



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

Кафедра вычислительной техники

ИТОГОВЫЙ ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ

По дисциплине

«Многоагентное моделирование»

(наименование дисциплины)

Студент группы

ИКБО-04-20

(учебная группа)

Хан Анастасия Александровна

(Фамилия Имя Отчество)

(подпись студента)

Руководитель

Тихвинский В.И.

(Должность, звание, ученая степень)

(подпись руководителя)

Работа представлена к защите « ____ » _____ 2022 г.

Допущен к защите « ____ » _____ 2022 г.

Москва 2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 ПРАКТИЧЕСКАЯ РАБОТА №1	6
1.1 Постановка задачи	6
1.2 Создание популяций	6
1.3 Диаграмма состояний	7
1.4 Гистограмма.....	10
1.5 Элемент управления	13
1.6 Итоговый запуск	14
2 ПРАКТИЧЕСКАЯ РАБОТА №2	15
2.1 Постановка задачи	15
2.2 Этап разгрузки паллетов	15
2.3 Этап управления фурой.....	16
2.4 Камера	17
3 ПРАКТИЧЕСКАЯ РАБОТА №3	19
3.1 Постановка задачи	19
3.2 Этап создания диаграммы потоков и накопителей	19
3.3 Этап добавления графика для визуальной динамики процесса.....	21
3.4 Этап экспериментального варьирования параметров.....	22
4 ПРАКТИЧЕСКАЯ РАБОТА №4	24
4.1 Постановка задачи	24
4.2 Создание популяции	24
4.3 Диаграмма состояний	25
4.4 Гистограмма.....	27

4.5 Элемент управления	28
4.6 Итоговый запуск	30
5 ПРАКТИЧЕСКАЯ РАБОТА №5	31
5.1 Постановка задачи	31
5.2 Этап создания пути для клиента.....	32
5.3 Этап загрузки аттракторов	33
5.4 Запуск программы.....	34
6 ПРАКТИЧЕСКАЯ РАБОТА №6	36
6.1 Постановка задачи	36
6.2 Этап создания диаграммы потоков и накопителей	36
6.3 Этап создания диаграммы состояний	38
6.4 Этап добавления графика для визуальной динамики процесса	39
6.5 Запуск проекта.....	40
6.6 Этап экспериментального варьирования параметров	40
7 ПРАКТИЧЕСКАЯ РАБОТА №7	42
7.1 Постановка задачи	42
7.2 Описание процесса создания проекта и его запуск.....	42
8 ПРАКТИЧЕСКАЯ РАБОТА №8	45
8.1 Постановка задачи	45
8.2 Описание процесса создания проекта и его реализация.....	45
9 ПРАКТИЧЕСКАЯ РАБОТА №9	49
9.1 Постановка задачи	49
9.2 Описание процесса создания проекта и его запуск.....	49
ЗАКЛЮЧЕНИЕ	52
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	53

ПРИЛОЖЕНИЯ.....	54
-----------------	----

ВВЕДЕНИЕ

На сегодняшний день под мультиагентными (многоагентными) технологиями понимают, как технологии разработки и использования мультиагентных систем (МАС), так и мультиагентное управление (МАУ). Задачи управления и распределенного взаимодействия в сетях динамических систем привлекают в последнее десятилетие внимание все большего числа исследователей. Во многом это объясняется широким применением мультиагентных систем в разных областях, включая автоматическую подстройку параметров нейронных сетей распознавания, управление формациями, распределенные сенсорные сети, управление перегрузкой в сетях связи, взаимодействие групп беспилотных летательных аппаратов (БПЛА), относительное выравнивание групп спутников, управление движением групп мобильных роботов, синхронизации в энергосистемах и др.

На практике все чаще используются распределенные системы, выполняющие определенные действия параллельно, для которых актуальна задача разделения пакета заданий между несколькими вычислительными потоками (устройствами). Подобные задачи возникают не только в вычислительных сетях, но также и в производственных сетях, сетях обслуживания, транспортных, логистических сетях и др.

1 ПРАКТИЧЕСКАЯ РАБОТА №1

1.1 Постановка задачи

Необходимо создать агентную модель, которая поможет изучить процесс вывода нового продукта на рынок.

- Мы рассмотрим относительно небольшой потребительский рынок численностью в 500 человек. С точки зрения реализации модели каждый потребитель будет являться агентом.
- Поскольку мы рассматриваем процесс вывода на рынок нового продукта, то изначально никто этим продуктом не пользуется.
- Люди начнут покупать продукт под влиянием рекламы.
- После этого начального этапа куда более сильное влияние на продажи будет оказывать общение людей друг с другом, рекомендации и положительные отзывы потребителей продукта, побуждающие других на его приобретение.
- Через некоторое время продукт, купленный людьми, начнет портиться, т.к. истечет срок его годности, и люди вновь начнут покупать продукт.
- Мы смоделируем также людей, возжелавших заказать продукт, а также тех людей, которые будут отменять заказ продукта.

1.2 Создание популяций

Мы создали проект и установили в нем все необходимые настройки. Создали агента и его популяцию, Рисунок 1.2.1.

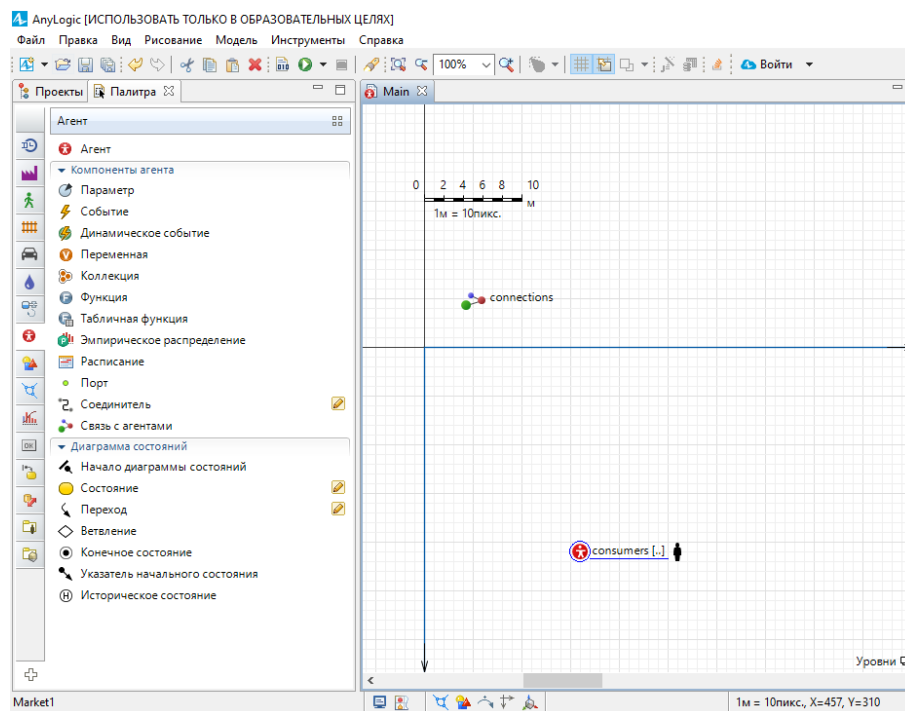


Рис 1.2.1 - Агент Consumer

Выбрали кнопку для компиляции, и запустили проект. В результате, на экране появилось окно запуска. Визуальная модель проекта представлена на Рисунке 1.2.2.

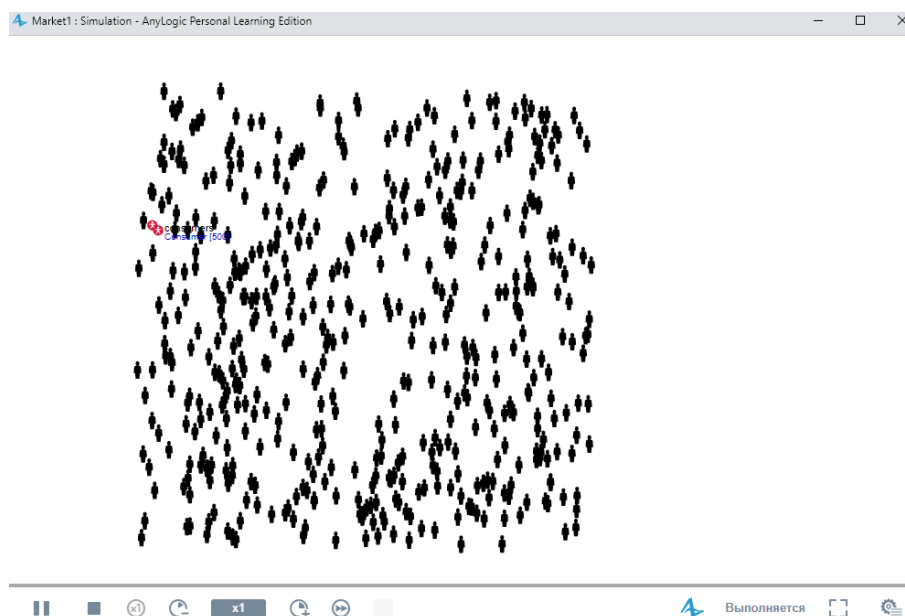


Рис 1.2.2 – Запуск проекта

1.3 Диаграмма состояний

Этап 2 – Задание поведения потребителей.

Двойным кликом выберем Consumer на дереве проекта и этим откроем его окно редактора. Выберем палитру, тип элемента диаграммы, и перетащим элемент Начало диаграммы в окно редактора. В результате выполненных действий будет построена диаграмма на дереве проекта Consumer, Рисунок 1.3.1.

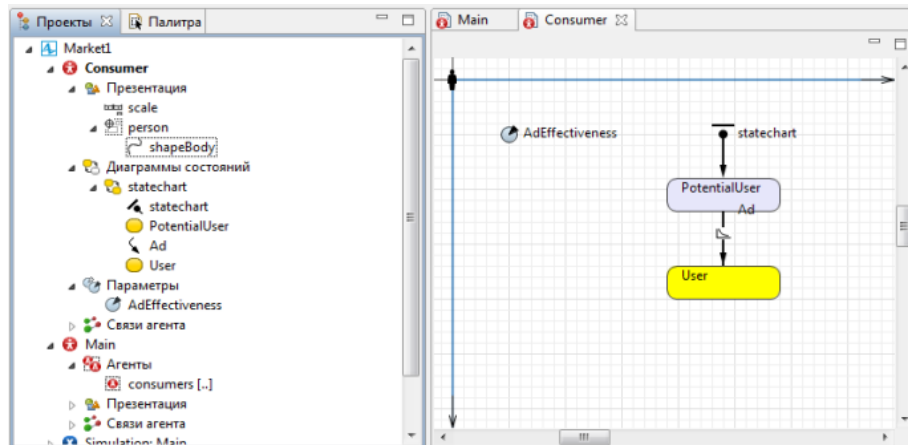


Рис 1.3.1 – Диаграмма состояний на дереве проекта Consumer

Этап 4 – Добавление эффекта рекомендаций.

Добавление параметров:

- ContactRate (определяет интенсивность контактов);
- AdoptionFraction (вероятность приобретения продукта в результате общения с пользователем этого продукта).

Добавление внутреннего перехода на диаграмму, с целью посылки сообщения, Рисунок 1.3.2.

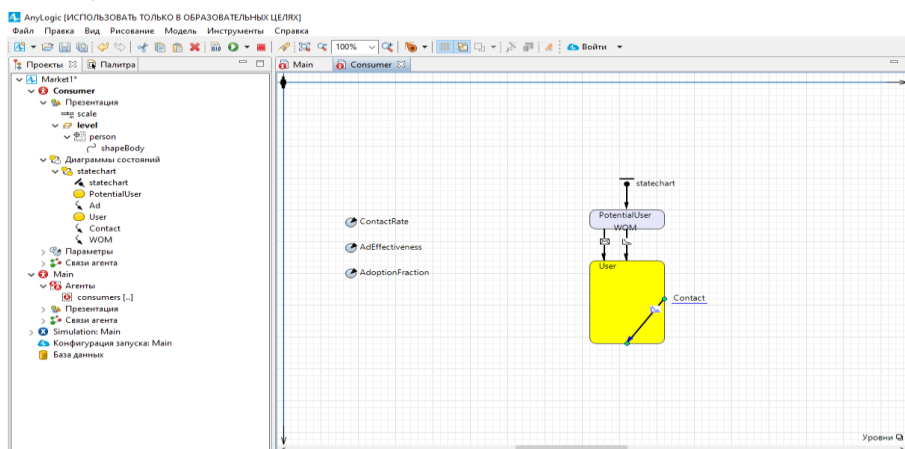


Рис 1.3.2 –Диаграмма после добавления всех параметров и внутреннего перехода

Этап 5 – Учет повторных продаж продукта.

На этом этапе мы делаем 2 основных действия на нашей диаграмме:

- Определяем срок службы продукта (параметр DiscardTime);
- Задаем обратный переход в потенциального покупателя (переход из состояния User в состояние PotentialUser).

Все этапы представлены на Рисунке 1.3.3.

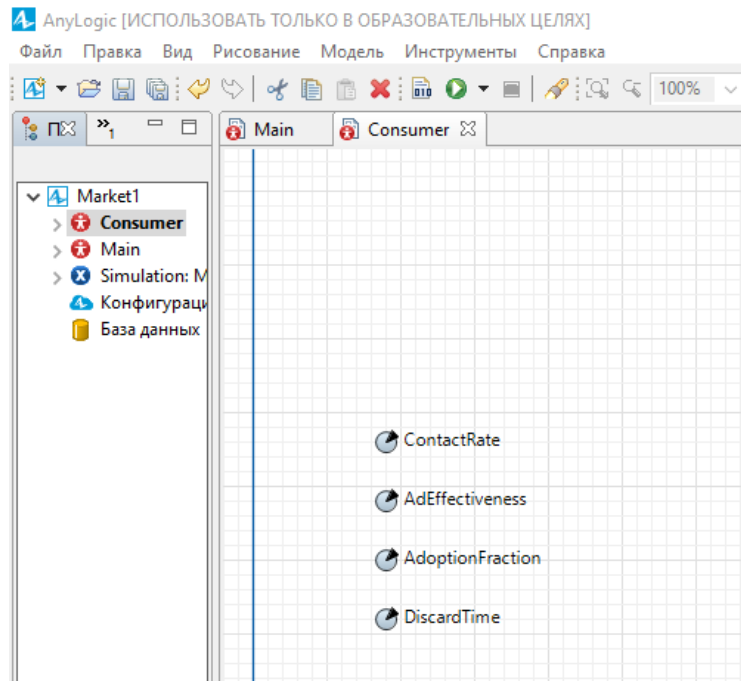


Рис 1.3.3 – Добавление нового параметра

Этап 7 – Моделирование отказов от покупки товара.

Основные действия на этом этапе:

- Добавление невидимых параметров модели;
- Добавление дополнительных переходов на диаграмму состояний;
- Добавление элементов управления.

Конечная диаграмма состояний представлена на Рисунке 1.3.4.

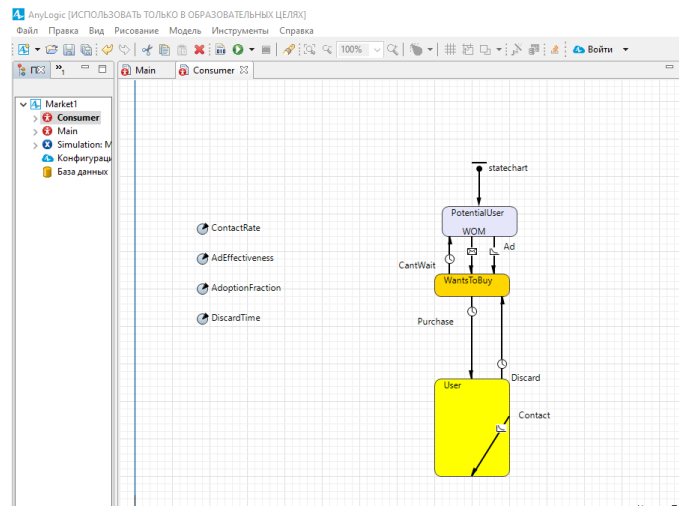


Рис 1.3.4 – Конечная диаграмма состояний на дереве проекта

1.4 Гистограмма

Этап 3 – Добавление графика для визуализации результатов.

Задание диаграммы, которая будет выводить количество потенциальных и действительных потребителей, Рисунок 1.4.1.

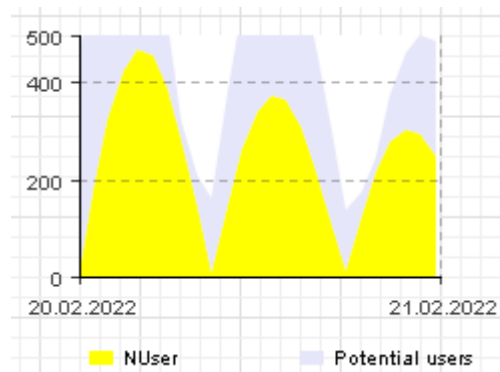


Рис 1.4.1 – График визуализации

Видим, что на диаграмме показывается количество потенциальных и реальных покупателей в определенный момент времени. Процесс насыщения рынка будет происходить очень медленно, на следующем этапе моделирования мы ускорим этот процесс путем оповещения клиентов. Визуальная модель проекта представлена на Рисунке 1.4.2.

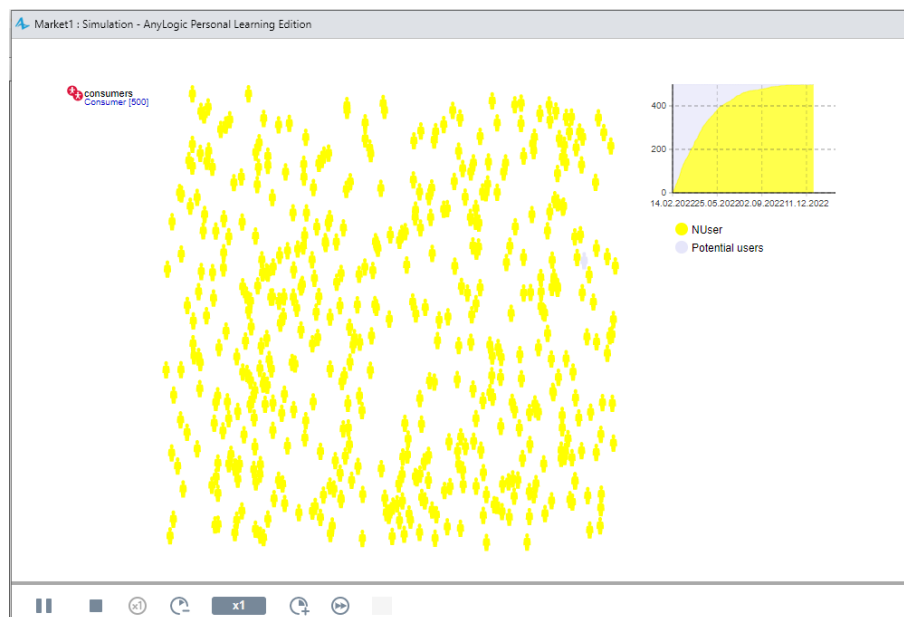


Рис 1.4.2 – Запуск проекта

Этап 4 – добавление эффекта рекомендаций.

Запустим проект и увидим, что из-за оповещения клиентов насыщения рынка будет теперь происходить значительно быстрее. Визуальная модель проекта представлена на Рисунке 1.4.3.

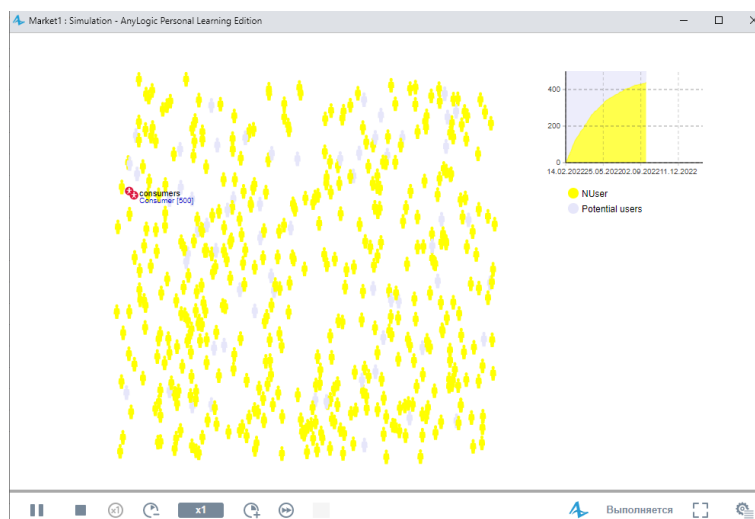


Рис 1.4.3 – Запуск проекта

Этап 5 – Учет повторных продаж продукта.

Запустим наш проект и увидим, что в результате выполненных нами операций на диаграмме часть покупателей, у которых испортился купленный продукт, вновь становятся потенциальными покупателями. Визуальная модель проекта представлена на Рисунке 1.4.4.

