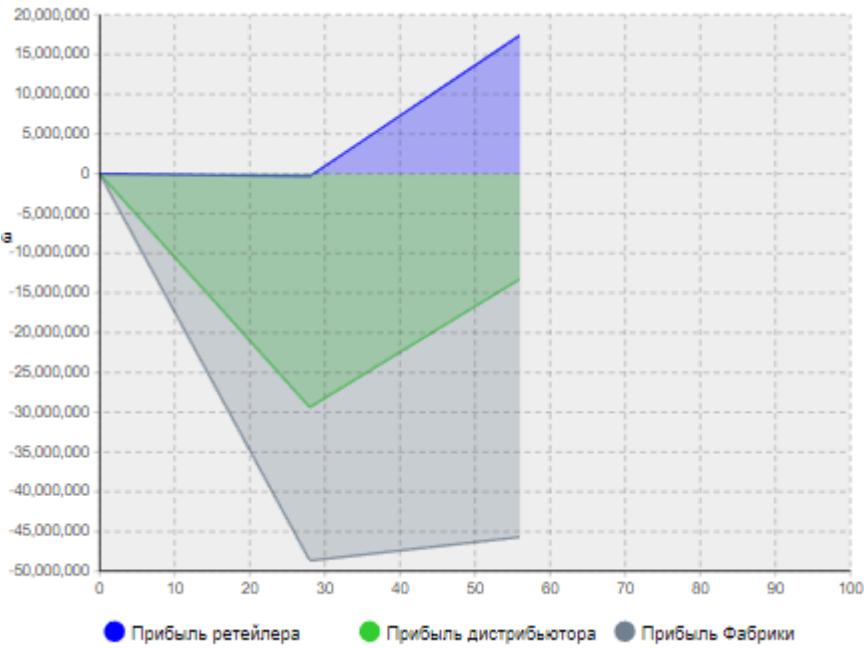
Ретейлер: 1 неделя,

Дистрибьютор: 1 неделя,

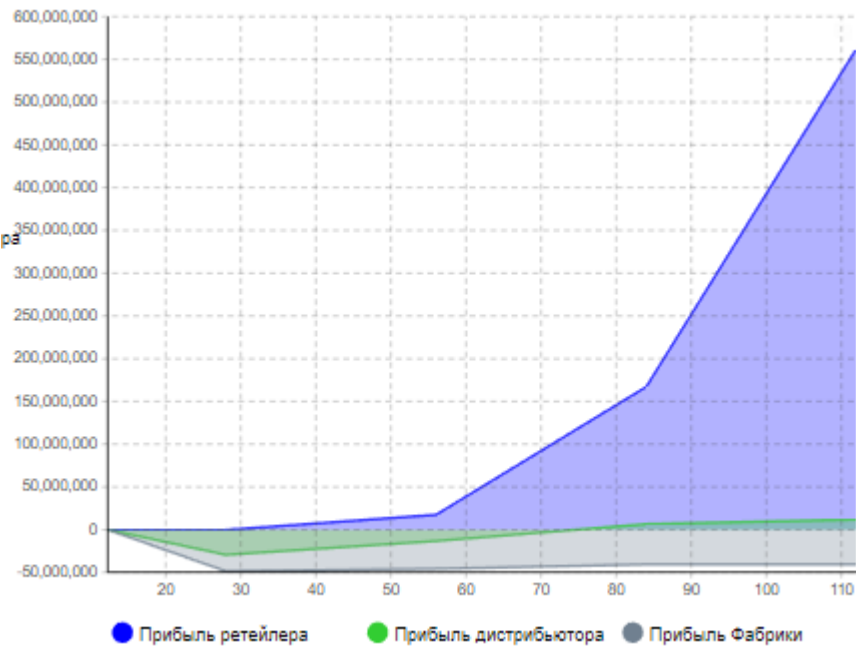
Фабрика: 1 неделя.

Все тесты проводятся для первого алгоритма.