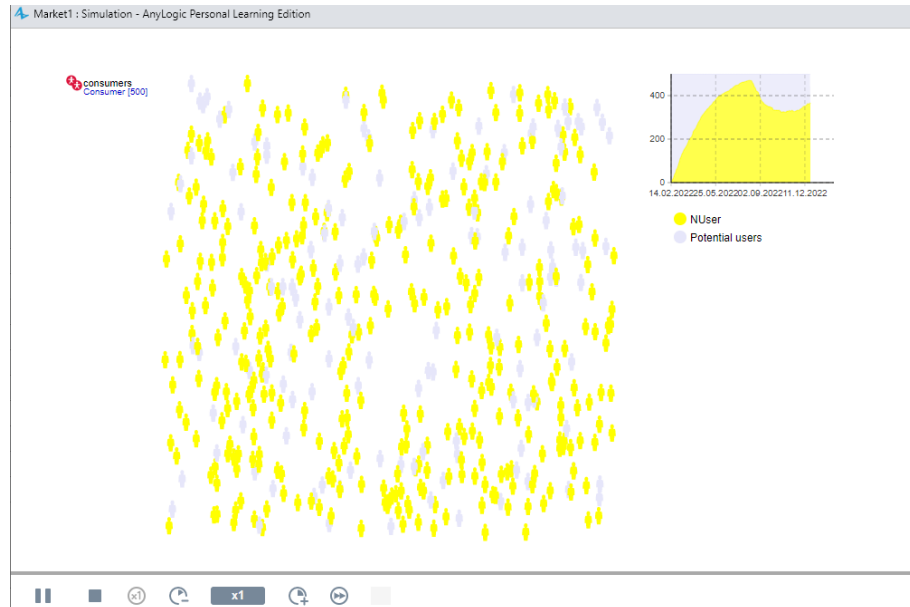


Рис 1.4.4 – Запуск проекта

Этап 6 – Учет времени доставки продукта.

На этом этапе мы:

- Добавляем промежуточное состояние агента;
- Добавляем новую функцию статистики;
- Добавляем новый элемент диаграммы, Рисунок 1.4.5.

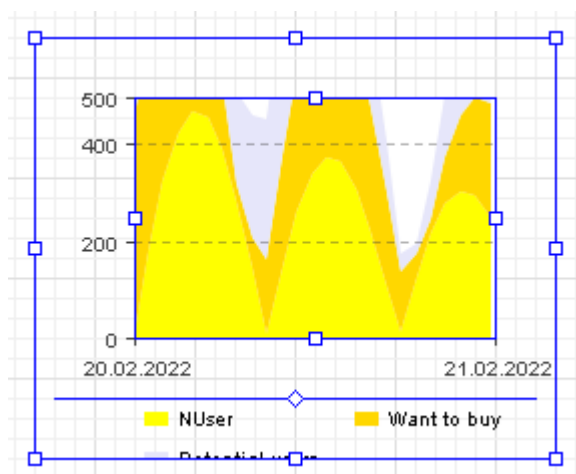


Рис 1.4.5 – График с добавлением нового элемента диаграммы

Запустим проект и в результате запуска проекта увидим Агентов возжелавших приобрести продукт. Визуальная модель проекта представлена на Рисунке 1.4.6.

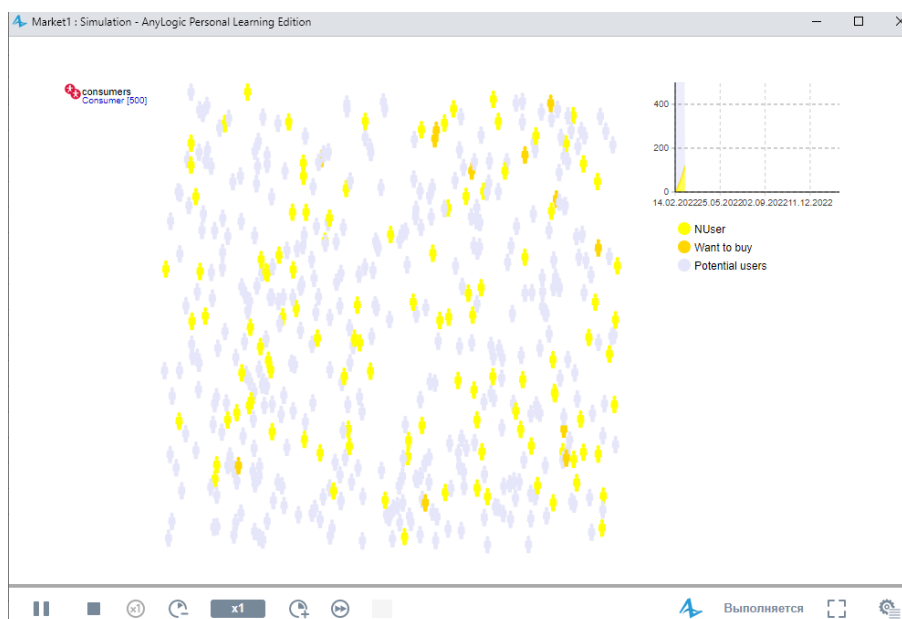


Рис 1.4.6 – Запуск проекта

1.5 Элемент управления

Установим свойства бегунка изменяющего параметр MaxWaitingTime в указанном диапазоне (изменения происходят при запуске модели и при передвижении бегунка вдоль горизонтальной оси), Рисунок 1.5.1

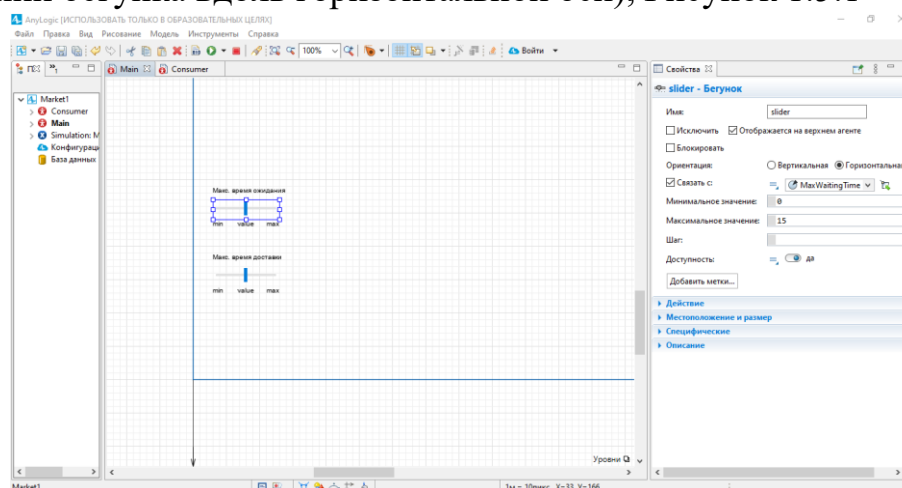


Рис 1.5.1 – Создание бегунка для MaxWaitingTime

Установим свойства бегунка изменяющего параметр MaxDeliveryTime в указанном диапазоне (изменения происходят при запуске модели и при передвижении бегунка вдоль горизонтальной оси), Рисунок 1.5.2.

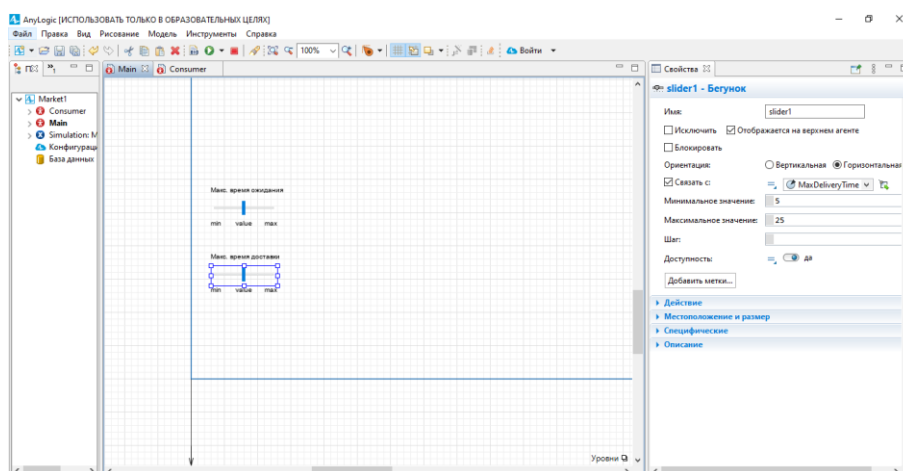


Рис 1.5.2 – Создание бегунка для MaxDeliveryTime

1.6 Итоговый запуск

Запустим проект и увидим, что теперь мы можем регулировать максимальное время ожидания и максимальное время доставки при запуске модели с помощью бегунков. Визуальная модель проекта представлена на Рисунке 1.6.1.

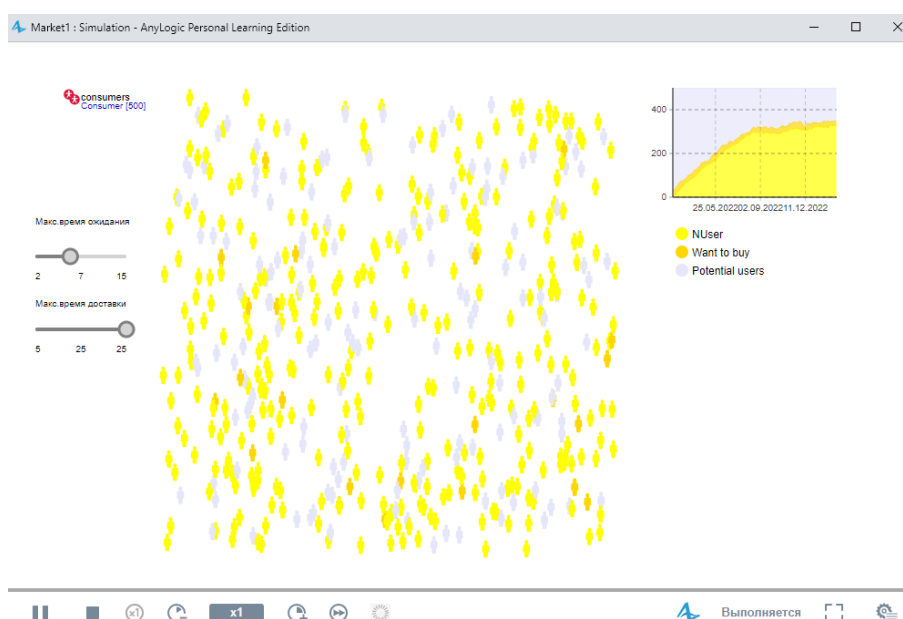


Рис 1.6.1 – Итоговый запуск проекта

2 ПРАКТИЧЕСКАЯ РАБОТА №2

2.1 Постановка задачи

Необходимо создать агентную модель, которая поможет изучить процесс вывода нового продукта на рынок.

- Мы рассмотрим относительно небольшой потребительский рынок численностью в 500 человек. С точки зрения реализации модели каждый потребитель будет являться агентом.
- Поскольку мы рассматриваем процесс вывода на рынок нового продукта, то изначально никто этим продуктом не пользуется.
- Люди начнут покупать продукт под влиянием рекламы.
- После этого начального этапа куда более сильное влияние на продажи будет оказывать общение людей друг с другом, рекомендации и положительные отзывы потребителей продукта, побуждающие других на его приобретение.
- Через некоторое время продукт, купленный людьми, начнет портиться, т.к. истечет срок его годности, и люди вновь начнут покупать продукт.
- Мы смоделируем также людей, возжелавших заказать продукт, а также тех людей, которые будут отменять заказ продукта.

2.2 Этап разгрузки паллетов

Source будет выступать у нас в качестве стартовой точки процесса, он будет использоваться для создания поддонов. Rack Store будет моделировать перемещение поддонов в заданные ячейки стеллажа.

Моделирование прибытия поддонов. Добавим объект Delay и зададим его свойства. Моделирование извлечения поддона из ячейки и прибытия его в заданное место. Добавим объект Rack Pick и зададим его свойства. Изменяем вид поддона. Выберем объект sourcePallets и нажмем кнопку в окне свойств создать другой тип.

Реализация представлена на Рисунке 2.2.1.

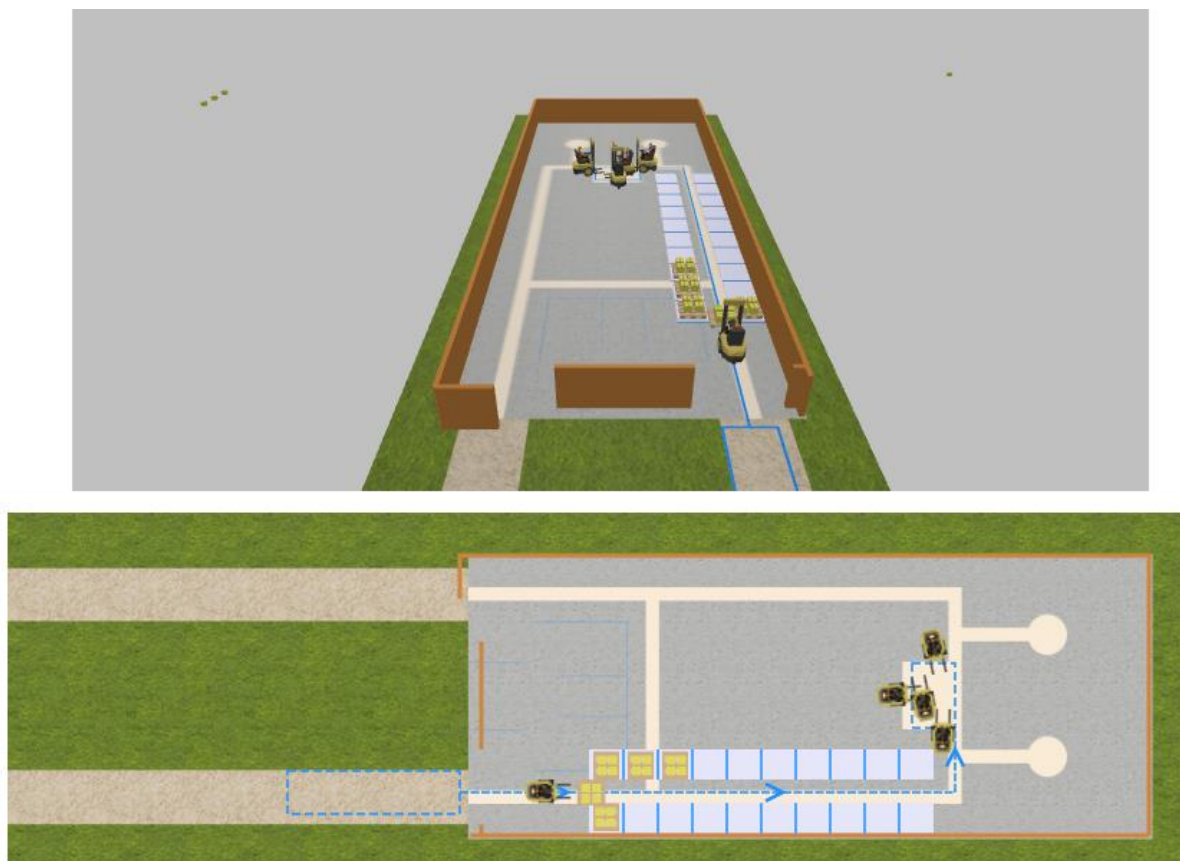


Рис 2.2.1 – Создание и работа поддонов

2.3 Этап управления фурой

В списке фигур анимации раскроем раздел Автодорожный транспорт и выберем из списка фигуру Фура и кликнем по кнопке «Готово», Рисунок 2.3.1.

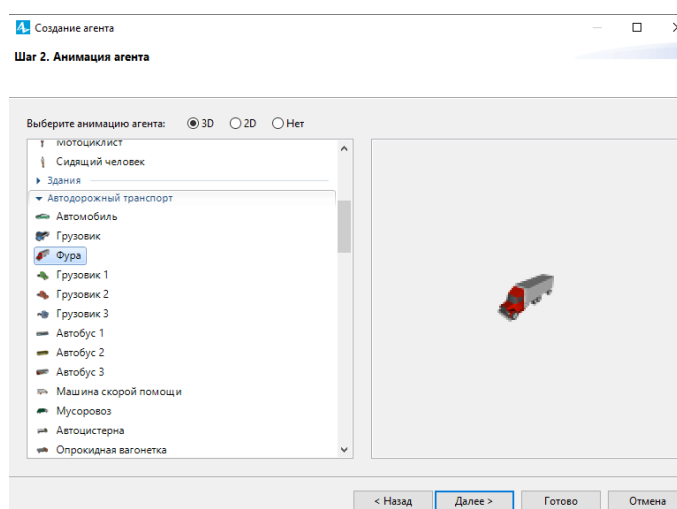


Рис 2.3.1 – Создание фуры

В блоке Source создается фура. Реализация модели представлена на Рисунке 2.3.2.

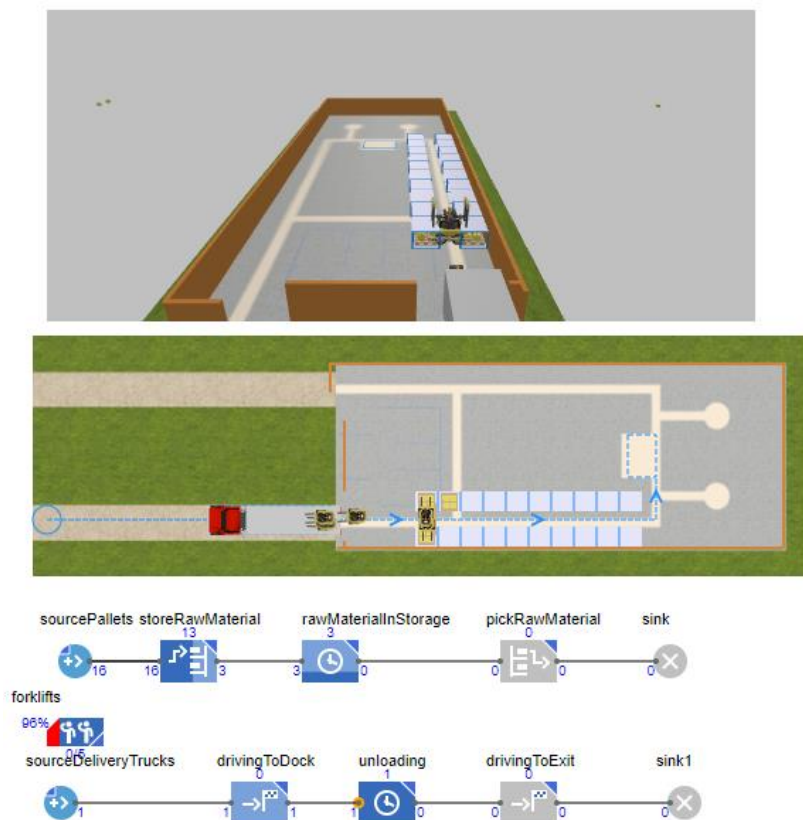


Рис 2.3.2 – Запуск модели

2.4 Камера

1. Устанавливаем с палитры презентации камеру на диаграмму Main и окно трехмерной анимации.
2. Для окна трехмерной анимации установим соответствующие свойства.
3. Запустим модель. На ней нам нужно выбрать начальное положение камеры через контекстное меню на соответствующем элементе управления.
4. В свойствах объекта Камера вставим координаты из буфера. После установки свойств камера должна переместиться в заданное положение.

Реализация представлена на Рисунке 2.4.1.