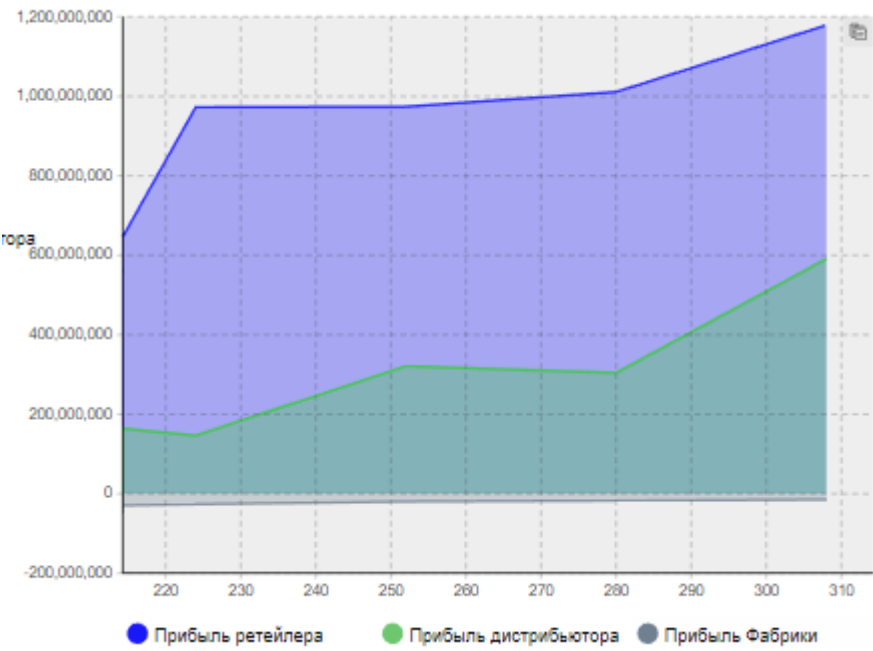
Как можно видеть по графику, сначала все участники терпят убытки:



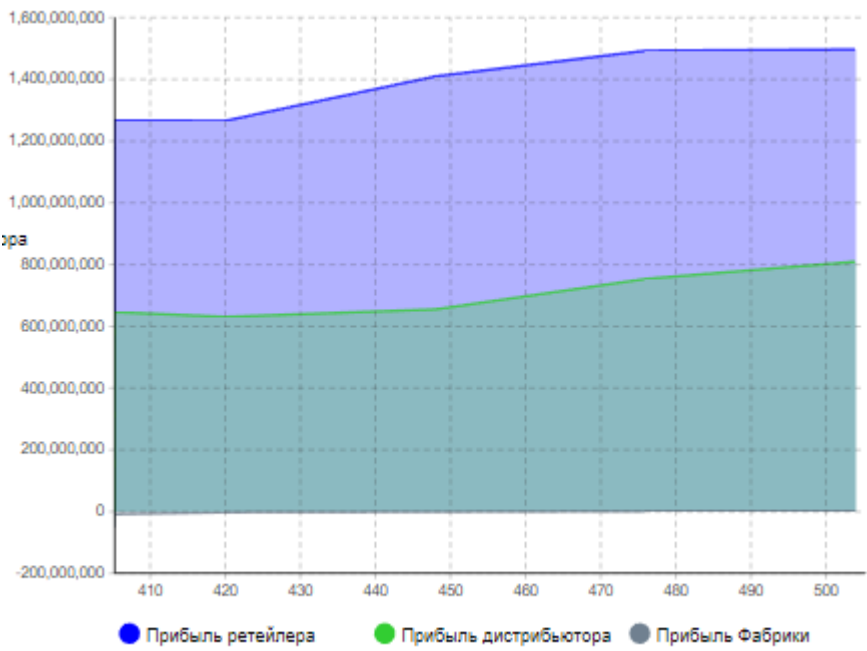
Однако, постепенно прибыль уходит в плюс. Наибольшую прибыль получает ретейлер, как конечное звено в цепи поставок, перед потребителем:



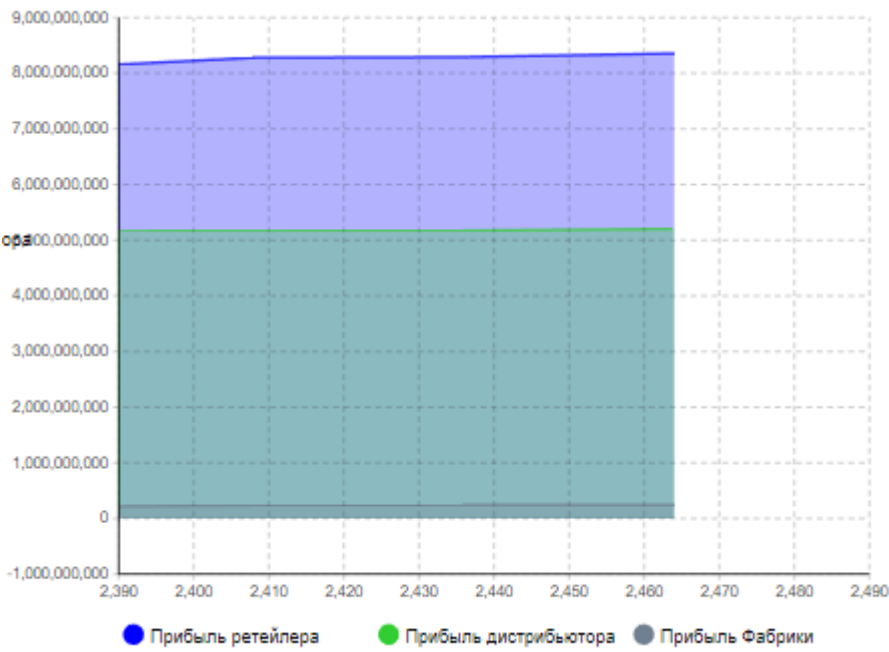
Медленнее всего растёт прибыль фабрики:



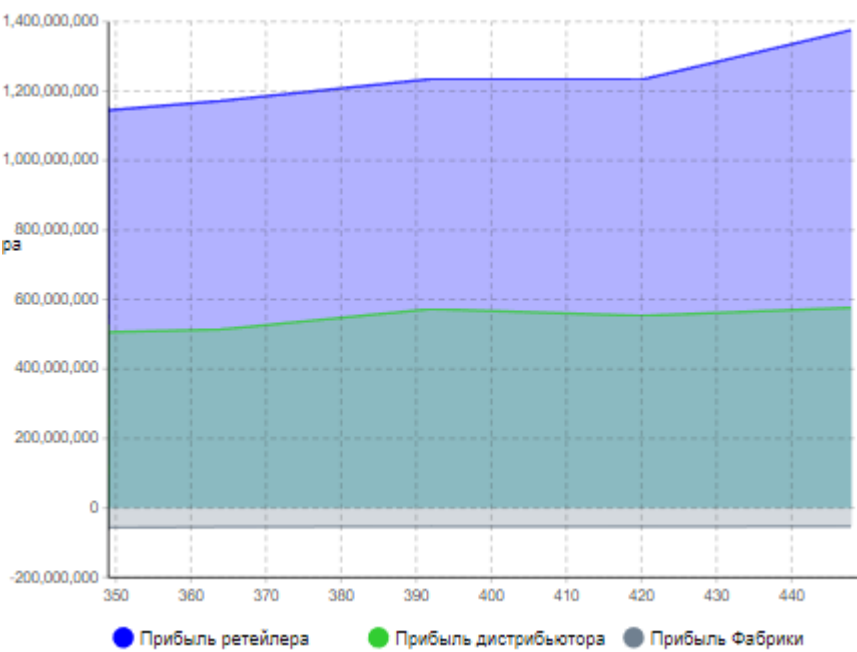
Как можно заметить, прибыль растёт скачкообразно:



Со временем рост идёт более стабильно:

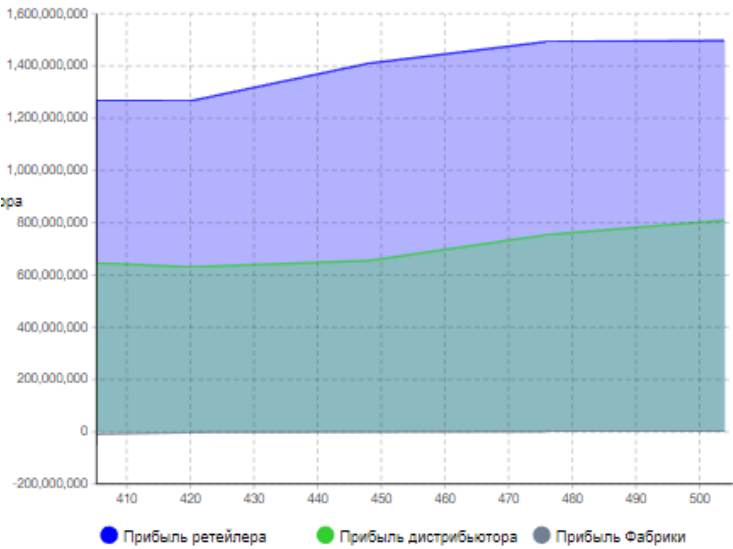


Изменим стоимость хранения на складе, сделав её равной стоимости производства товара, т.е. 250 у.е.:

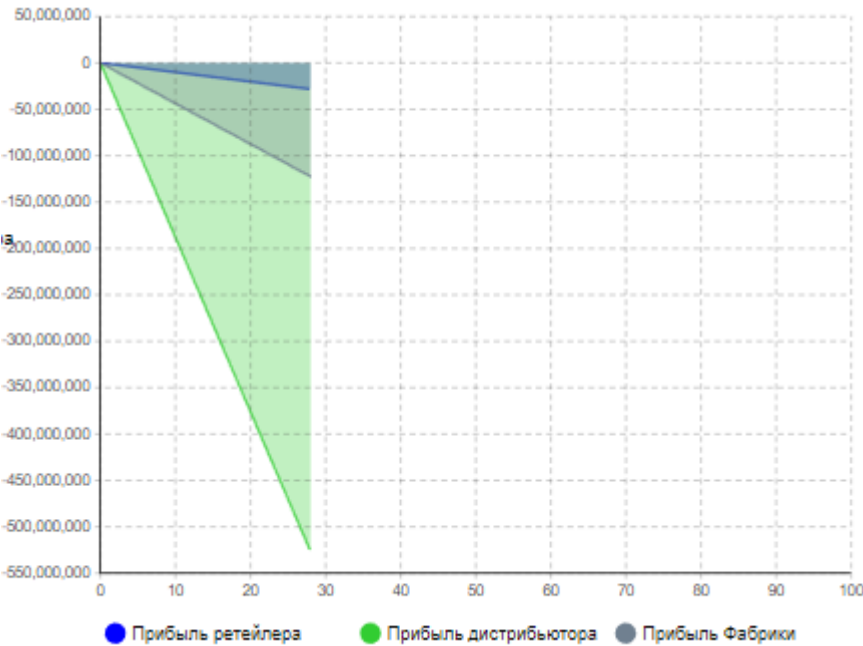


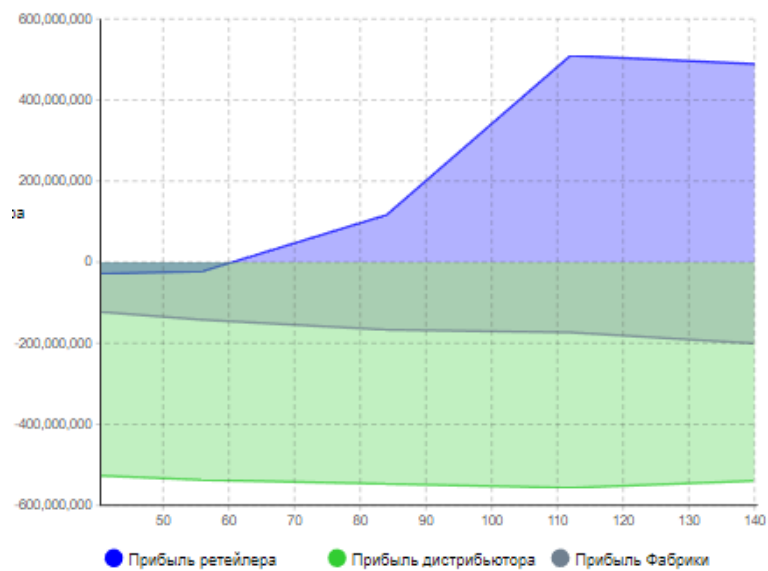
Можно заметить, что на рост прибыл подобное изменение не особенно повлияло:

График к 400-тым неделям при стоимости хранения в 200 у. е.:

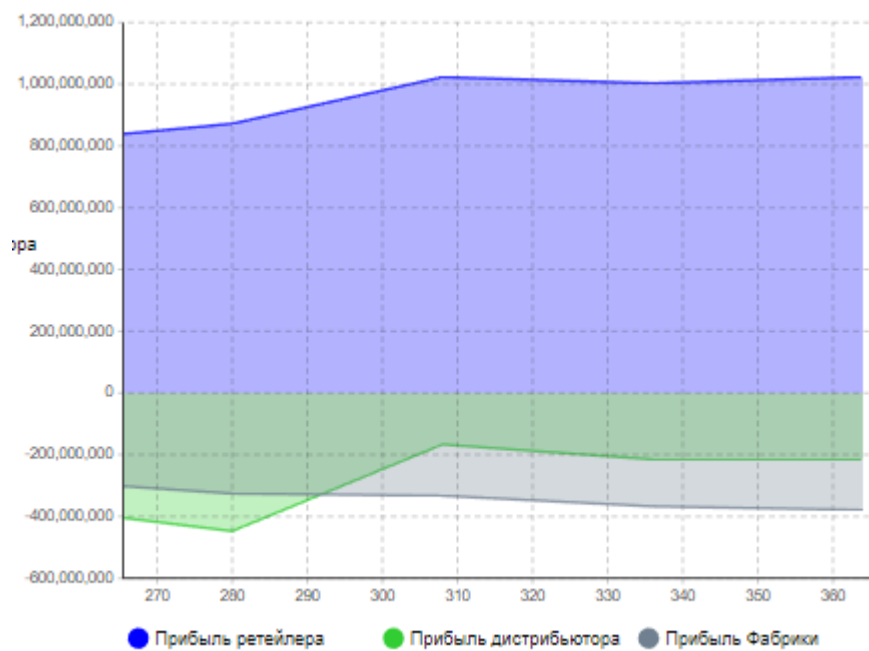


Прибыль начинает падать при приближении стоимости хранения товара к 500 у.е.:

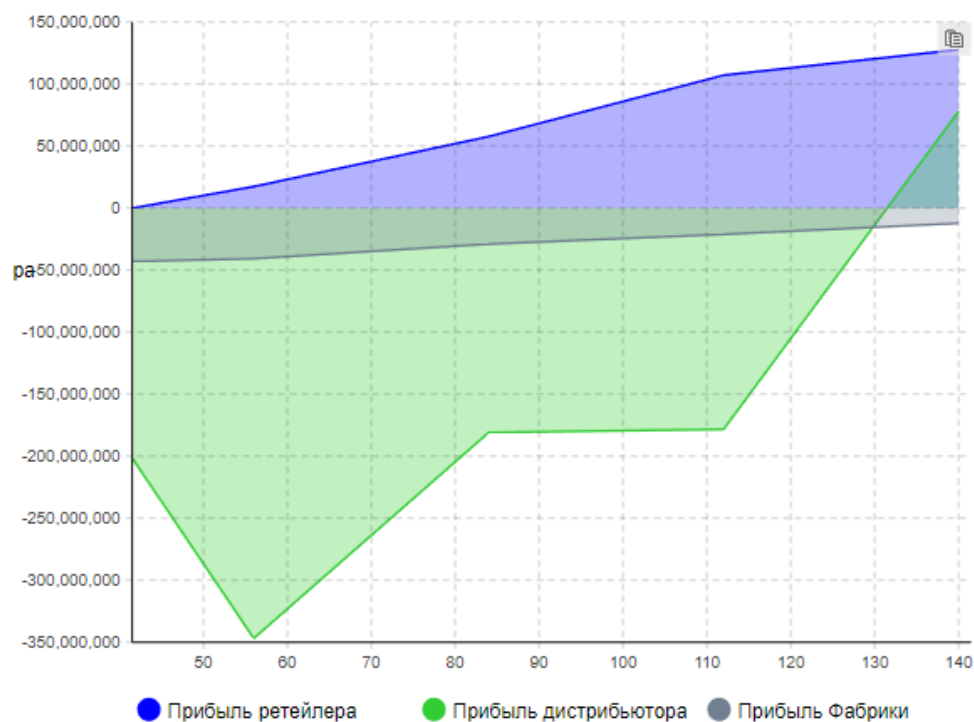




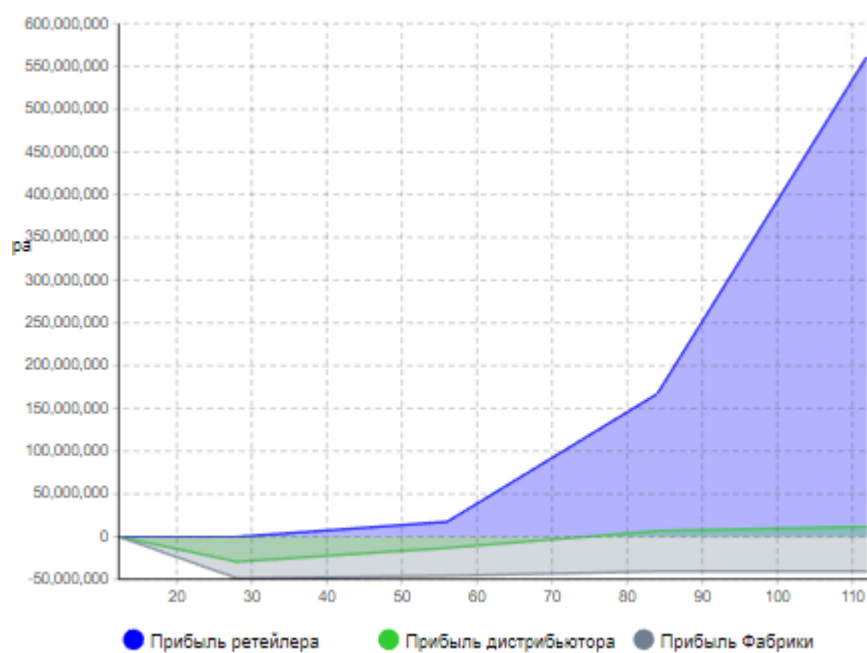
В основном от повышения стоимости хранения товара страдают фабрика и дистрибьютор, так как их цена продажи товара ниже стоимости продажи у ретейлера.