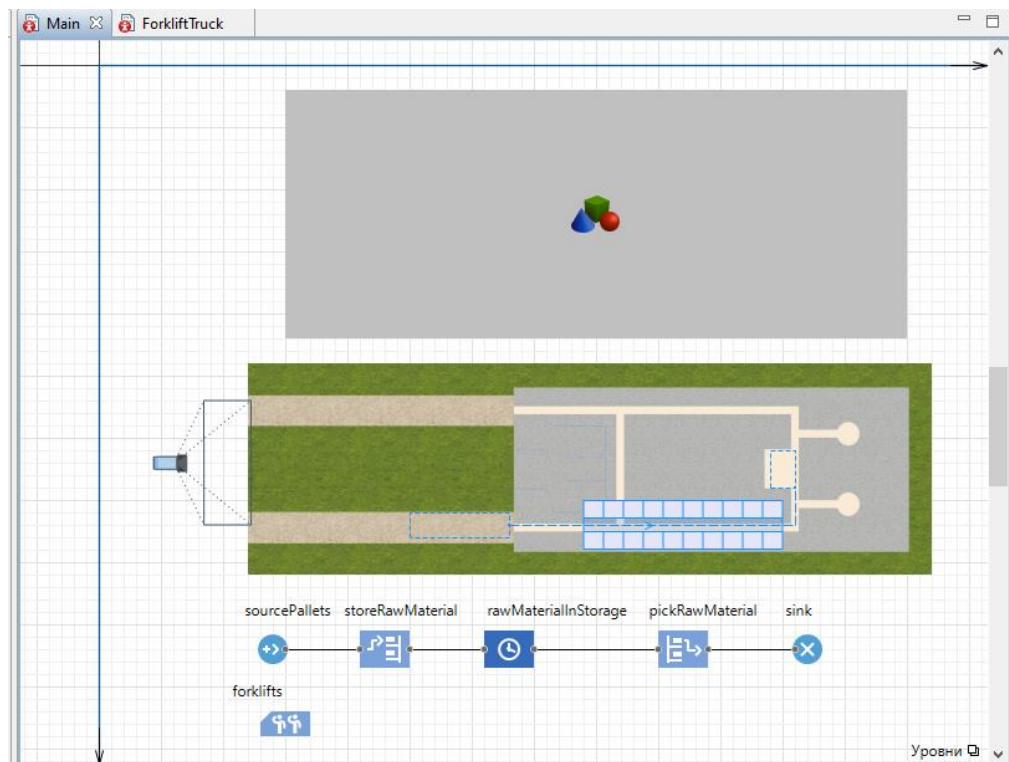


Рис 2.4.1 – Установка камеры и трехмерной анимации

3 ПРАКТИЧЕСКАЯ РАБОТА №3

3.1 Постановка задачи

Необходимо построить модель, изучающую распространение инфекционного заболевания среди населения. Численность населения пусть будет равна 10 000 человек, и задаваться параметром с именем TotalPopulation. На первоначальном этапе заражения популяции болен один человек, а все остальные лишь восприимчивы к болезни. Человек, в организм которого попал вирус, становится латентно зараженным. Латентно зараженные люди, это те люди, у которых инкубационный период для вируса еще не прошел, и нет выраженных симптомов болезни, в этот период вирус в организме человека не способен заражать других людей. После инкубационного периода человек становится больным с выраженными симптомами, и вирус, находящийся в его организме, способен заражать других людей на протяжении болезни человека. Любой человек после выздоровления становится невосприимчивым к болезни т.к. у него вырабатывается иммунитет.

3.2 Этап создания диаграммы потоков и накопителей

Название нашей модели SEIR. SEIR – это аббревиатура, образованная сокращением названий основных стадий распространения инфекции: Susceptible - Exposed - Infectious - Recovered.

- Susceptible – Восприимчивые к заражению люди, которые еще не были заражены вирусом.
- Exposed – Люди, находящиеся в латентной стадии заражения (они уже заражены, но еще не могут заражать других).
- Infectious – Люди в активной стадии заражения (они могут заражать других людей).
- Recovered – Выздоровевшие люди (они приобрели иммунитет к данному заболеванию).

Процесс создания накопителей, потоков и связей представлен на Рисунке 3.2.1 и Рисунке 3.2.2.

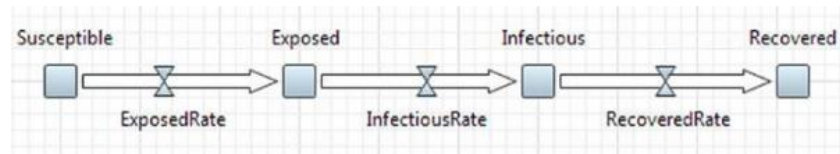


Рис 3.2.1 – Создание накопителей и потоков

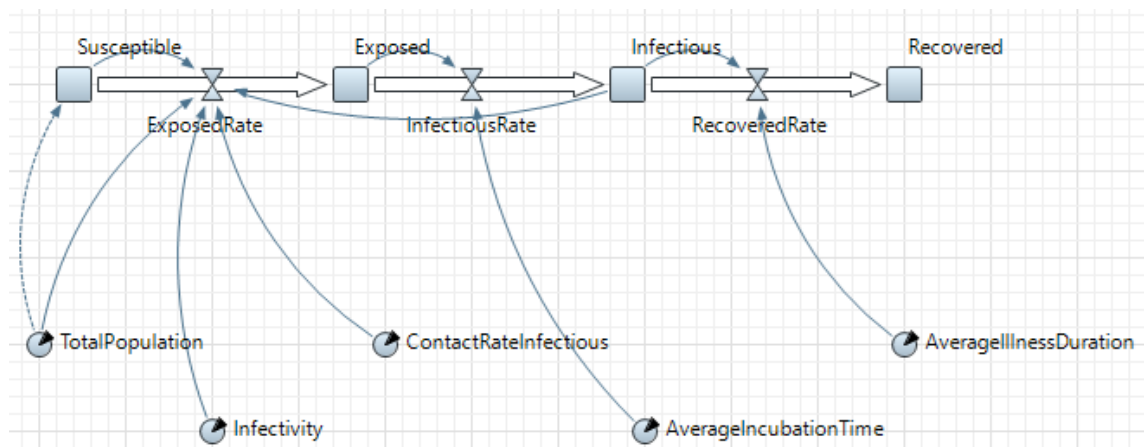


Рис 3.2.2 – Создание связей зависимости

Запустим модель и исследуя динамику процесса с помощью похожих на виджеты информационных окон этих переменных. Мы переключаем виджет в режим графика, для большей наглядности. Работа представлена на Рисунке 3.2.3.

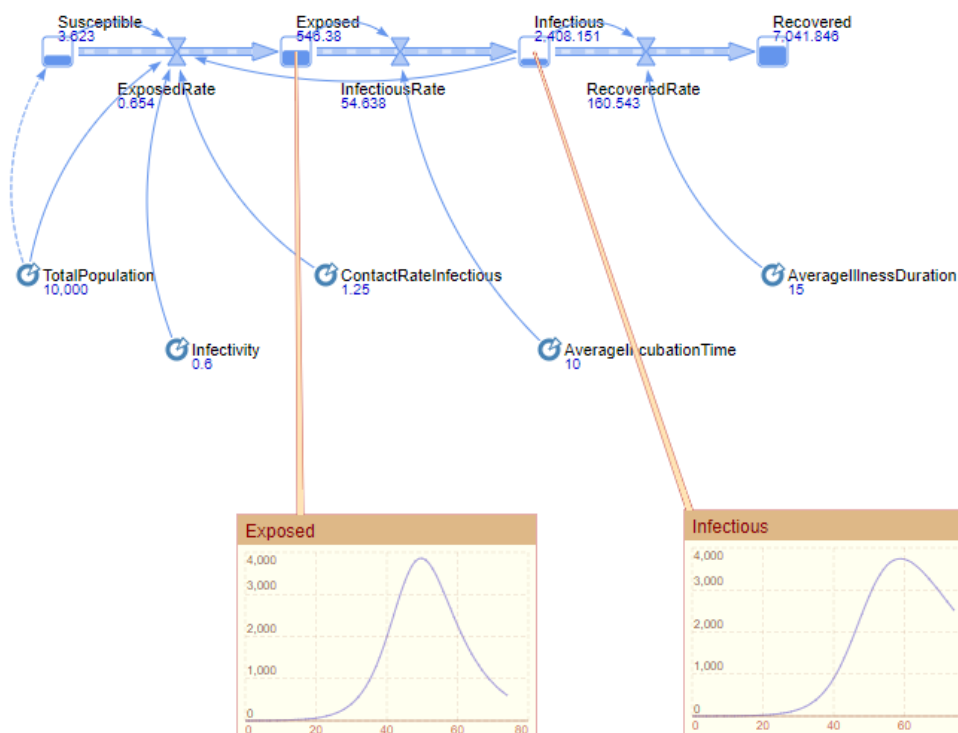


Рис 3.2.3 – Запуск модели

3.3 Этап добавления графика для визуальной динамики процесса

Реализация представлена на Рисунке 3.3.1, Рисунке 3.3.2.

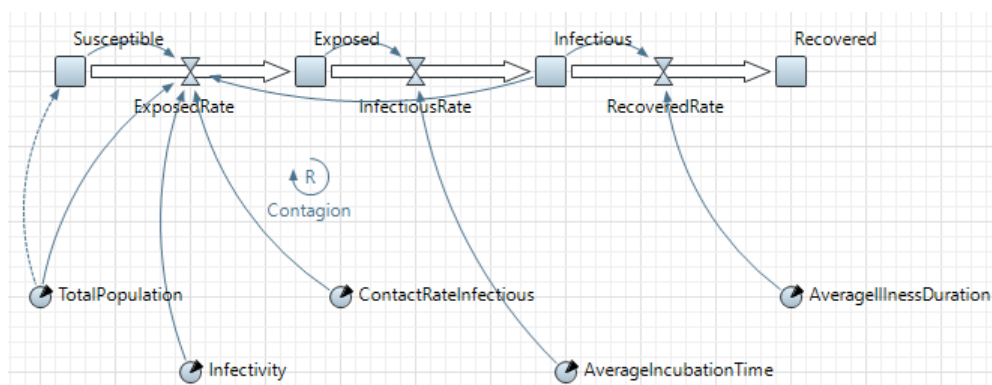


Рис 3.3.1 - Элемент цикла на диаграмме

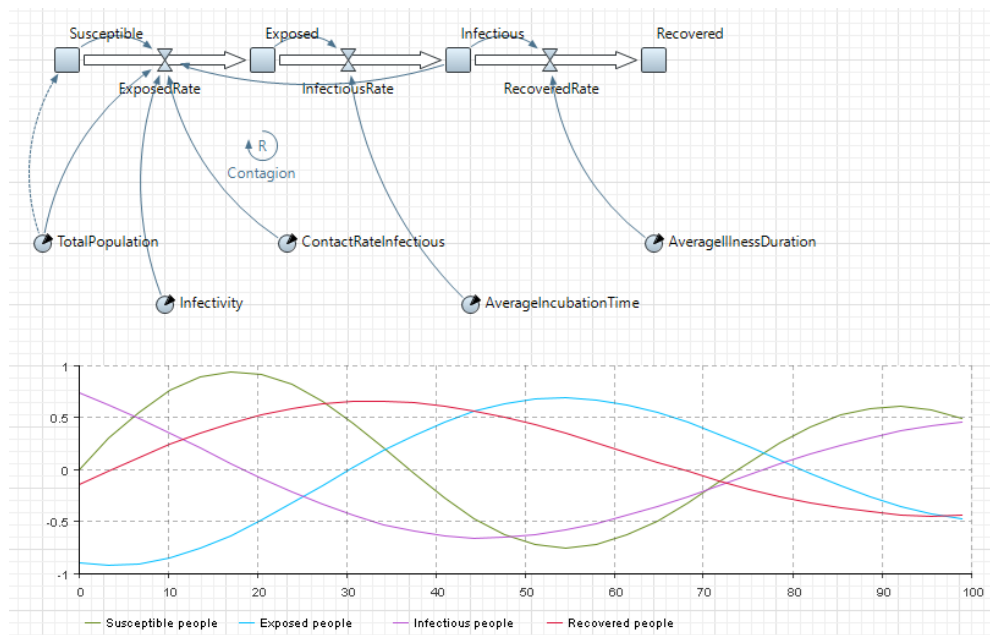


Рис 3.3.2 – Создание временного графика

Запустим модель и изучим динамику распространения болезни с помощью временного графика. Результат представлен на Рисунке 3.3.3.

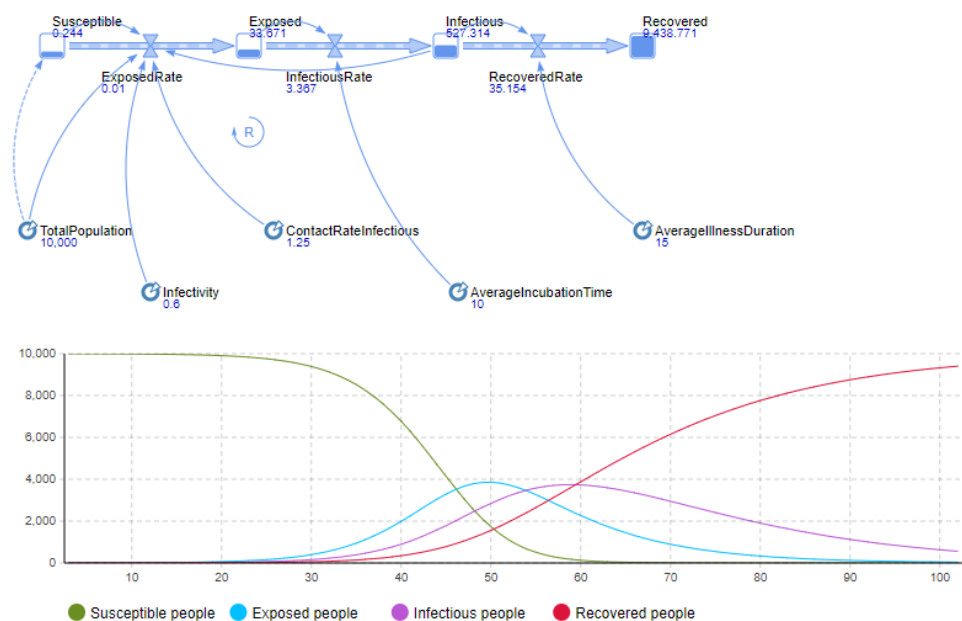


Рис 3.3.3 - Запуск модели с временным графиком

3.4 Этап экспериментального варьирования параметров

Реализация представлена на Рисунке 3.4.1 и Рисунке 3.4.2.

Новый эксперимент

Эксперимент
Выберите тип эксперимента, задайте имя и выберите агента верхнего уровня.

Имя:

Агент верхнего уровня:

Тип эксперимента:

- Простой эксперимент
- Оптимизация
- Варьирование параметров**
- Сравнение "прогнозов"
- Монте-Карло
- Анализ чувствительности
- Калибровка
- Обучение ИИ
- Нестандартный

Выполняет несколько "прогнозов" модели с варьированием одного или нескольких параметров, с возможностью использования репликаций. Впоследствии Вы можете создать для эксперимента интерфейс любой сложности.

☒ Копировать установки модельного времени из:

< Назад Далее > **Готово** Отмена

Рис 3.4.1 – Создание эксперимента

SEIR : ContactRateVariation

Итерация: 18

Параметры

TotalPopulation	10,000
Infectivity	0.6
ContactRateInfectious	2
AverageIncubationTime	10
AverageIllnessDuration	15

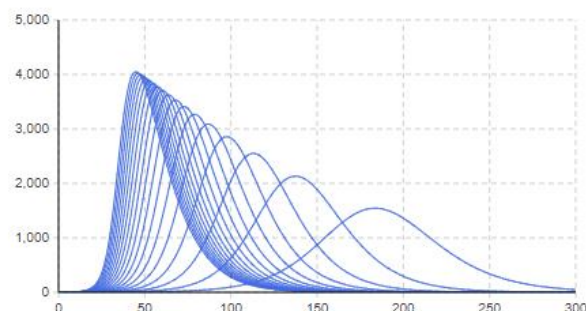


Рис 3.4.2 – Запуск эксперимента

4 ПРАКТИЧЕСКАЯ РАБОТА №4

4.1 Постановка задачи

Необходимо создать агентную модель, которая поможет изучить процесс доставки товара пользователю, с момента совершения покупки.

- Мы рассмотрим относительно небольшой потребительский рынок численностью в 500 человек. С точки зрения реализации модели каждый потребитель будет являться агентом.
- Поскольку мы рассматриваем весь процесс, сначала мы рассмотрим на выбор товара пользователем.
- Люди определяются с товаром, и начинают оформлять доставку.
- После этого начального этапа куда более сильное влияние на продажи будет оказывать общение людей друг с другом, рекомендации и положительные отзывы потребителей продукта, побуждающие других на его приобретение.
- Через некоторое время продукт будет доставлен покупателю.
- Мы смоделируем также людей, возжелавших заказать какой-нибудь новый продукт.

4.2 Создание популяции

Мы создали проект и установили в нем все необходимые настройки. Создали агента и его популяцию, Рисунок 4.2.1.

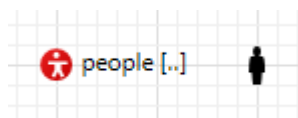


Рис 4.2.1 - Агент People

Выбрали кнопку для компиляции, и запустили проект. В результате, на экране появилось окно запуска. Визуальная модель проекта представлена на Рисунке 4.2.2.

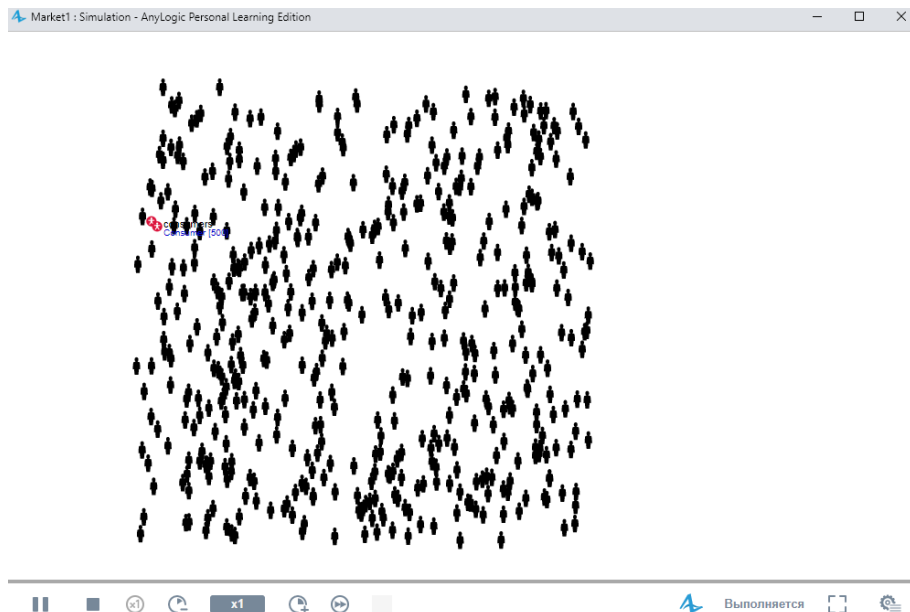


Рис 4.2.2 – Запуск проекта

4.3 Диаграмма состояний

Этап 2 – Задание поведения потребителей.

Двойным кликом выберем People на дереве проекта и этим откроем его окно редактора. Выберем палитру, тип элемента диаграммы, и перетащим элемент Начало диаграммы в окно редактора. В результате выполненных действий будет построена диаграмма на дереве проекта People, Рисунок 4.3.1.

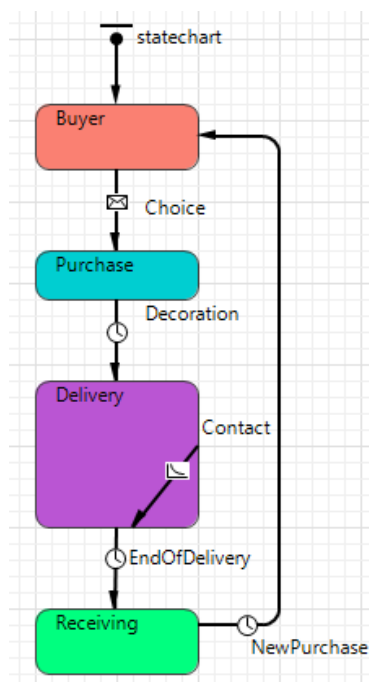


Рис 4.3.1 – Диаграмма состояний

Каждая из составляющих отображает определенный процесс:

- Состояние Buyer – олицетворяет нашего покупателя;
- Состояние Purchase – покупка товара;
- Состояние Delivery – доставка товара;
- Состояние Receiving – получение товара;
- Переход Choice – выбор покупателем товара;
- Переход Decoration – оформление доставки товара;
- Переход EndOfDelivery – конец доставки товара;
- Переход NewPurchase – покупатель возвращается к новым покупкам.

Мы добавляем параметры для нашей работы, Рисунок 4.3.2:

- DecorationMin/Max – оформление покупки доставки;
- DeliveryMin/Max – срок доставки товара;
- ReceivingMin/Max – срок получения товара, в пункте выдачи;
- DelayPosibility – вероятность задержки доставки;
- ContactsPerDay – контакт с консультантами в день.

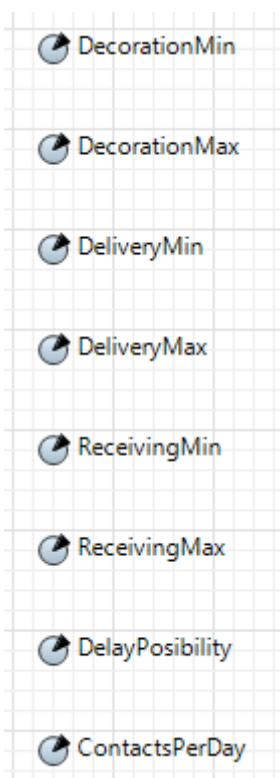



Рис 4.3.2 - Параметры

4.4 Гистограмма

В качестве параметров и свойств, используем исходные данные, Рисунок 4.4.1.

 **chart - Временная диаграмма с накоплением**

Заголовок:

NDelivery

Значение:

people.NDelivery()

Цвет:

mediumOrchid

☒ Значение ☐ Набор данных

Заголовок:

NPurchase

Значение:

people.NPurchase()

Цвет:

darkTurquoise

☒ Значение ☐ Набор данных

Заголовок:

NBuyer

Значение:

people.NBuyer()

Цвет:

salmon

☒ Значение ☐ Набор данных

Заголовок:

NReceiving

Значение:

people.NReceiving()

Цвет:

springGreen

+

×

↑

↓

Рис 4.4.1 – Данные гистограммы

Реализация гистограммы представлена на Рисунке 4.4.2.

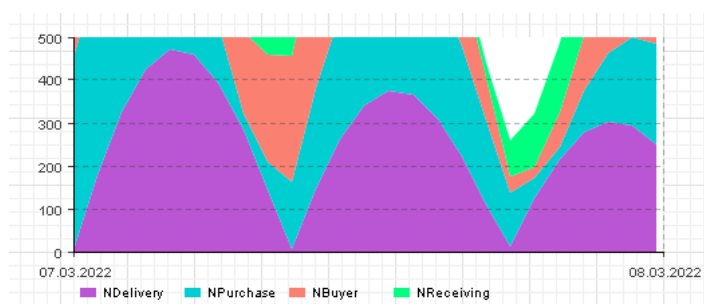


Рис 4.4.2 - Реализация гистограммы

4.5 Элемент управления

Установим свойства для бегунков, изменяющих параметры DecorationMin, DecorationMax, DeliveryMin, DeliveryMax, ReceivingMin, ReceivingMax в указанном диапазоне (изменения происходят при запуске модели и при передвижении бегунка вдоль горизонтальной оси), Рисунок 4.5.1 – Рисунок 4.5.4.

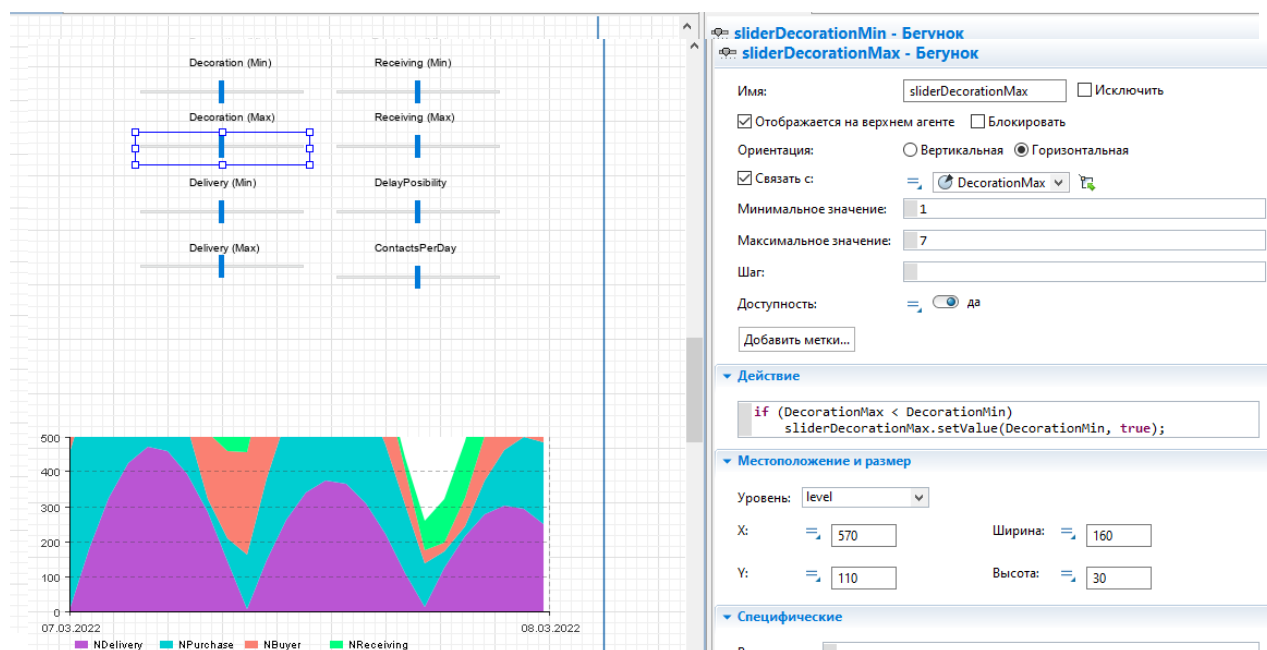


Рис 4.5.1 – Создание бегунка для DecorationMin

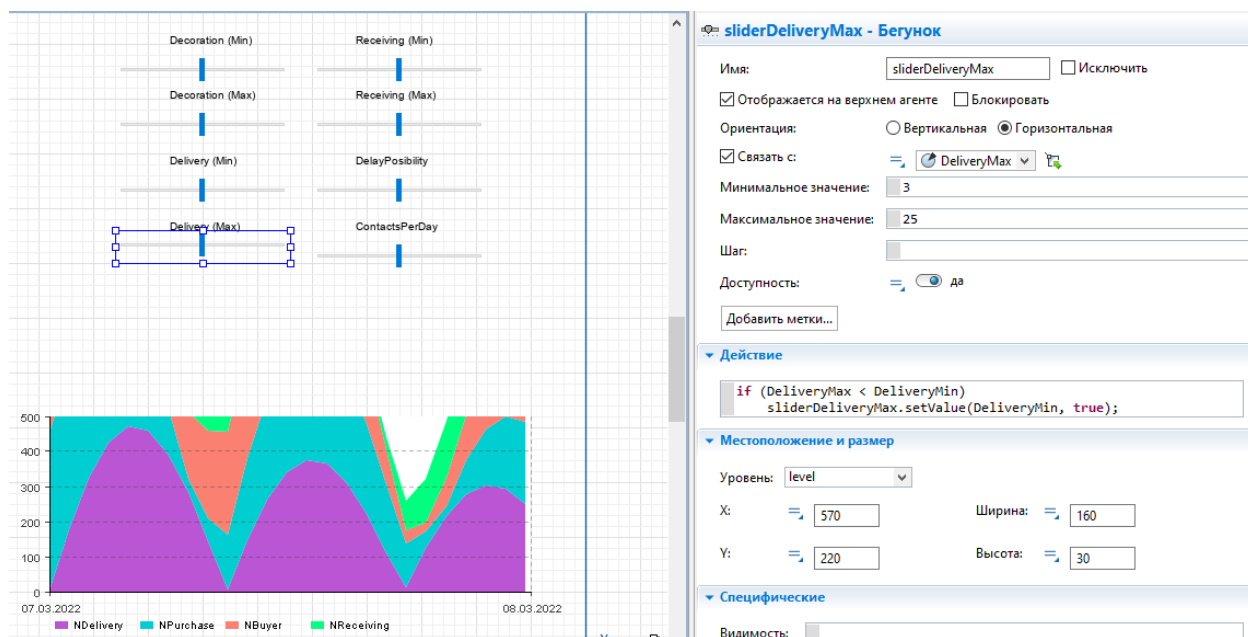


Рис 4.5.2– Создание бегунка для DeliveryMax

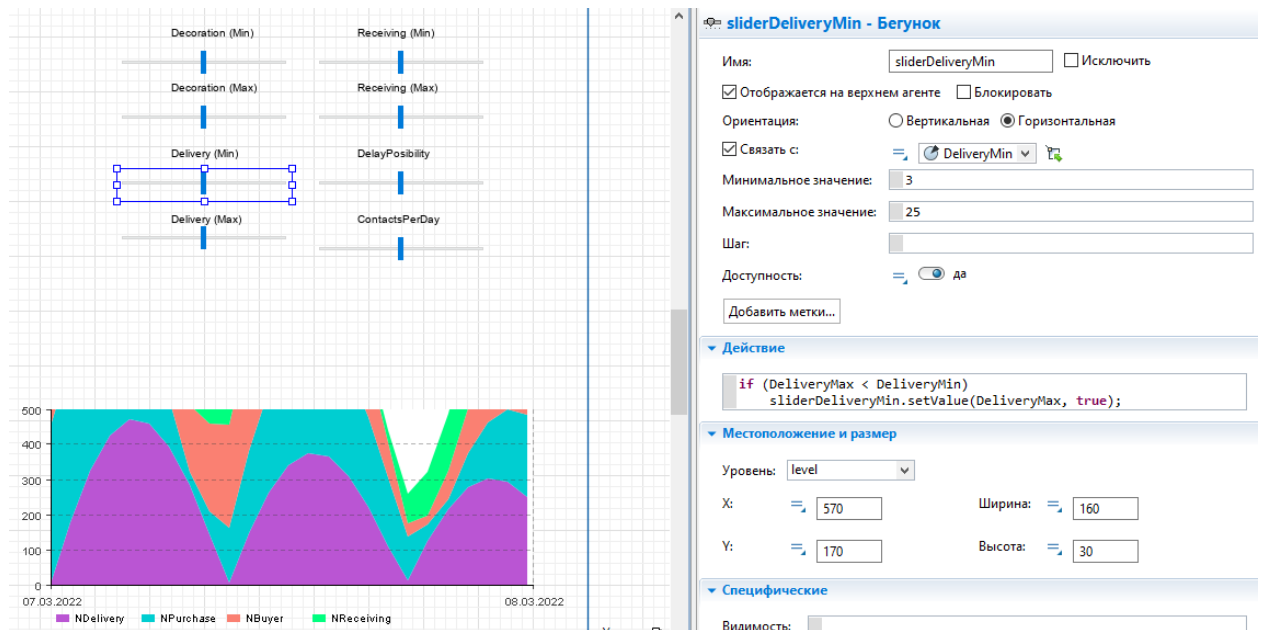


Рис 4.5.3— Создание бегунка для DeliveryMin

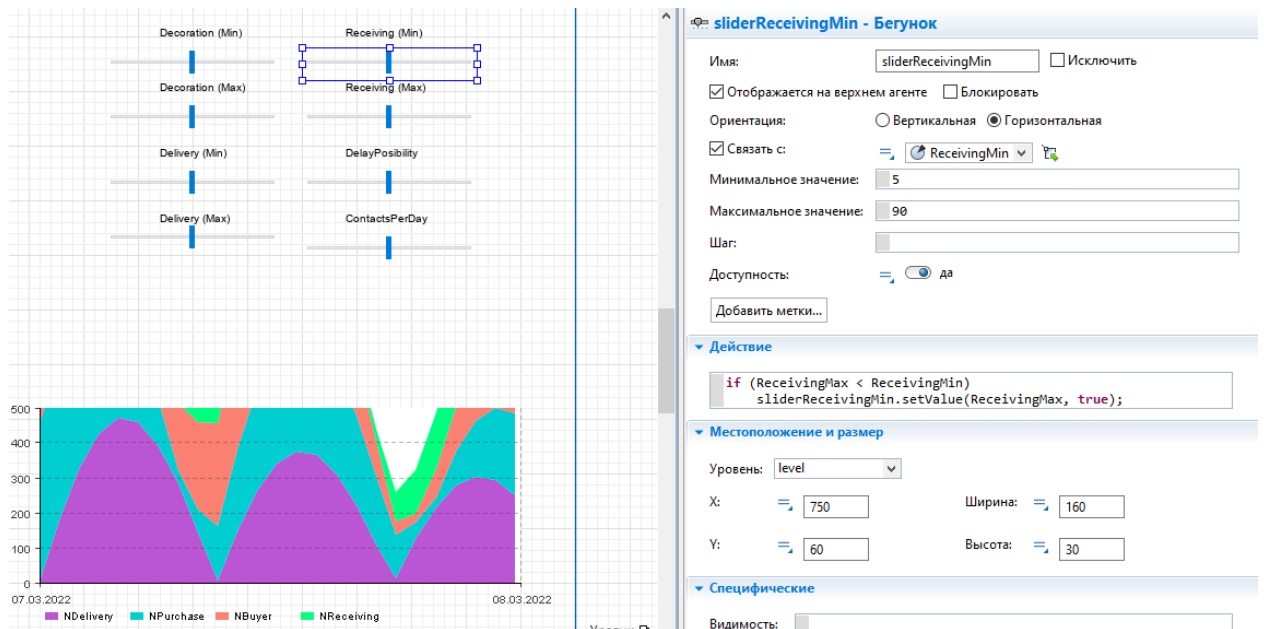


Рис 4.5.4— Создание бегунка для ReceivingMin

4.6 Итоговый запуск

Запустим проект и увидим, что теперь мы можем регулировать максимальное время ожидания и максимальное время доставки при запуске модели с помощью бегунков. Визуальная модель проекта представлена на Рисунке 4.6.1.

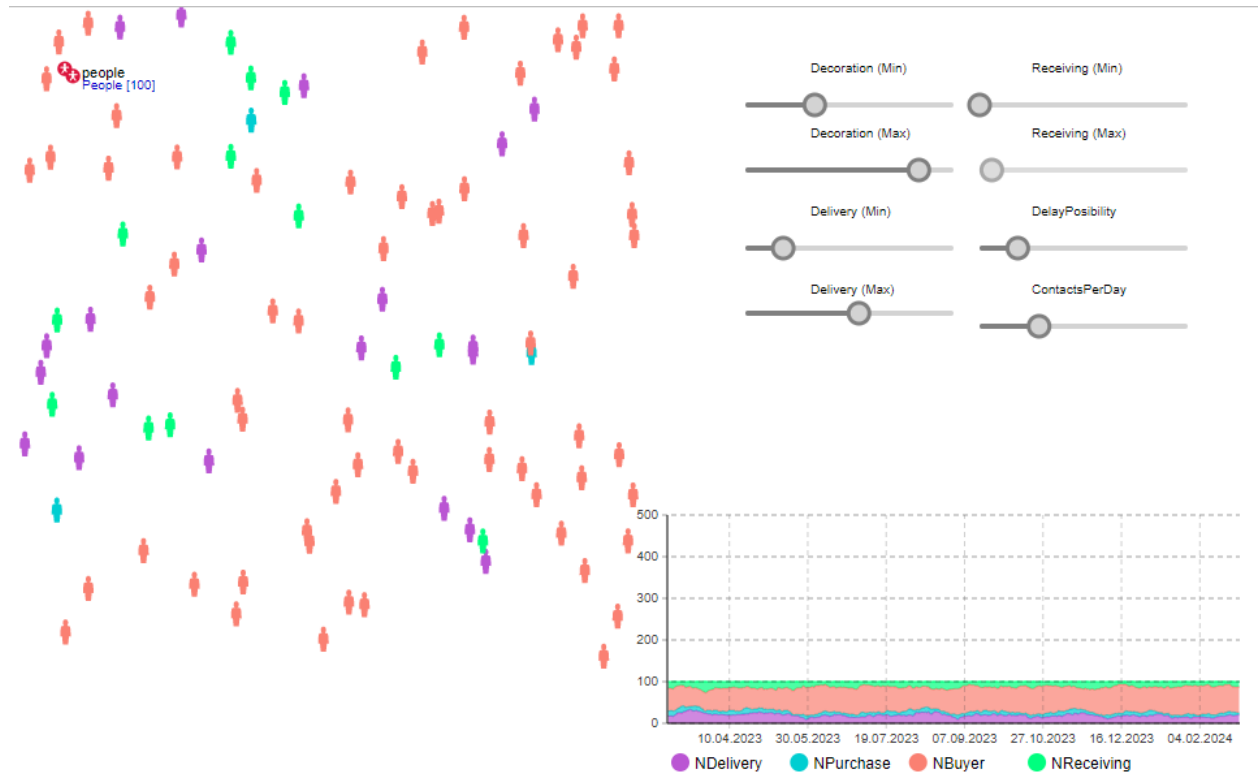


Рис 4.6.1 – Итоговый запуск проекта

5 ПРАКТИЧЕСКАЯ РАБОТА №5

5.1 Постановка задачи

Мы промоделируем производственный процесс работы в банке. У нас есть:

- Банкомат – через который люди производят операции;
- Консультанты – которые помогают посетителям;
- Посетители – которым, либо нужна помощь, либо они хотят сделать денежные операции через банкомат.

5.2 Этап создания пути для клиента

После того, как мы создали нашего объекта – Client. Мы должны прописать его передвижение, и действия (Рисунок 5.2.1):

- Source – задает движение нашему объекту, т.е. начальная точка;
- Queue – очередь;

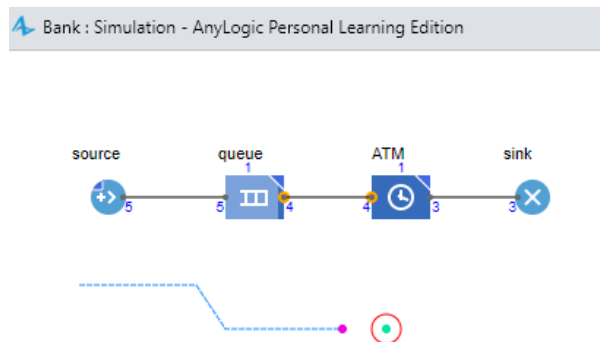


Рис 5.2.1-Создание траектории пути

На этом этапе мы создаем саму траекторию движения нашего главного объекта, а именно, клиента банка. Сделаем 3D-визуализацию нашего проекта (Рисунок 5.2.2).

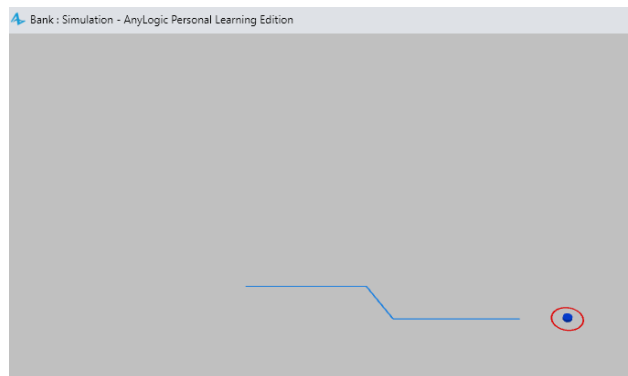


Рис 5.2.2 – 3D визуализация движения объекта

5.3 Этап загрузки аттракторов

Обозначим наши объекты, и зададим агенту – Client возможность движения. Мы создаем аттракторы, которые отвечают за поворот объекта. Помимо наших основных объектов, для красивой картинки банка, мы добавляем в наше поле 3D – объекты (Рисунок 5.3.1).

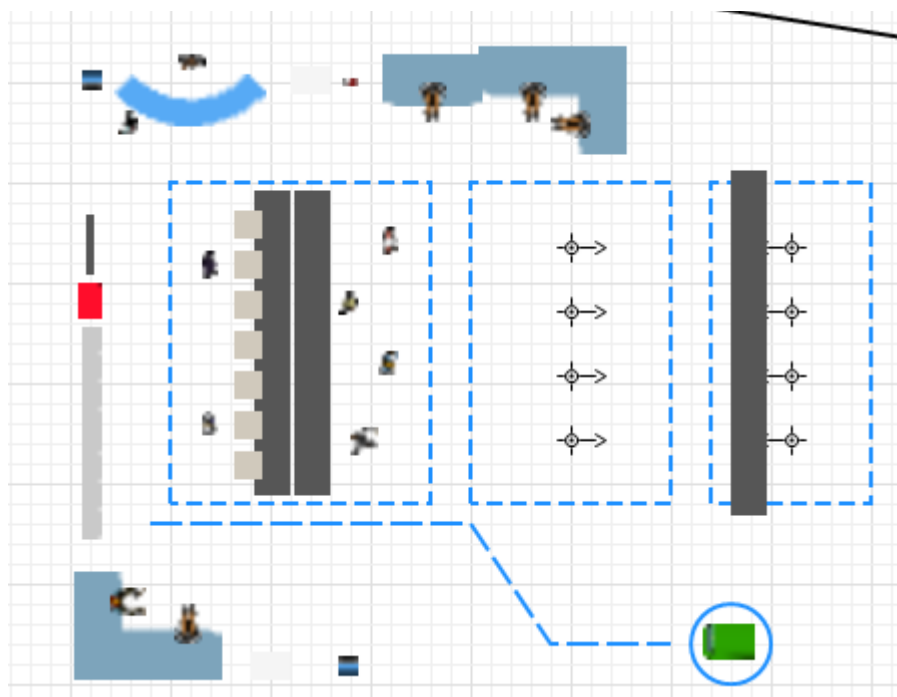


Рис 5.3.1 – Добавление аттракторов и 3D-объектов

5.4 Запуск программы

На Рисунке 5.4.1 представлен фрагмент первичного запуска нашей модели. Когда из объектов у нас был только Client.

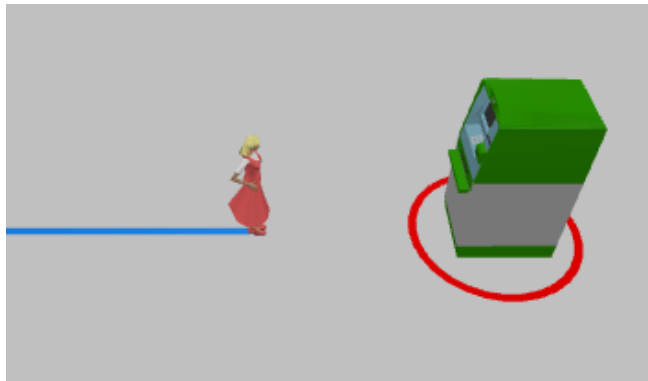


Рис 5.4.1 – Первичный запуск проекта

На конечном запуске проекта у нас уже имеется 2 агента (Рисунок 5.4.2):

- Client – наш пользователь (клиент);
- Teller – кассир, помощник.