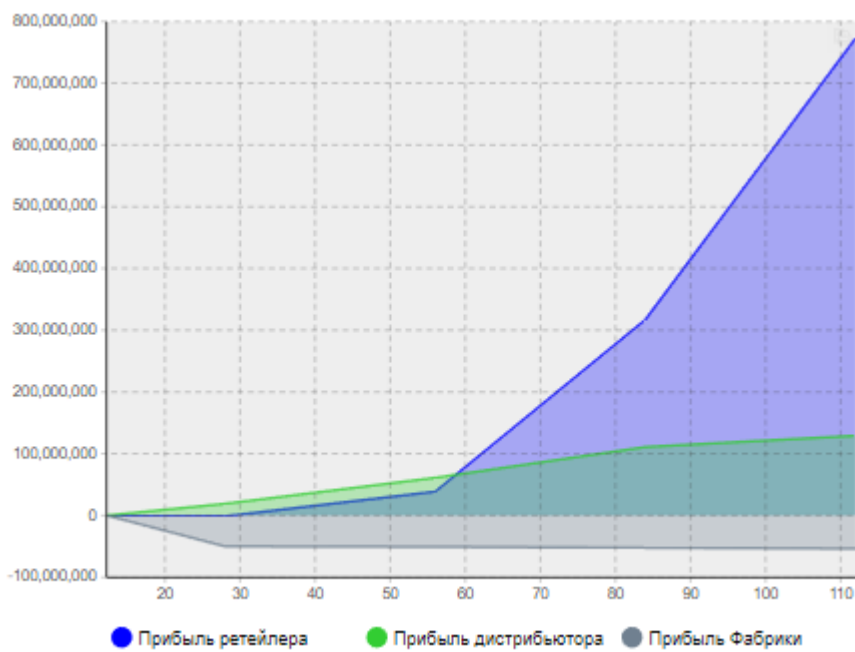
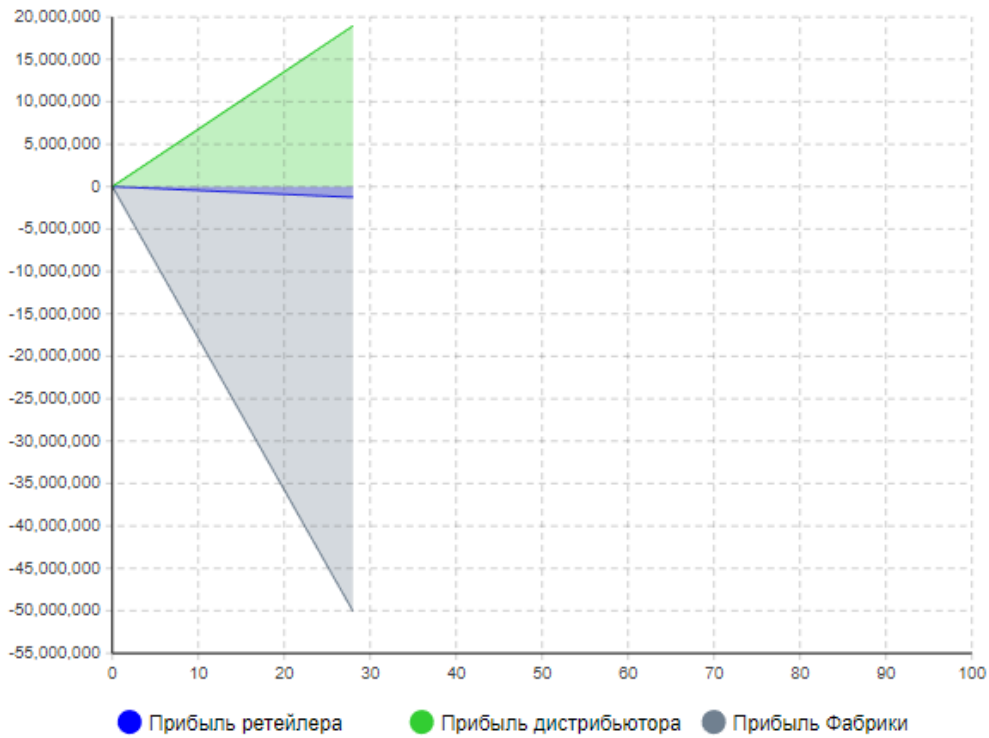
Увеличим задержку поставок у дистрибьютора до 2 недель, а у фабрики до 4:



Как видно по графику, задержка в поставках ведёт к мгновенному изменению прибылей в меньшую сторону. Для сравнения ниже представлен график для задержки в одну неделю у всех членов цепи поставок:



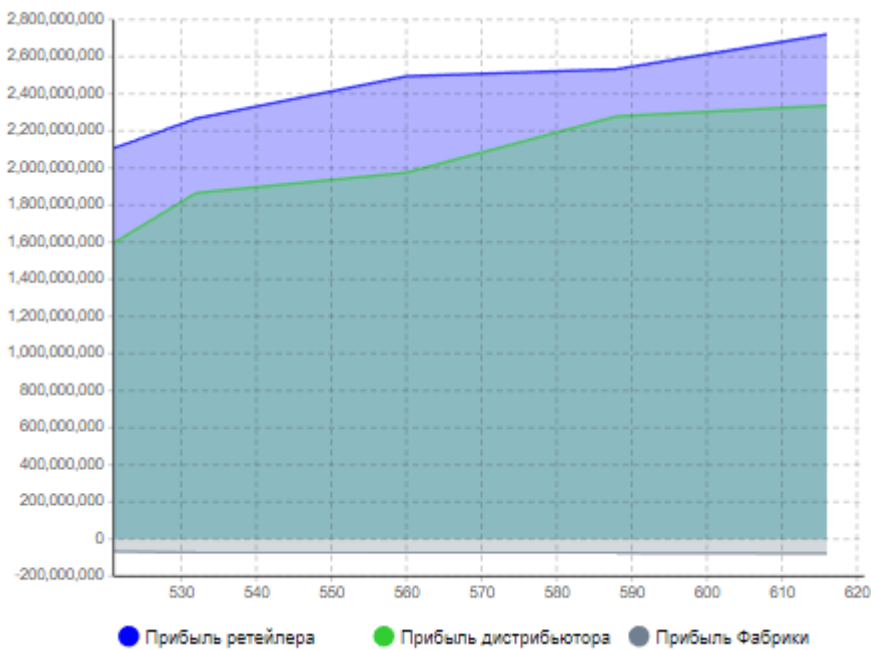
Проверим как изменяется прибыль при использовании второго алгоритма:



Как можно заметить, прибыль при втором алгоритме растёт заметно быстрее.