А также, дополнительные 3D-объекты, которые выступают в качестве оформления банка.

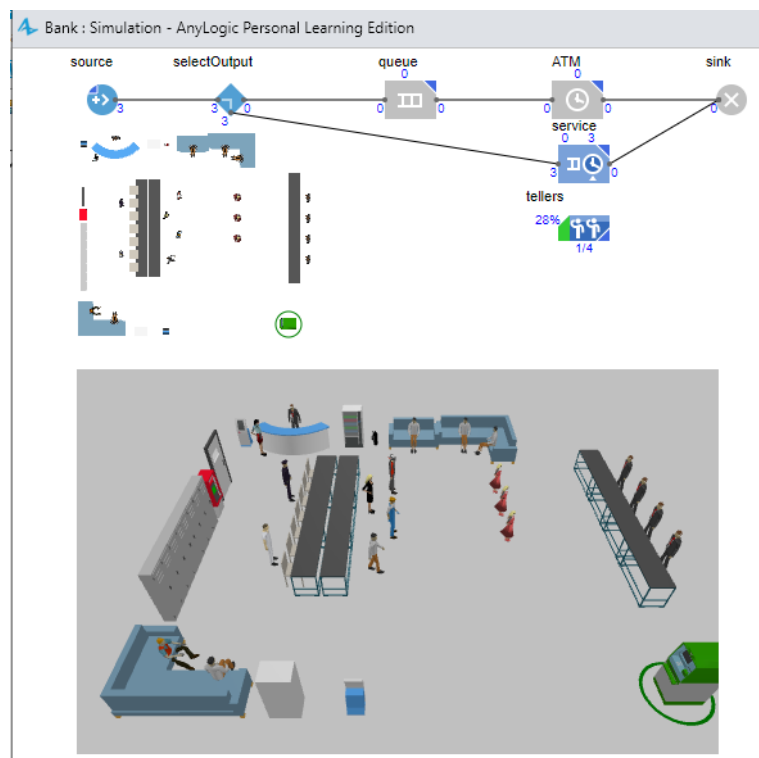


Рис 5.4.2 – Конечный запуск

6 ПРАКТИЧЕСКАЯ РАБОТА №6

6.1 Постановка задачи

В качестве основы для этого задания, я взяла пример с двумя предприятиями.

Допустим, есть два альтернативных продукта А и В, производимых разными (и конкурирующими) компаниями, каждая из которых имеет свою собственную цепочку поставок. Потребители изначально не используют какие-либо продукты, но чувствительны к рекламе и сарафанному радио.

Через некоторое время оба продукта отказываются и генерируют необходимость купить замену той же марки. Переключение продукта может произойти, если продукт предпочтительной марки недоступен из-за проблем в цепочке поставок. Потребительский рынок моделируется агентным методом, т.е. создаем популяцию с клиентами, для наглядности. Цепочки поставок моделируются с помощью системной динамики. Выход модели включает в себя доли рынка для А, В и спрос.

6.2 Этап создания диаграммы потоков и накопителей

В работе представлены две цепочки поставок (Supply chain), которые олицетворяют оба производства, для более удобной работы, были добавлены бегунки, они помогают посмотреть на сдвиги поставок, с учетом дней доставки.

- Production A/B – представляет собой производство A/B;
- FactoryStock A/B – заводские запасы;
- RetailerStock A/B – представляют собой запасы розничных торговцев;
- Параметр Delivery A/B – доставка;
- Параметр Forecast A/B – прогноз;
- Переменная NWantA – представляет собой потребителей, которые хотят только продукт А;

- Переменная NWantAny - представляет собой потребителей, которые хотят продукт любой марки;
- Переменная NWantB – представляет собой потребителей, которые хотят только продукт В.

Процесс создания накопителей, потоков и связей представлен на Рисунке 6.2.1.

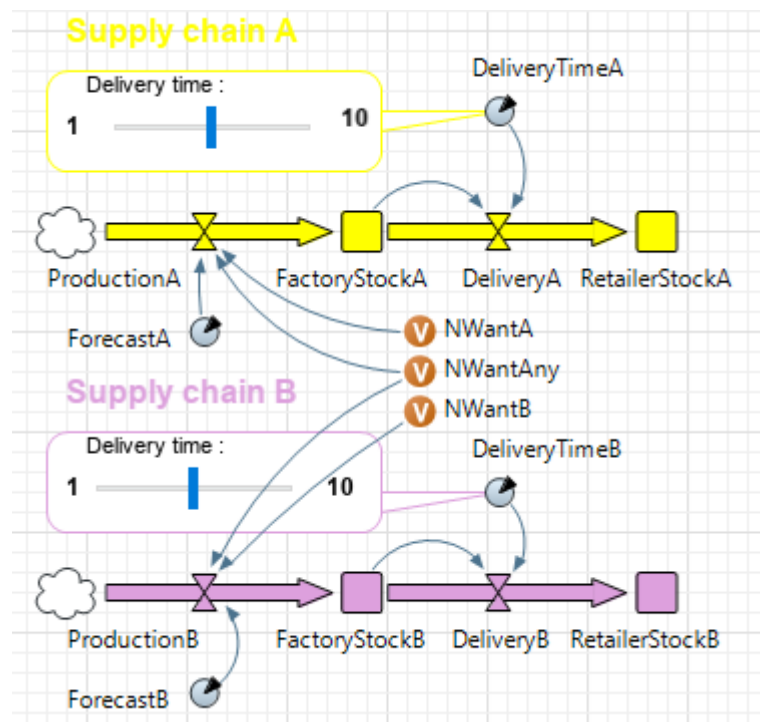


Рис 6.2.1 – Создание накопителей и потоков

Запустим модель и исследуя динамику процесса с помощью похожих на виджеты информационных окон этих переменных. Мы переключаем виджет в режим графика, для большей наглядности. Работа представлена на Рисунке 6.2.2.



Рис 6.2.2 – Запуск модели

6.3 Этап создания диаграммы состояний

Диаграмма состояний представляет собой схему взаимодействия объектов и действий, Рисунок 6.3.1.

У нас есть 3 категории потребителей:

- Те, кому важно использовать товар производителя А;
- Те, кому все равно на производителя;
- Те, кому важно использовать товар производителя В.

И из этого у нас образовывается взаимосвязь, между продуктом и потребителем.

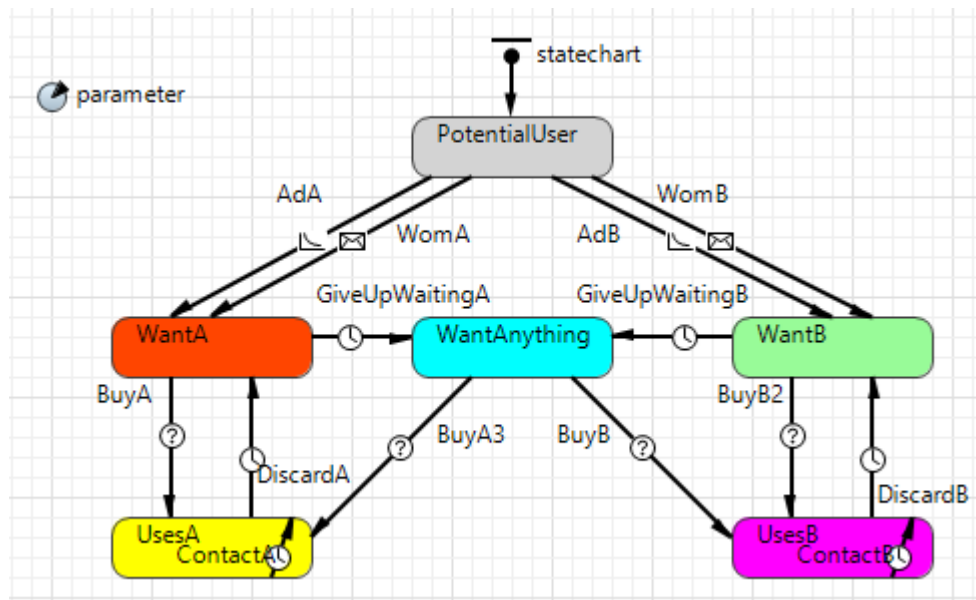


Рис 6.3.1 – Диаграмма состояний

6.4 Этап добавления графика для визуальной динамики процесса

Реализация представлена на Рисунке 6.4.1 и Рисунке 6.4.2.

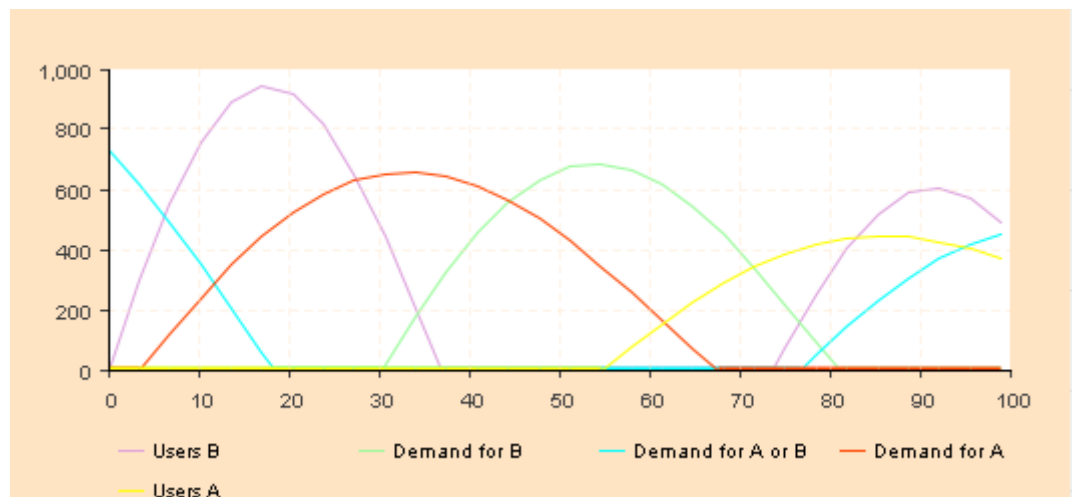


Рис 6.4.1 – Добавление модели с временным графиком

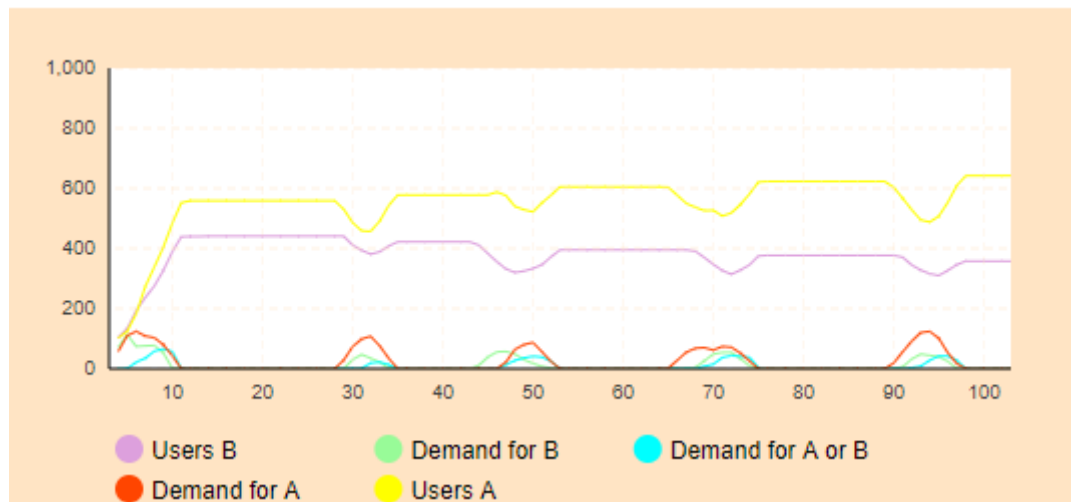


Рис 6.4.2 – Запуск модели с временным графиком

6.5 Запуск проекта

Запуск программы представлен на Рисунке 6.5.1.

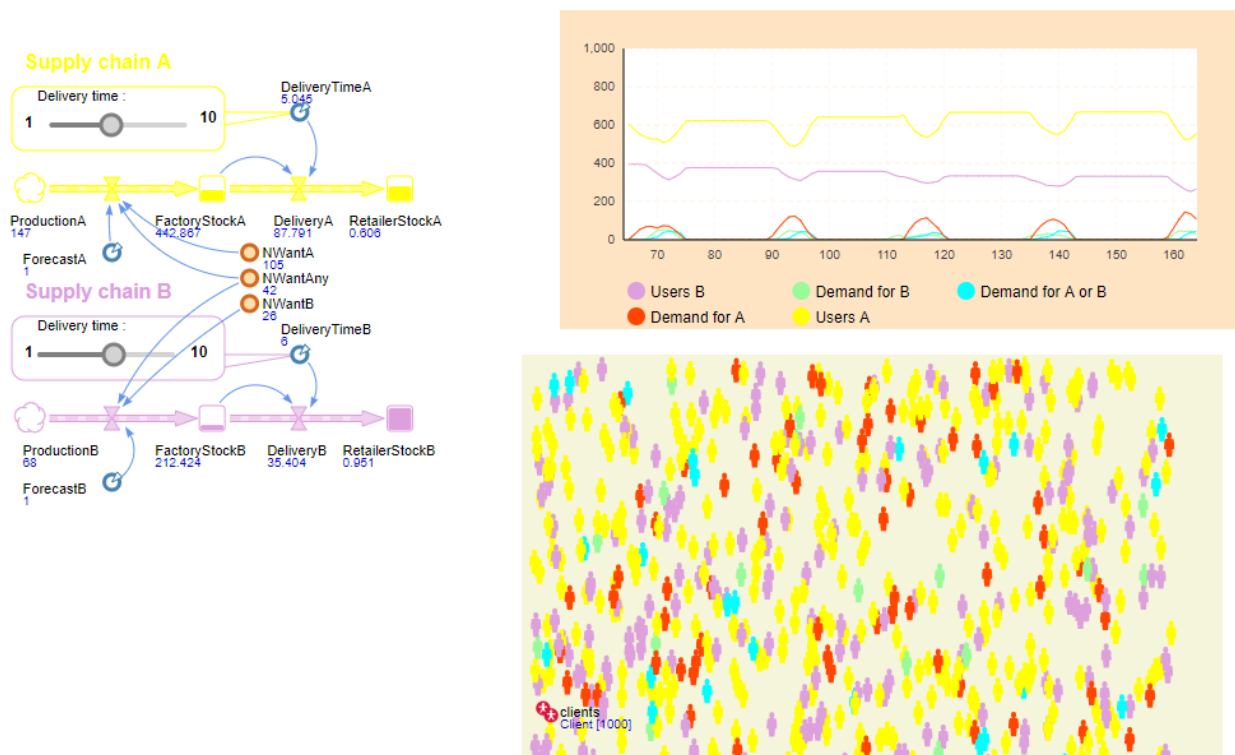


Рис 6.5.1 – Запуск проекта

6.6 Этап экспериментального варьирования параметров

Реализация представлена на Рисунке 6.6.1 и Рисунке 6.6.2.

PR_6_1 : ParametersVariation

Итерация: 21

Параметры	
Delivery time A	10
Delivery time B	10
Forecast A	300
Forecast B	100

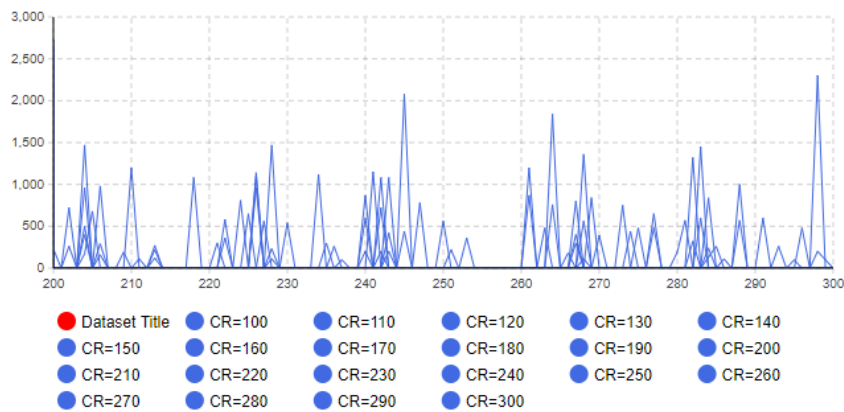


Рис 6.6.1 – Для случая с предприятием А

PR_6_1 : ParametersVariation

Итерация: 21

Параметры	
Delivery time A	10
Delivery time B	10
Forecast A	100
Forecast B	300

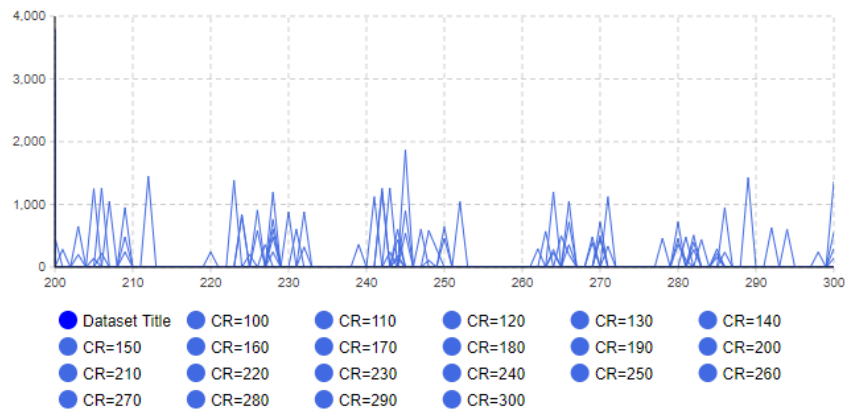


Рис. 5.2 – Для случая с предприятием В

7 ПРАКТИЧЕСКАЯ РАБОТА №7

7.1 Постановка задачи

Необходимо создать популяцию агентов и показать ее в среде обитания. Каждый агент является потенциальным покупателем, определенный процент агентов в течение дня становится покупателем (пользователем продукта) и это видно в среде и на графике. Каждый пользователь продукта после покупки продукта может оповестить одного потенциального покупателя, что увеличит процент покупок продукта в день. Продукт портится через несколько дней, и пользователь продукта через определенное время становится вновь потенциальным покупателем, и это отражается в среде обитания и на графике.

7.2 Описание процесса создания проекта и его запуск

Изначально, у нас в проекте есть только 2 состояния:

- Люди, в течение дня становятся покупателем (пользователем продукта);
- Люди, которые становятся вновь потенциальным покупателем.

Но я добавила третье состояние. Людей, которые рекомендуют потребителям товар и увеличивают продажи.

У каждого из видов потребителей есть свой собственный цвет:

- Желтый – первые покупатели (Consumers);
- Зеленый – постоянные покупатели (Buyers);
- Синий – покупатели, которые советуют товар другим (Advisers).

Для реализации программы было реализовано 6 классов:

- Person - отвечает за наших агентов;
- City - отвечает за состояние агентов в городе;
- Constants - отвечает за константы (параметры);
- GraphPanel - отвечает за окно с графиком;
- CityPanel - отвечает за сборку окна с агентами;
- App - отвечает за запуск проекта, и сборку окна.

Первоначальное состояние агентов можно увидеть на Рисунке 7.2.1.

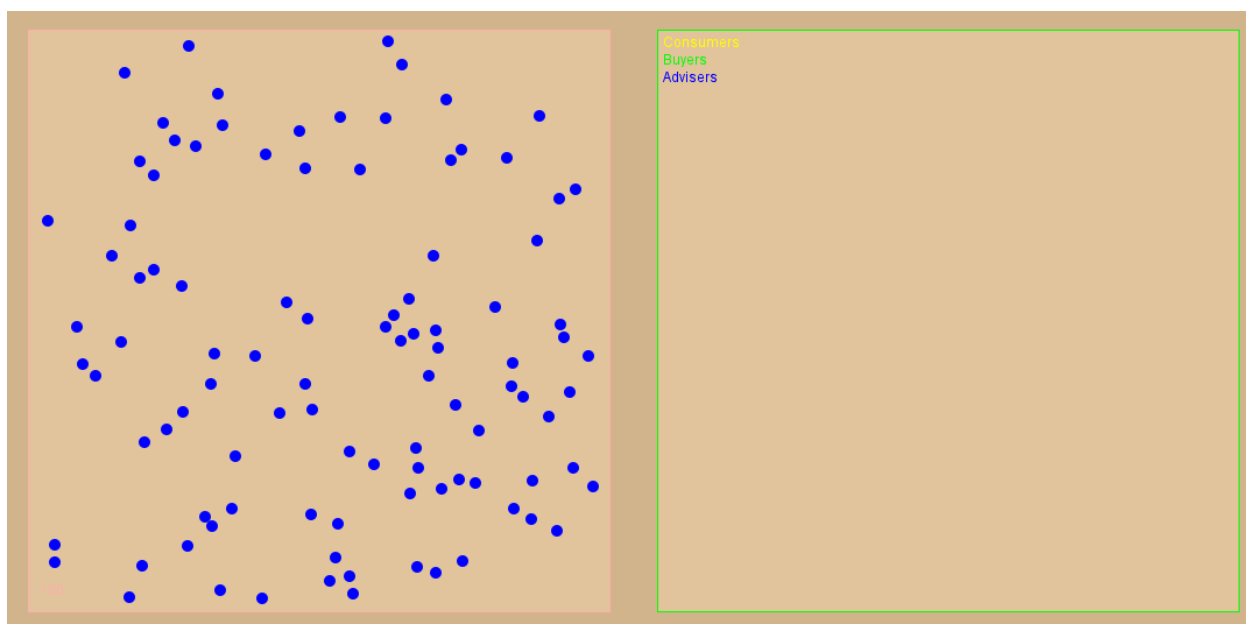


Рис 7.2.1 – Первоначальное состояние

Мы видим, что наши покупатели синего цвета, а значит, популярность товара начинает распространяться.