Следовательно можно сделать вывод, что второй алгоритм предпочтительнее. Однако, можно, заметить, что из-за того, что наценка при перекупке на 700 у. е. больше стоимости продажи

товара фабрикой, второй алгоритм приносит ей только убытки:



Исходя из того, как меняется прибыль при изменении размера штрафа при перекупке, можно сделать вывод, что данный параметр, не должен превышать стоимости продажи более чем в два раза.

Кроме того, если стоимость производства составляет половину от стоимости продажи, окупить производство быстро не получится. На всех графиках видно, что фабрика дольше всех находится в в минусе по прибыли. Для второго алгоритма окупаемости у фабрики нет вообще.

Изменим стоимость производства с 250 у. е. до 150 у. е.:

Как можно заметить, фабрика начала получать прибыль в гораздо больших объёмах:

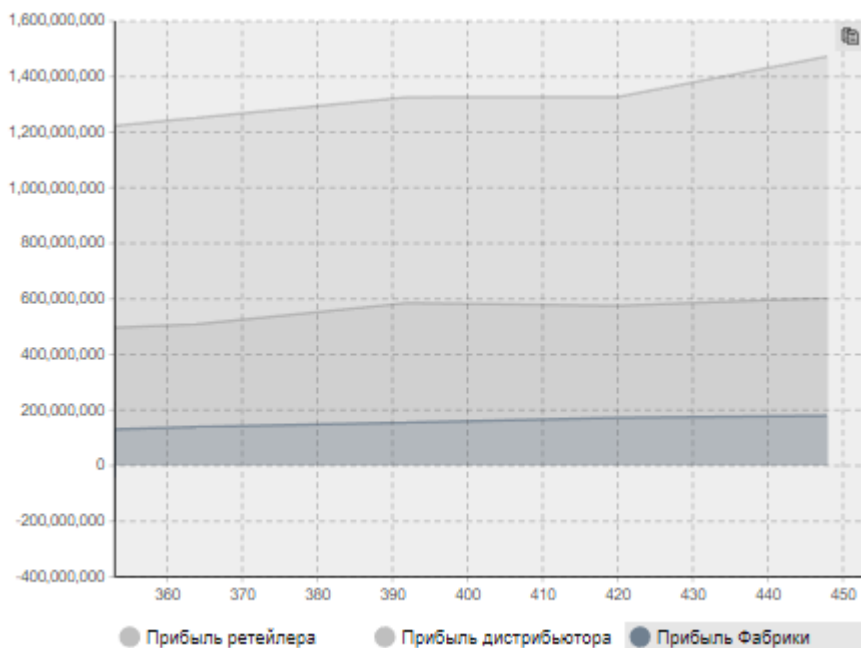
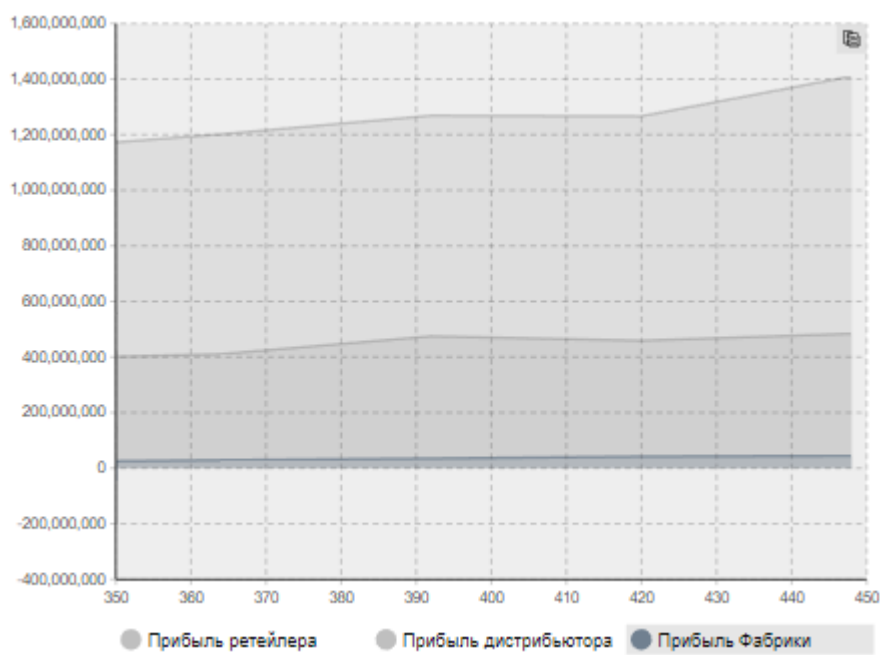
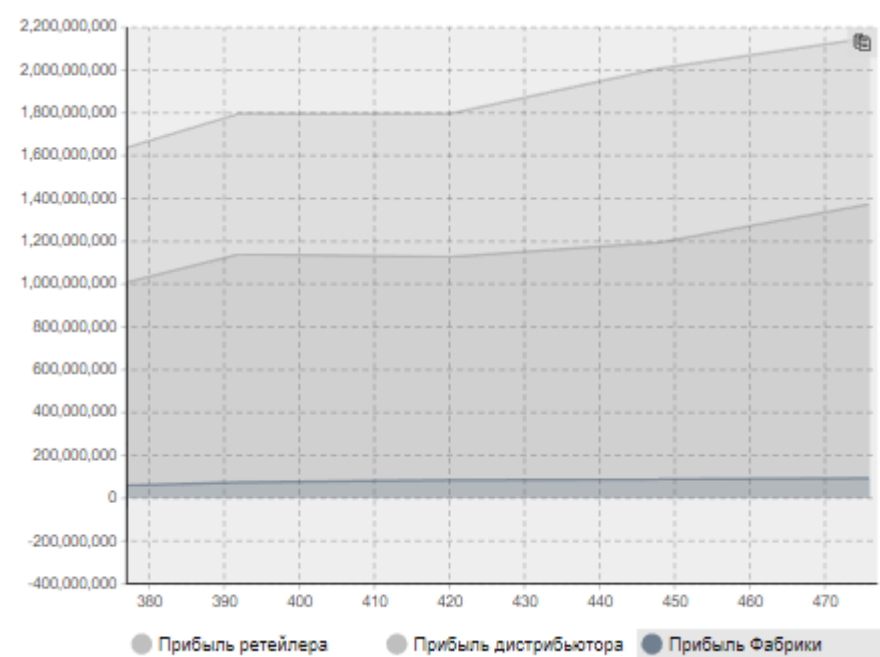