На Рисунке 7.2.2 показан следующий шаг.

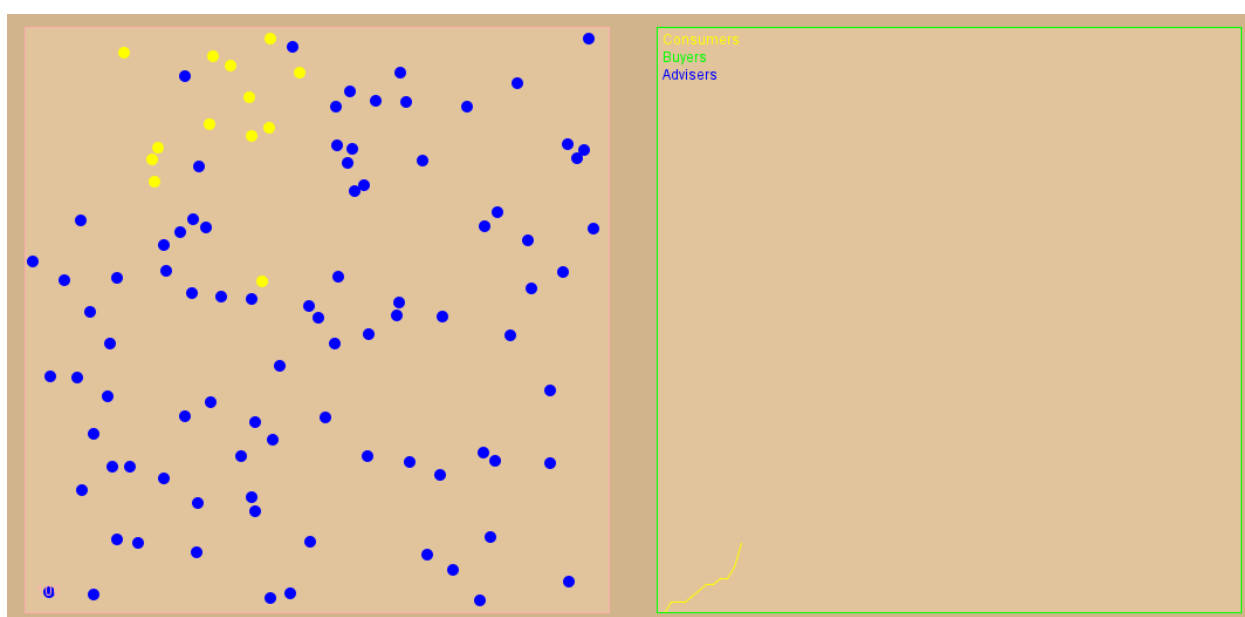


Рис 7.2.2 – Процесс распространения товара

Мы видим, что покупатели начинают менять цвет, о товаре начинают узнавать люди, и это можно увидеть на графике.

На Рисунке 7.3.3 показано конечное состояние.

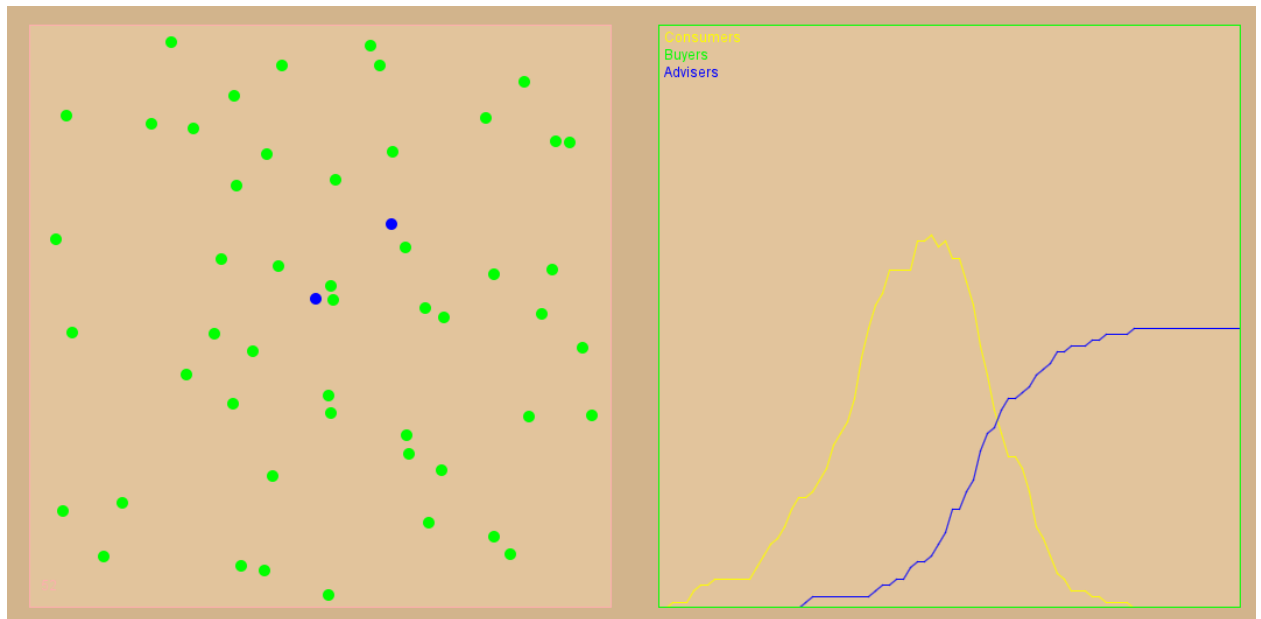


Рис 7.3.3 – Конечное состояние

Мы видим, что почти все наши покупатели зеленого цвета, это означает, что товар людям понравился, и они снова его покупают. Однако, у нас остались люди, которые советовали товар на ранних этапах, это показатель того, что товар еще будет рекомендоваться другим.

Код реализации программы представлен в Листинге А.

8 ПРАКТИЧЕСКАЯ РАБОТА №8

8.1 Постановка задачи

Имеется популяция людей, которая постепенно заражается вирусом, а затем полностью выздоравливает.

Имеем три накопителя, в которых показывается число заболевших, выздоровевших, а также летальные исходы. Имеются начальные условия, при которых число не заболевших равно числу людей в популяции минус единица. Число латентно зараженных и выздоровевших равно нулю. Число заболевших равно 1.

При реализации задачи для вывода промежуточных данных необходимо использовать объект таблицы и желательно объект диаграмму, а также кнопка, которая осуществляет запуск программы.

8.2 Описание процесса создания проекта и его реализация

Изначально, у нас есть 3 группы людей:

- Болеющие;
- Выздоровевшие;
- Летальные исходы.

Программа генерирует графики и статистику относительно этих групп, и показывает сколько людей находилось в определенном состоянии в какой-то день.

Для реализации на языке Java были созданы классы:

- Main – объединяет в себе работу всех классов, внешний вид и запуск проекта;
- TimeSeriesDataPacket – отвечает за отображение все стран;
- DateComparator – отвечает за контроль даты;
- CSVUtilities – отвечает за чтение данных с файлов, данные со статистикой;

- FileUtil – оповещает о том, что работа выполнена успешно и все данные были успешно проанализированы;
- Printer – отвечает за соответствие календарной даты и страны;
- Utilities – отвечает за диаграммы.

Запуск проекта представлен на Рисунке 8.2.1 – Рисунке 8.2.6.

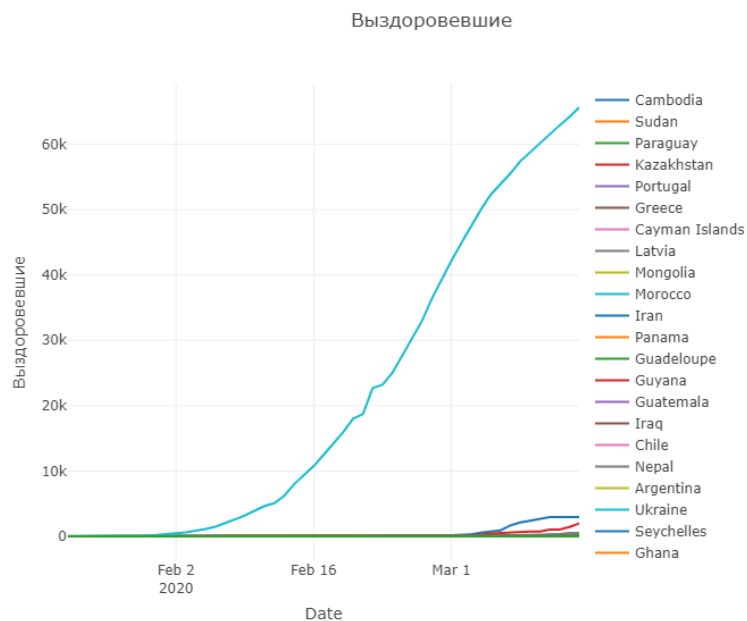


Рис 8.2.1 – Диаграмма состояния «Выздоровевшие»

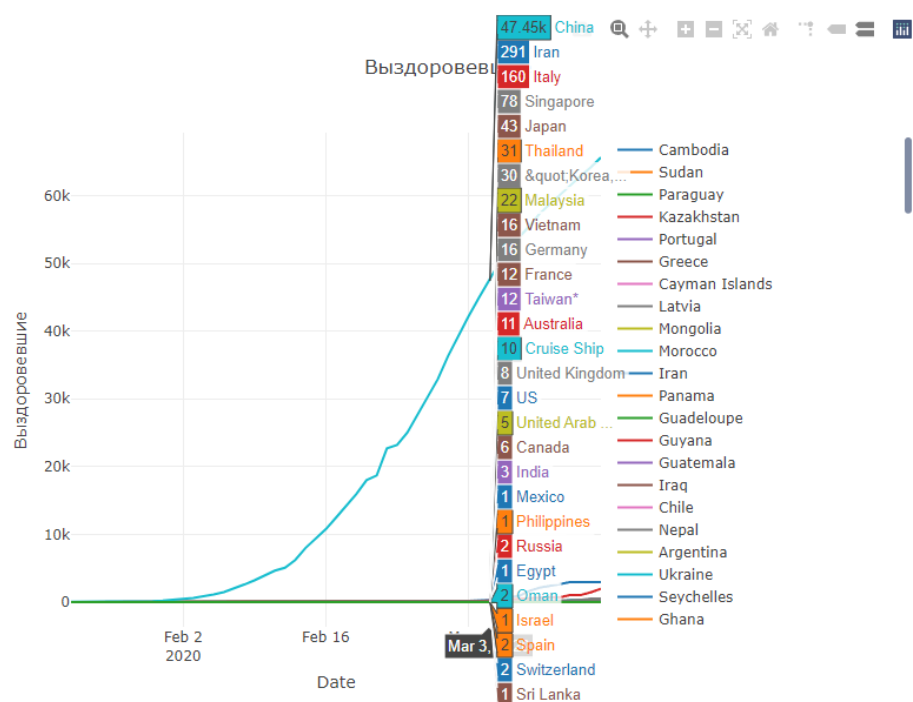


Рис 8.2.2 – Статистика состояния «Выздоровевшие»

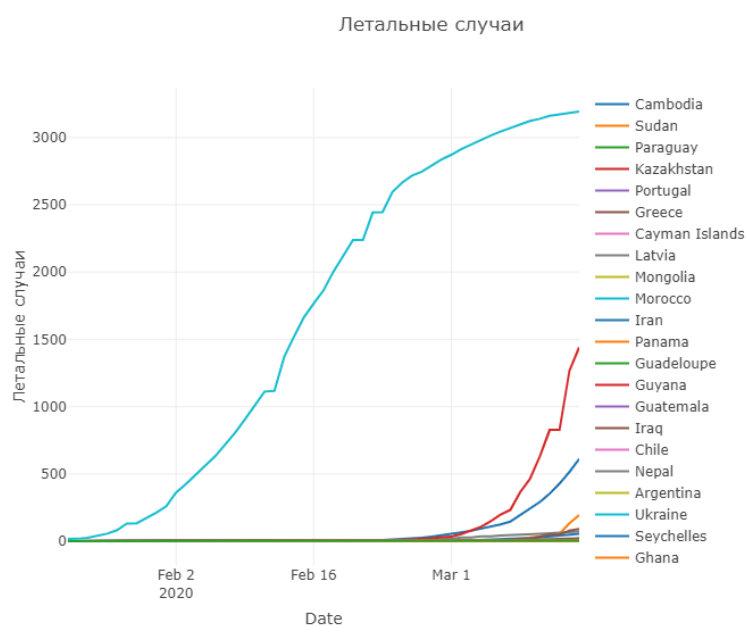


Рис 8.2.3 – Диаграмма состояния «Летальные исходы»

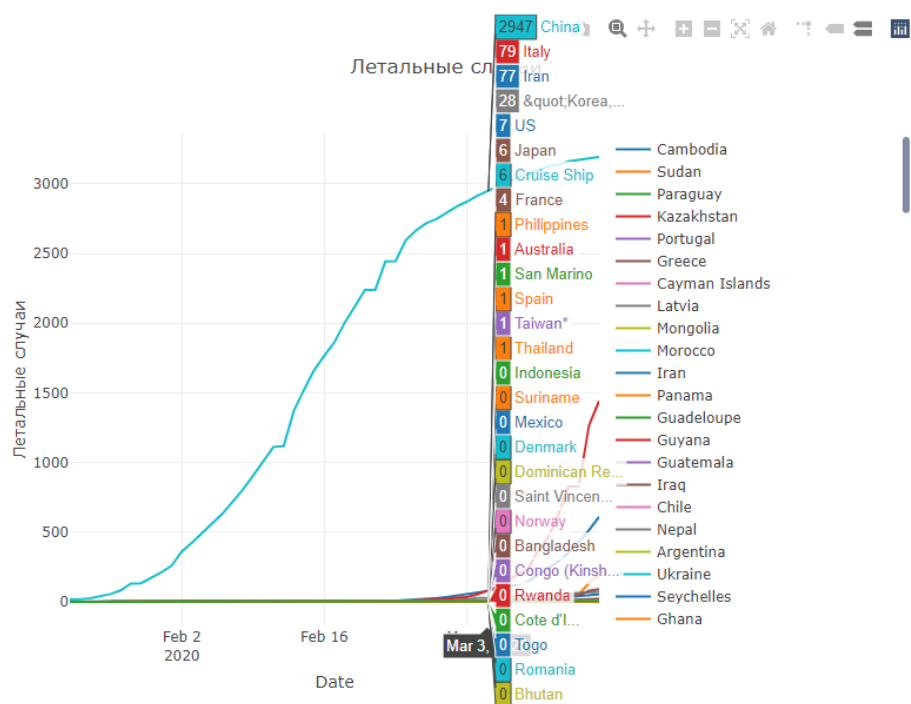


Рис 8.2.4 – Статистика состояния «Летальные исходы»

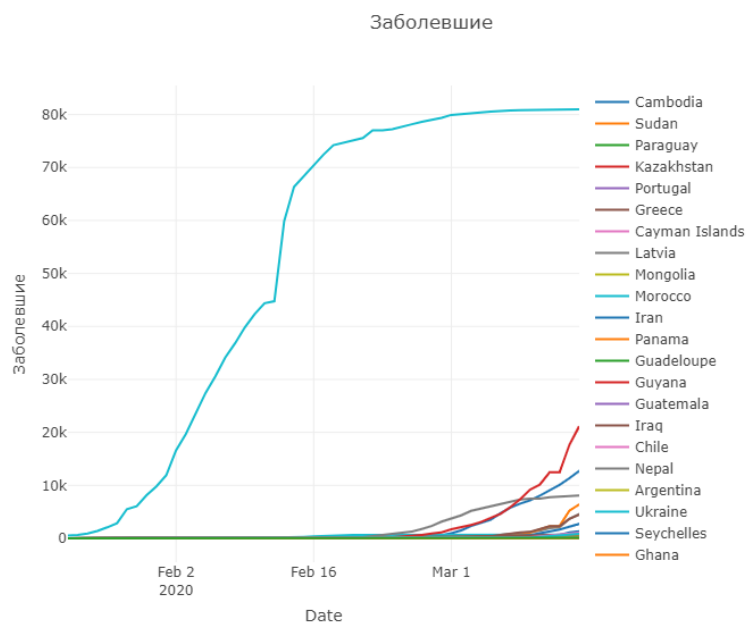


Рис 8.2.5 – Диаграмма состояния «Заболевшие»

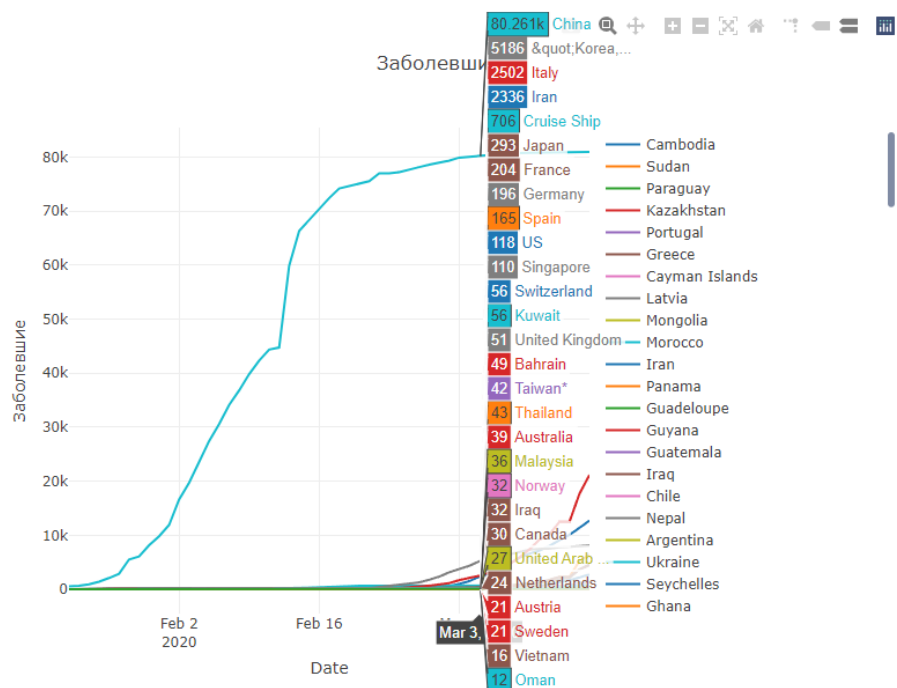


Рис 8.2.6 – Статистика состояния «Заболевшие»

Код реализации программы представлен в Листинге Б.

9 ПРАКТИЧЕСКАЯ РАБОТА №9

9.1 Постановка задачи

Имеются объекты 2 лифта, 5 этажей, и люди, которые передвигаются между этажами.

Изначально, лифт располагается на первом этаже, но после вызова, он начинает подниматься на нужный ему этаж. Люди представлены с 3 видами данных:

- Вес;
- Этаж, на котором они были изначально;
- Этаж, на который им необходимо попасть.

Данные с весом мы используем, так как обычно, в лифте стоят ограничения по весу, до 1000 кг. Соответственно, лифт не будет двигаться, если вес будет превышать эту цифру. Но сам процесс будет идти регулярно, либо же до того момента, пока вы не остановите процесс.

9.2 Описание процесса создания проекта и его запуск

В данной работе у нас есть 5 классов:

- Main – отвечает за запуск приложения, а также рабочие кнопки «Старт» и «Стоп»;
- World – отвечает за строение программы, а именно 2 лифта, этажи, а также состояния работы, т.е. во время работы программы, будет прописываться вся хронология событий;
- Passenger – отвечает за пассажиров, за их вес, и направление;
- Floor – отвечает за конструирование этажей;
- Elevator – отвечает за движение лифта.

Реализация проекта представлена на Рисунке 9.2.1 – Рисунке 9.2.3.

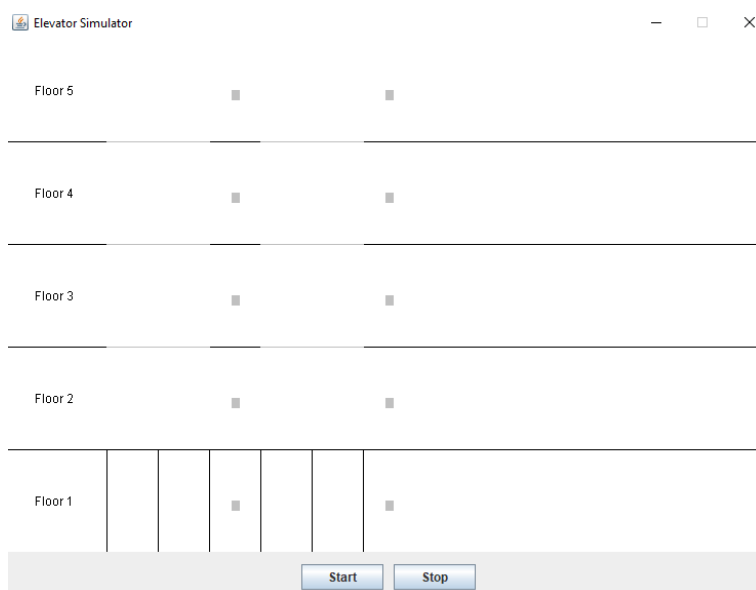


Рис 9.2.1 – Первоначальное состояние

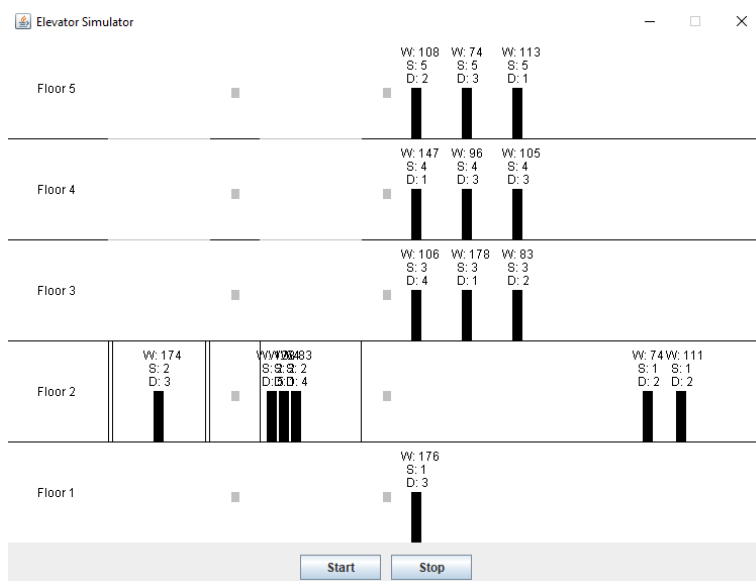


Рис 9.2.2 – Погрузка людей

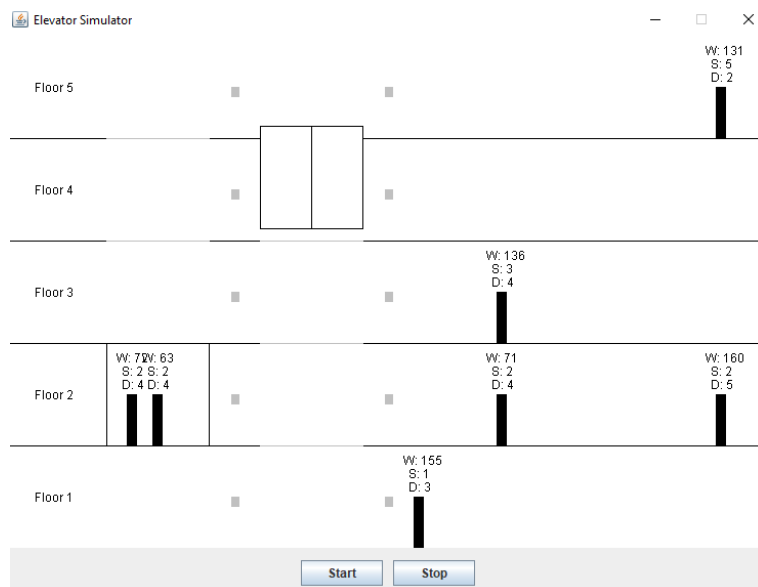


Рис 9.2.3 – Движение лифта

Код реализации программы представлен в Листинге В.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы были изучены различные варианты многоагентного моделирования, начиная с имитационного моделирования в AnyLogic, заканчивая моделированием на различных языках программирования.

Из этого можно сказать, что это та сфера, которая активно сейчас развивается. Она помогает нам не только визуально представить какой-то процесс, но и посмотреть на него со стороны. А также, позволяет, изменить условия и посмотреть на работу системы, с другой стороны реализации. Это гораздо упрощает работу как компаниям, так и студентам.

В заключении могу сказать, что мне посчастливилось получить знания не только в области моделирования, но и в области анализа. Так как при создании модели мы активно анализировали свою работу, и предлагали наилучшие варианты решения.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Имитационное моделирование: практикум [Электронный ресурс]: Учебно-методическое пособие / Тихвинский В.И., Сорокин А.Б. — М.: Московский технологический университет (МИРЭА), 2018.
2. Многоагентное моделирование: теория [Электронный ресурс]: Учебно-методическое пособие / Тихвинский В.И. — М.: Московский технологический университет (МИРЭА), 2020.

ПРИЛОЖЕНИЯ

Листинг А – Реализация практической работы №7

Листинг Б – Реализация практической работы №8

Листинг В – Реализация практической работы №9

Листинг А

Листинг 1 – Класс App

```
package simulation;

import entities.City;
import settings.Constants;
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class App extends JFrame {

    public static void main(String[] args) {
        App app = new App();
        app.init();
    }

    // -----
    ----- //

    private final Timer timer;
    private final CityPanel cityPanel;
    private final GraphPanel graphPanel;

    int counter = 0;
    boolean started = false;

    private App() {
        super("Virus Simulation");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setExtendedState(JFrame.MAXIMIZED_BOTH);
        setUndecorated(true);

        addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                if (!started) {
                    started = true;
                    cityPanel.getCity().getPeople()[0].setConsumer();
                }
            }
        });
    }
}
```

```

        }

    }

});

this.cityPanel = new CityPanel(new City(Constants.CITY_SIZE));
this.graphPanel = new GraphPanel(cityPanel.getCity());
this.timer = new Timer(25, e -> {
    this.cityPanel.getCity().update();
    this.cityPanel.repaint();

    if (counter++ < 2 || (started && counter %
Constants.GRAPH_UPDATE_PER_FRAMES == 0)) {
        this.graphPanel.update();
        this.graphPanel.repaint();
    }
});

this.cityPanel.setPreferredSize(new Dimension(500, 500));
this.graphPanel.setPreferredSize(new Dimension(500, 500));

cityPanel.setBackground(Color.decode("#e2c49c"));
graphPanel.setBackground(Color.decode("#e2c49c"));

JPanel mainPanel = new JPanel(new GridBagLayout());
mainPanel.setBackground(Color.decode("#d2b48c"));

GridBagConstraints x = new GridBagConstraints();
x.insets = new Insets(20, 20, 20, 20);

mainPanel.add(cityPanel, x);

x.gridx = 1;
mainPanel.add(graphPanel, x);

add(mainPanel);
}

private void init() {
    pack();
    setLocationRelativeTo(null);
    setVisible(true);
    timer.start();
}

```



```
}  
}
```

Листинг 2 – Класс CityPanel

```
package simulation;  
  
import entities.City;  
import javax.swing.*;  
import java.awt.*;  
import java.util.Arrays;  
  
public class CityPanel extends JPanel {  
  
    private City city;  
    private boolean x = false;  
  
    public CityPanel(City city) {  
        this.city = city;  
    }  
  
    @Override  
    protected void paintComponent(Graphics _g) {  
        super.paintComponent(_g);  
        if(x) {  
            Graphics2D g = (Graphics2D) _g;  
            g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
RenderingHints.VALUE_ANTIALIAS_ON);  
            city.drawPeople(g, getPreferredSize().width);  
  
            g.setColor(Color.pink);  
            g.drawRect(0, 0, getWidth()-1, getHeight()-1);  
  
            g.drawString(Arrays.stream(city.getPeople()).filter(p  
-> !p.isAdviser()).count()+"", 10, getHeight()-15);  
        }else x = true;  
    }  
  
    public City getCity() {  
        return city;  
    }  
  
    public void setCity(City city) {
```

```
        this.city = city;
    }
}
```

Листинг 3 – Класс GraphPanel

```
package simulation;

import entities.City;
import entities.Person;
import settings.Constants;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class GraphPanel extends JPanel {

    private City city;
    private List<Long> consumer = new ArrayList<>();
    private List<Long> adviser = new ArrayList<>();

    private boolean x = false;

    public GraphPanel(City city) {
        this.city = city;
    }

    public void update() {

        consumer.add(Arrays.stream(city.getPeople()).filter(Person::isConsumer).count());

        adviser.add(Arrays.stream(city.getPeople()).filter(Person::isAdviser).count());
    }

    @Override
    public void repaint(Rectangle r) {
        if(consumer.size() > 600)
            super.repaint(r);
    }
}
```

```

    }

    @Override
    protected void paintComponent(Graphics g_) {
        super.paintComponent(g_);
        Graphics2D g = (Graphics2D) g_;
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        if(x) {
            if(consumer.size() < 2)
                return;

            g.setColor(Color.yellow);
            int xUnit = 600 / Constants.GRAPH_SIZE;
            int yUnit = getPreferredSize().height / city.getPeople().length;

            g.setColor(Color.yellow);
            for(int i = 0; i < consumer.size()-1; i++) {
                g.drawLine(i*xUnit, (int) (getPreferredSize().height-
(consumer.get(i)*yUnit)), (i+1)*xUnit, getPreferredSize().height-
(int) (consumer.get(i+1)*yUnit));
            }

            g.setColor(Color.blue);
            for(int i = 0; i < adviser.size()-1; i++) {
                g.drawLine(i*xUnit, (int) (getPreferredSize().height-
(adviser.get(i)*yUnit)), (i+1)*xUnit, getPreferredSize().height-
(int) (adviser.get(i+1)*yUnit));
            }

            g.setColor(Color.green);
            g.drawLine(0, (int) (getHeight()*(1-Constants.SHOP_INDUSTRY)),
getWidth()-1, (int) (getHeight()*(1-Constants.SHOP_INDUSTRY)));

            g.setColor(Color.yellow);
            g.drawString("Consumers", 5, 15);

            g.setColor(Color.green);
            g.drawString("Buyers", 5, 30);

            g.setColor(Color.blue);

```

```

        g.drawString("Advisers", 5, 45);

        g.setColor(Color.green);
        g.drawRect(0, 0, getWidth()-1, getHeight()-1);
    }else x = true;
}

public City getCity() {
    return city;
}
}

```

Листинг 4 – Класс Constants

```

package settings;

public class Constants {

    public static final int GRAPH_SIZE = 100;
    public static final int GRAPH_UPDATE_PER_FRAMES = 15;

    public static final int CITY_SIZE = 100;
    public static final double MAX_SPEED = 0.5;
    public static final double PEOPLE_SIZE = 2;
    public static final int CONSUMERS_TIME = 250;

    public static final double SHOP_CHANCE = 0.50;
    public static final double CONSUM_CHANCE = 0.75;

    public static final double SHOP_INDUSTRY = 1;
}

```

Листинг 5 – Класс City

```

package entities;

import entities.internal.Vector;
import settings.Constants;
import java.awt.*;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@SuppressWarnings("All")

```

```

public class City {

    private final int worldSize;
    private final Person[] people;

    public City(int worldSize) {
        this.worldSize = worldSize;
        this.people = new Person[worldSize];

        for(int i = 0; i < people.length; i++) {
            Vector position = new Vector(Math.random() * worldSize,
Math.random() * worldSize);
            Vector velocity = new Vector(Math.random() * Constants.MAX_SPEED
* 2, Math.random() * Constants.MAX_SPEED * 2);
            velocity.sub(Constants.MAX_SPEED);

            this.people[i] = new Person(position, velocity);
        }
    }

    public void update() {
        updatePeople();
        double percentConsumer =
Arrays.stream(people).filter(Person::isConsumer).count() /
((double)people.length);
        Arrays.stream(people).forEach(p -> p.update(percentConsumer));
    }

    private void updatePeople() {
        // Check world collision
        for(Person p : people) {
            if(p.getPosition().x < Constants.PEOPLE_SIZE && p.getVelocity().x
< 0) {
                p.getVelocity().mult(new Vector(-1, 1));
                p.setChanged(true);
            }

            if(p.getPosition().x > worldSize - Constants.PEOPLE_SIZE &&
p.getVelocity().x > 0) {
                p.getVelocity().mult(new Vector(-1, 1));
                p.setChanged(true);
            }
        }
    }
}

```

```

        if(p.getPosition().y < Constants.PEOPLE_SIZE && p.getVelocity().y
< 0) {

            p.getVelocity().mult(new Vector(1, -1));
            p.setChanged(true);

        }

        if(p.getPosition().y > worldSize - Constants.PEOPLE_SIZE &&
p.getVelocity().y > 0) {
            p.getVelocity().mult(new Vector(1, -1));
            p.setChanged(true);

        }

    }

    // Check person collision
    for(Person p : people) {
        if(p.isChanged() || p.isAdviser())
            continue;

        List<Person> closeBy = Arrays.asList(people).stream().filter(c -
> !c.isChanged() && !c.isAdviser())
            .filter(c -> Vector.dist(p.getPosition(),
c.getPosition()) < Constants.PEOPLE_SIZE)
            .collect(Collectors.toList());

        if(closeBy.size() > 1) {
            Vector temp = closeBy.get(0).getVelocity();
            for(int i = 0; i < closeBy.size()-1; i++) {
                double separation =
Vector.dist(closeBy.get(i).getPosition(), closeBy.get(i+1).getPosition());
                double extra = (Constants.PEOPLE_SIZE - separation)/2;
                Vector v = Vector.sub(closeBy.get(i).getPosition(),
closeBy.get(i+1).getPosition());
                v.normalize();
                v.mult(extra);
                closeBy.get(i).getPosition().add(v);

                closeBy.get(i).setVelocity(closeBy.get(i
+
1).getVelocity());
            }
            double separation = Vector.dist(closeBy.get(closeBy.size()-
1).getPosition(), closeBy.get(0).getPosition());
            double extra = (Constants.PEOPLE_SIZE - separation)/2;

```

```

        Vector v = Vector.sub(closeBy.get(closeBy.size()-
1).getPosition(), closeBy.get(0).getPosition());
        v.normalize();
        v.mult(extra);
        closeBy.get(closeBy.size()-1).getPosition().add(v);

        closeBy.get(closeBy.size()-1).setVelocity(temp);
        closeBy.stream().forEach(c -> c.setChanged(true));

        if(closeBy.stream().filter(x -> x.isConsumer()).count() > 0)
        {
            closeBy.forEach(x -> {
                if(Math.random() < Constants.CONSUM_CHANCE)
                    x.setConsumer();
            });
        }

        // Reset the changed state
        Arrays.stream(people).forEach(p -> p.setChanged(false));
    }

    public void drawPeople(Graphics2D g, int panelSize) {
        double scale = panelSize / ((double)worldSize);
        Arrays.stream(people).filter(p -> !p.isAdviser()).forEach(p -> {
            if(p.isConsumer()) {
                g.setColor(Color.yellow);
            }else if(p.isBuyer()) {
                g.setColor(Color.green);
            }else {
                g.setColor(Color.blue);
            }

            int x = (int)((p.getPosition().x -
Constants.PEOPLE_SIZE/2)*scale);
            int y = (int)((p.getPosition().y -
Constants.PEOPLE_SIZE/2)*scale);
            g.fillOval(x, y, (int)(Constants.PEOPLE_SIZE*scale), (int)
(Constants.PEOPLE_SIZE*scale));
        });
    }
}

```

```

    public int getWorldSize() {
        return worldSize;
    }

    public Person[] getPeople() {
        return people;
    }
}

```

Листинг 6 – Класс Person

```

package entities;

import entities.internal.Vector;
import settings.Constants;
import java.util.Arrays;

public class Person {

    private static final int REGULAR = 0;
    private static final int CONSUMER = 1;
    private static final int BUYER = 2;
    private static final int ADVISER = 3;

    // --- --- --- --- --- --- --- --- --- --- //

    private Vector position;
    private Vector velocity;

    private boolean changed = false;
    private int lifeTime = Constants.CONSUMERS_TIME;
    private int state = 0;

    public Person(Vector position, Vector velocity) {
        this.position = position;
        this.velocity = velocity;
    }

    public void update(double percentInfected) {
        if(isConsumer()) {
            if(--lifeTime <= 0) {

```



```

        state = percentInfected > Constants.SHOP_INDUSTRY ||
Math.random() > Constants.SHOP_CHANCE
        ? ADVISER : BUYER;
    }
}

    position.add(velocity);
}

public Vector getPosition() {
    return position;
}

public void setPosition(Vector position) {
    this.position = position;
}

public Vector getVelocity() {
    return velocity;
}

public void setVelocity(Vector velocity) {
    this.velocity = velocity;
}

public boolean isChanged() {
    return changed;
}

public void setChanged(boolean changed) {
    this.changed = changed;
}

public boolean isConsumer() {
    return state == CONSUMER;
}

public void setConsumer() {
    if(!isBuyer()) {
        state = CONSUMER;
    }
}
}

```

```
public boolean isBuyer() {  
    return state == BUYER;  
}  
  
public void setBuyer() {  
    state = BUYER;  
}  
  
public boolean isAdviser() {  
    return state == ADVISER;  
}  
  
public void setAdviser() {  
    state = ADVISER;  
}  
}
```

Листинг Б

Листинг 1 – Класс CSVUtilities

```
import java.util.ArrayList;
import java.util.List;

public class CSVUtilities {
    private static final char DEFAULT_SEPARATOR = ',';
    private static final char DEFAULT_QUOTE = '"';

    public static List<String> parseLine(String cvsLine) {
        return parseLine(cvsLine, DEFAULT_SEPARATOR, DEFAULT_QUOTE);
    }

    public static List<String> parseLine(String cvsLine, char separators) {
        return parseLine(cvsLine, separators, DEFAULT_QUOTE);
    }

    public static List<String> parseLine(String cvsLine, char separators,
char customQuote) {
        List<String> result = new ArrayList<>();
        assert cvsLine != null;
        if (customQuote == ' ') {
            customQuote = DEFAULT_QUOTE;
        }
        if (separators == ' ') {
            separators = DEFAULT_SEPARATOR;
        }

        StringBuffer curVal = new StringBuffer();
        boolean inQuotes = false;
        boolean startCollectChar = false;
        boolean doubleQuotesInColumn = false;

        char[] chars = cvsLine.toCharArray();
```

```

for (char ch : chars) {
    if (inQuotes) {
        startCollectChar = true;
        if (ch == customQuote) {
            inQuotes = false;
            doubleQuotesInColumn = false;
        } else {
            if (ch == '\"') {
                if (!doubleQuotesInColumn) {
                    curVal.append(ch);
                    doubleQuotesInColumn = true;
                }
            } else {
                curVal.append(ch);
            }
        }
    } else {
        if (ch == customQuote) {
            inQuotes = true;
            if (chars[0] != '"' && customQuote == '\"') {
                curVal.append('"');
            }
            if (startCollectChar) {
                curVal.append('"');
            }
        } else if (ch == separators) {

            result.add(curVal.toString());

            curVal = new StringBuffer();
            startCollectChar = false;
        } else if (ch == '\r') {
            continue;
        } else if (ch == '\n') {
            break;
        } else {
            curVal.append(ch);
        }
    }
}
result.add(curVal.toString());

```

```

        return result;
    }
}

```

Листинг 2 – Класс DateComparator

```

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Comparator;

public class DateComparator implements Comparator<String> {

    public int compare(String date1, String date2) {
        try {
            return new SimpleDateFormat("M/d/yy").parse(date1).compareTo(new
SimpleDateFormat("M/d/yy").parse(date2));
        } catch (ParseException e) {
            throw new IllegalArgumentException(e);
        }
    }
}

```

Листинг 3 – Класс FileUtil

```

import java.io.File;

public class FileUtil {

    private String mainFolderName = Main.getFramework_Name() +
File.separator;

    private Printer printer;

    public FileUtil() {
        this.printer = new Printer();
        this.createMainFolder();
    }

    public File createMainFolder() {
        File folder;
        folder = new File(mainFolderName);
        try {
            if (folder.mkdirs()) {