График для сравнения (стоимость производства составляет 250 у. е.):

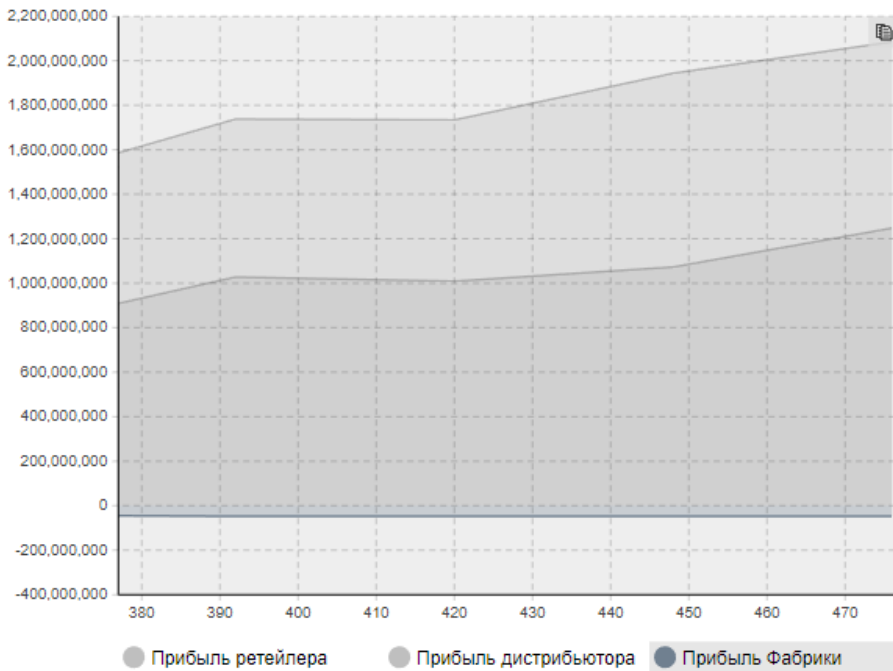


Схожие графики для второго алгоритма:

Стоимость производства 150 у. е.:



Стоимость производства 250 у. е.:



3. Выводы

В ходе проведённой работы нами была построена модель цепи поставок на основе BeerGame. Проведя исследование работы модели в зависимости от различных параметров системы, можно сделать вывод, что прибыль точнее и реалистичнее рассчитывается по второму алгоритму, так как он учитывает несколько прошлых состояний системы, а не только предыдущее.