```

```

        folder.createNewFile();

        this.printer.printConsole(String.format("Main-Folder[%s]
created!", folder.getName()));
    } else {
        this.printer.printConsoleError(String.format("Main-
Folder[%s] cannot created or already exist!", folder.getName()));
    }
} catch (Exception ignored) {}
return folder;
}

public File createFileInFolder(String folderPath, String fileName) {
    this.createFolder(folderPath);
    File file;
    file = new File(getMainFolderName() + folderPath + File.separator +
fileName);
    try {
        if (!file.exists()) {
            file.createNewFile();
            this.printer.printConsole(String.format("File[%s] created!",
file.getName()));
        } else {
            this.printer.printConsoleError(String.format("File[%s]
cannot created or already exist!", file.getName()));
        }
    } catch (Exception ignored) {}
    return file;
}

public File createFolder(String folderName) {
    File folder;
    folder = new File(getMainFolderName() + folderName + File.separator);
    try {
        if (folder.mkdirs()) {
            folder.createNewFile();
            this.printer.printConsole(String.format("Folder[%s]
created!", folder.getName()));
        } else {
            this.printer.printConsoleError(String.format("Folder[%s]
cannot created or already exist!", folder.getName()));
        }
    } catch (Exception ignored) {}
}

```

```

        return folder;
    }

    public File createFile(String fileName) {
        File file;
        file = new File(getMainFolderName() + File.separator + fileName);
        try {
            if(!file.exists()) {
                file.createNewFile();
                this.printer.printConsole(String.format("File[%s] created!",
file.getName()));
            } else {
                this.printer.printConsoleError(String.format("File[%s]
cannot created or already exist!", file.getName()));
            }
        } catch (Exception ignored) {}
        return file;
    }

    private String getMainFolderName() {
        return this.mainFolderName;
    }
}

```

Листинг 4 – Класс Printer

```

import java.text.SimpleDateFormat;
import java.util.Calendar;

public class Printer {

    public Printer() {}

    private String getTime() {
        Calendar calendar = Calendar.getInstance();
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
        return simpleDateFormat.format(calendar.getTime());
    }

    public void printConsole(String message) {
        System.out.println(String.format("[%s] [%s]:   %s",   this.getTime(),
Main.getFramework_Name(), message));
    }
}

```

```

    }

    public void printConsoleSL(String message) {
        System.out.print(String.format("[%s] [%s]:   %s",      this.getTime(),
Main.getFramework_Name(), message));
    }

    public void printConsoleError(String message) {
        System.err.println(String.format("[%s] [%s]:   %s",      this.getTime(),
Main.getFramework_Name(), message));
    }
}

```

Листинг 5 – Класс Utilities

```

public class Utilities {

    private static Printer printer = new Printer();

    public static void printVector(double[] vector) {
        Utilities.printer.printConsoleSL("Vector: [");
        for(int i = 0; i < vector.length; i++) {
            if(i < vector.length - 1) {
                System.out.print(String.format("%s,          ",
Math.round(vector[i])));
            } else {
                System.out.print(String.format("%s]",
Math.round(vector[i])));
            }
        }
        System.out.println();
    }

    public static void printVector(int[] vector) {
        Utilities.printer.printConsoleSL("Vector: [");
        for(int i = 0; i < vector.length; i++) {
            if(i < vector.length - 1) {
                System.out.print(String.format("%s, ", vector[i]));
            } else {
                System.out.print(String.format("%s]", vector[i]));
            }
        }
        System.out.println();
    }
}

```



```
}  
}
```

Листинг 6 – Класс Main

```
import com.workday.insights.timeseries.arima.Arima;  
import com.workday.insights.timeseries.arima.struct.ArimaParams;  
import com.workday.insights.timeseries.arima.struct.ForecastResult;  
import covid.data.TimeSeriesDataPacket;  
import covid.utilities.CSVUtilities;  
import covid.utilities.DateComparator;  
import covid.utilities.FileUtil;  
import covid.utilities.Utilities;  
import tech.tablesaw.api.DateColumn;  
import tech.tablesaw.api.DoubleColumn;  
import tech.tablesaw.api.StringColumn;  
import tech.tablesaw.api.Table;  
import tech.tablesaw.plotly.Plot;  
import tech.tablesaw.plotly.api.TimeSeriesPlot;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
import java.util.*;  
  
public class Main {  
  
    private FileUtil fileUtil;  
  
    public Main() {  
        this.fileUtil = new FileUtil();  
    }  
  
    public void load() throws IOException {  
        // Loading data from csv files.  
        List<TimeSeriesDataPacket> confirmedPackets =  
this.sumToCountry(this.loadTimeSeriesPackets("time_series_covid_19_confirmed.c  
sv"));;
```

```

        List<TimeSeriesDataPacket>                deathPackets                =
this.sumToCountry(this.loadTimeSeriesPackets("time_series_covid_19_deaths.csv"
));

        List<TimeSeriesDataPacket>                recoveredPackets            =
this.sumToCountry(this.loadTimeSeriesPackets("time_series_covid_19_recovered.c
sv"));

        // Filter. Shows only that countries in plot.
        String[] filter = new String[] {
                "Italy",
                "Germany",
                "China",
                "US"
        };

        // Data for prediction
        //          List<List<Double>>                confirmedTrainingData        =
this.extractData(confirmedPackets);
        //          List<List<Double>>                deathTrainingData            =
this.extractData(deathPackets);
        //          List<List<Double>>                recoveredTrainingData        =
this.extractData(recoveredPackets);

        // Plotting data
        // where null: insert filter for plotting custom countries
        this.plot("Заболевшие", "Date", "Заболевшие", confirmedPackets, null);
        this.plot("Летальные случаи", "Date", "Летальные случаи", deathPackets,
null);
        this.plot("Выздоровевшие", "Date", "Выздоровевшие", recoveredPackets,
null);

        // Predict rate for next 10 days.
        TimeSeriesDataPacket                packet                =                this.getPacket("Germany",
confirmedPackets);
        System.out.println(packet);
        assert packet != null;
        this.predict(this.extractData(packet));
        System.out.println();

        packet = this.getPacket("Italy", confirmedPackets);
        System.out.println(packet);
        this.predict(this.extractData(packet));

```

```

        System.out.println();

        packet = this.getPacket("US", confirmedPackets);
        System.out.println(packet);
        this.predict(this.extractData(packet));
    }

    public void predict(List<Double> list) {
        System.out.println(list);

        // Example subList for prediction: Useful to test prediction
        list = list.subList(0, list.size() / 2 + list.size() / 3);

        System.out.println(list);

        double[] array = new double[list.size()];
        for(int i = 0; i < array.length; i++) {
            array[i] = list.get(i);
        }

        // Predict next 10 days
        ForecastResult forecastResult = Arima.forecast_arima(array, 10, new
ArimaParams(1, 3, 1, 1, 1, 0, 0));
        Utilities.printVector(forecastResult.getForecast());
    }

    /**
     * Returns requested TimeSeriesDataPacket
     * @param countryName name of country
     * @param packets list of packets to search in.
     * @return tsdp
     */
    private TimeSeriesDataPacket getPacket(String countryName,
List<TimeSeriesDataPacket> packets) {
        for(TimeSeriesDataPacket packet : packets) {
            if(packet.getCountry().equalsIgnoreCase(countryName)) {
                return packet;
            }
        }
        return null;
    }
}

```

```

/**
 * Extract data from tsdp for prediction.
 * @param packet input tsdp for extracting data.
 * @return extracted data.
 */
private List<Double> extractData(TimeSeriesDataPacket packet) {
    List<Double> data = new ArrayList<>();
    for(String date : packet.getSeries().keySet()) {
        data.add(packet.getSeries().get(date));
    }
    return data;
}

/**
 * Extract data from packets.
 * @param packets input packets for extraction.
 * @return return extracted data.
 */
private List<List<Double>> extractData(List<TimeSeriesDataPacket> packets)
{
    List<List<Double>> matrix = new ArrayList<>();
    for(TimeSeriesDataPacket packet : packets) {
        List<Double> data = new ArrayList<>();
        for(String date : packet.getSeries().keySet()) {
            data.add(packet.getSeries().get(date));
        }
        matrix.add(data);
    }
    return matrix;
}

/**
 * Load and create TimeSeriesDataPacket's (tsdp).
 * @param fileName input file (csv)
 * @return list of tsdp.
 * @throws IOException
 */
private List<TimeSeriesDataPacket> loadTimeSeriesPackets(String fileName)
throws IOException {
    File file = this.fileUtil.createFile(fileName);
    BufferedReader bufferedReader = new BufferedReader(new
    FileReader(file));

```

```

        int idx = 0;
        String line = "";
        List<String> descriptions = new ArrayList<>();
        List<TimeSeriesDataPacket> packets = new ArrayList<>();
        while((line = bufferedReader.readLine()) != null) {
            if(idx++ == 0) {
                descriptions = CSVUtilities.parseLine(line);
            } else {
                List<String> tokens = CSVUtilities.parseLine(line);
                String state = tokens.get(0);
                String country = tokens.get(1);
                double lat = Double.parseDouble(tokens.get(2));
                double lon = Double.parseDouble(tokens.get(3));
                TreeMap<String, Double> series = new TreeMap<>(new
DateComparator());
                for(int i = 4; i < tokens.size(); i++) {
                    series.put(descriptions.get(i),
Double.parseDouble(tokens.get(i)));
                }
                TimeSeriesDataPacket timeSeriesDataPacket = new
TimeSeriesDataPacket(state, country, lat, lon, series);
                packets.add(timeSeriesDataPacket);
            }
        }
        return packets;
    }

    /**
     * Sums data up from a country. As example sum all states from US up to one.
     * @param packets packets to sum up (clean up)
     * @return no duplicates in list (clear packets).
     */
    private List<TimeSeriesDataPacket> sumToCountry(List<TimeSeriesDataPacket>
packets) {
        HashMap<String, List<TimeSeriesDataPacket>> map = new HashMap<>();
        for(TimeSeriesDataPacket packet : packets) {
            List<TimeSeriesDataPacket> mapPackets =
map.get(packet.getCountry());
            if (mapPackets == null) {
                mapPackets = new ArrayList<>();
            }
            mapPackets.add(packet);
        }
    }

```

```

        map.put(packet.getCountry(), mapPackets);
    }
    List<TimeSeriesDataPacket> resultPackets = new ArrayList<>();
    for(String country : map.keySet()) {
        TreeMap<String, Double> series = new TreeMap<>(new
DateComparator());
        for(TimeSeriesDataPacket packet : map.get(country)) {
            packet.getSeries().forEach((k, v) -> series.merge(k, v,
Double::sum));
        }
        TimeSeriesDataPacket resultPacket = new TimeSeriesDataPacket();
        resultPacket.setCountry(country);
        resultPacket.setSeries(series);
        resultPackets.add(resultPacket);
    }
    return resultPackets;
}

/**
 * Plotting data.
 * @param title name of plot
 * @param x name of the xCol
 * @param y name of the yCol
 * @param packets that are represented in the plot
 * @param filterCountries show's only the countries from array. NULL to show
all.
 */
private void plot(String title, String x, String y,
List<TimeSeriesDataPacket> packets, String[] filterCountries) {
    Table table = Table.create();
    table.addColumn(DateColumn.create(x));
    table.addColumn(DoubleColumn.create(y));
    table.addColumn(StringColumn.create("Country"));
    for(String date : packets.get(0).getSeries().keySet()) {
        for(TimeSeriesDataPacket packet : packets) {
            DateColumn strings = (DateColumn) table.column(0);
            DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("M/d/yy");
            strings.append(LocalDate.parse(date, dateTimeFormatter));

            DoubleColumn doubles = (DoubleColumn) table.column(1);
            doubles.append(packet.getSeries().get(date));
        }
    }
}

```

```

        StringColumn country = (StringColumn) table.column(2);
        country.append(packet.getCountry());
    }
}

// Filter
if(filterCountries != null) {
    Table filter =
table.where(table.stringColumn("Country").isin(filterCountries));
    Plot.show(TimeSeriesPlot.create(title, filter, x, y, "Country"));
} else {
    Plot.show(TimeSeriesPlot.create(title, table, x, y, "Country"));
}
}

public static String getFramework_Name() {
    return "COVID";
}

public static void main(String[] args) throws IOException {
    Main loader = new Main();
    loader.load();
}
}

```

Листинг 7 – Класс TimeSeriesDataPacket

```

import java.util.TreeMap;

public class TimeSeriesDataPacket {

    private String state;
    private String country;

    private double latitude;
    private double longitude;

    private TreeMap<String, Double> series;

    public TimeSeriesDataPacket() {}

    public TimeSeriesDataPacket(String state, String country, double lat,
double lon, TreeMap<String, Double> series) {
        this.state = state;
        this.country = country;
        this.latitude = lat;
        this.longitude = lon;
        this.series = series;
    }
}

```

```

    }

    @Override
    public String toString() {
        return "TimeSeriesDataPacket{" +
            "state='" + state + '\'' +
            ", country='" + country + '\'' +
            ", latitude=" + latitude +
            ", longitude=" + longitude +
            ", series=" + series +
            '}';
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public TreeMap<String, Double> getSeries() {
        return series;
    }

    public void setSeries(TreeMap<String, Double> series) {
        this.series = series;
    }
}

```


Листинг В

Листинг 1 – Класс Main

```
import java.awt.Dimension;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Main
{
    private final JFrame frame;
    private final JPanel panel;
    private final JButton btnStart;
    private final JButton btnStop;
    private World world;

    public Main()
    {
        world = new World();
        world.setBounds(0, 0, 750, 500);

        btnStart = new JButton("Start");
        btnStart.setBounds(290, 512, 80, 25);
        btnStart.setFocusPainted(false);
        btnStart.addActionListener((e) -> {
            world.start();
            System.out.println("pressed start");
        });

        btnStop = new JButton("Stop");
```

```

        btnStop.setBounds(380, 512, 80, 25);
        btnStop.setFocusPainted(false);
        btnStop.addActionListener((e) -> {
            world.stop();
            System.out.println("pressed stop");
        });

        panel = new JPanel(null);
        panel.setPreferredSize(new Dimension(750, 550));
        panel.add(world);
        panel.add(btnStart);
        panel.add(btnStop);

        frame = new JFrame("Elevator Simulator");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setAlwaysOnTop(true);
        frame.setResizable(false);
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args)
    {
        Main main = new Main();
    }
}

```

Листинг 2 – Класс World

```

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
import javax.swing.JPanel;
import javax.swing.Timer;

```

```

public class World extends JPanel
{
    private Timer timer;
    private boolean isRunning;
    private int counter;
    private int timeElapsedInSecs;

    private Elevator elevator1;
    private Elevator elevator2;
    private List<Floor> floors;

    private Random random;

    public World(int refreshRate)
    {
        this.isRunning = false;
        this.counter = 0;
        this.timeElapsedInSecs = 0;

        elevator1 = new Elevator(100, 400);
        elevator2 = new Elevator(250, 400);

        floors = Collections.synchronizedList(new ArrayList());

        for (int i = 0; i < 5; i++)
        {
            floors.add(new Floor(100, 200, 250, 350, 100 * (5 - i),
750, i));
        }

        random = new Random();

        timer = new Timer(refreshRate, (e) -> {

            counter += refreshRate;

            if (counter / 1000 == 1)

```

```

        {
            counter = 0;
            ++timeElapsedInSecs;

            if (timeElapsedInSecs % 5 == 0)
            {
                spawn();
            }
        }

        if (isRunning)
        {
            repaint();
        }
    });

    timer.start();
}

public World()
{
    this(10);
}

public void start()
{
    if (!isRunning)
    {
        isRunning = true;
    }
}

public void stop()
{
    if (isRunning)
    {
        isRunning = false;
    }
}

```

```

    }

    public void spawn()
    {
        int numOfPassenger = random.nextInt(3) + 1;

        for (int i = 0; i < numOfPassenger; i++)
        {
            int srcFloor = random.nextInt(5);
            int destFloor;

            do
            {
                destFloor = random.nextInt(5);
            }
            while (destFloor == srcFloor);

            int weight = random.nextInt(120) + 60;

            floors.get(srcFloor).addPassenger(new Passenger(0, 100 *
(5 - srcFloor) - 50, weight, srcFloor, destFloor));
        }
    }

    public void maneuver(Elevator elevator)
    {
        // maneuver() is being called every paint session.
        if (elevator.test1)
        {
            System.out.println("init elev");
            for (Floor floor : floors)
            {
                if (floor.getPassengers().isEmpty())
                {
                    continue;
                }
            }
        }
    }

```

```

        if      (floor.getPassengers().get(0).getX() ==
floor.getPassengers().get(0).getDestPosX())
        {
            elevator.setMode(Elevator.Mode.UP);
            elevator.test1 = false;
            elevator.test2 = true;

            break;
        }
    }

    if (elevator.test2)
    {
        System.out.println("open elev");
        if (elevator.getY() % 100 == 0)
        {
            boolean alightCheck = false;

            for (Passenger passenger : elevator.getPassengers())
            {
                if      (passenger.getDestinationFloor() ==
elevator.getFloor())
                {
                    alightCheck = true;
                }
            }

            if
(floors.get(elevator.getFloor()).getPassengers().isEmpty() &&
!alightCheck)
            {
                elevator.setMode(elevator.getDirection() ==
Elevator.Mode.UP ? Elevator.Mode.UP : Elevator.Mode.DOWN);
            }
            else
            {
                elevator.setMode(Elevator.Mode.OPEN);
            }
        }
    }

```

```

        elevator.test2 = false;
        elevator.test3 = true;
    }
}

if (elevator.isOpen() && elevator.test3)
{
    System.out.println("alight & board");

elevator.alight(floors.get(elevator.getFloor()).getDeparting());

elevator.board(floors.get(elevator.getFloor()).getPassengers());

    if (!elevator.getPassengers().isEmpty())
    {
        elevator.test3 = false;
        elevator.test4 = true;
    }
    else
    {
        elevator.test3 = false;
        elevator.test7 = true;
    }
}

if (elevator.test7)
{
    elevator.setMode(Elevator.Mode.CLOSE);

    if (elevator.isClose())
    {
        elevator.test7 = false;
        elevator.test1 = true;
    }
}

```

```

        if (elevator.test4)
        {
            System.out.println("check if boarders positioned");
            List<Passenger> temp = elevator.getPassengers();

            if (!temp.isEmpty())
            {
                if      (temp.get(temp.size()      -      1).getX()      ==
temp.get(temp.size() - 1).getDestPosX())
                {
                    elevator.setMode(Elevator.Mode.CLOSE);
                    elevator.test4 = false;
                    elevator.test5 = true;
                }
            else //here
            {
                Iterator i = temp.iterator();

                while (i.hasNext())
                {
                    Passenger p = (Passenger) i.next();

                    if (p.getX() < elevator.getX())
                    {
                        i.remove();
                    }
                }
            }
        }
        else
        {
            elevator.test4 = false;
            elevator.test7 = true;
        }
    }

    if (elevator.isClose() && elevator.test5)
    {

```



```

        System.out.println("closed & re-init");

        elevator.setMode(elevator.getDirection()
Elevator.Mode.UP ? Elevator.Mode.UP : Elevator.Mode.DOWN);

        elevator.test5 = false;
        elevator.test6 = true;
    }

    if (elevator.test6)
    {
        System.out.println("transition to cond 2");
        elevator.test6 = false;
        elevator.test2 = true;
        System.out.println("working fine til here");
    }
}

@Override
protected void paintComponent(Graphics g)
{
    super.paintComponent(g);
    this.setBackground(Color.WHITE);

    for (Floor floor : floors)
    {
        floor.draw(g);
    }

    elevator1.draw(g);
    elevator2.draw(g);

    maneuver(elevator1);
    maneuver(elevator2);
}
}

```

Листинг 3 – Класс Passenger

```

import java.awt.Graphics;

public class Passenger
{
    private int xAxis;
    private int yAxis;
    private int width;
    private int height;
    private int weight;
    private int sourceFloor;
    private int destinationFloor;
    private int currentFloor;
    private int destinationPosX;
    private Mode mode;
    private Mode direction;

    public Passenger(int xAxis, int yAxis, int weight, int src, int
dest)
    {
        this(xAxis, yAxis, 10, 50, weight, src, dest);
    }

    public Passenger(int xAxis, int yAxis, int width, int height, int
weight, int src, int dest)
    {
        this.xAxis = xAxis;
        this.yAxis = yAxis;
        this.width = width;
        this.height = height;
        this.weight = weight;
        this.sourceFloor = src;
        this.destinationFloor = dest;
        this.currentFloor = src;
        this.destinationPosX = -1;
        this.mode = Mode.WAIT;
        this.direction = Mode.LEFT;
    }
}

```

```

public void draw(Graphics g)
{
    g.fillRect(xAxis, yAxis, width, height);
    g.drawString("W: " + weight, xAxis - 10, yAxis - 31);
    g.drawString("S: " + (sourceFloor + 1), xAxis - 5, yAxis -
18);
    g.drawString("D: " + (destinationFloor + 1), xAxis - 5, yAxis
- 5);

    if (xAxis == destinationPosX)
    {
        mode = Mode.WAIT;
    }
    else
    {
        mode = direction == Mode.LEFT ? Mode.LEFT : Mode.RIGHT;
    }

    step();
}

public void step()
{
    switch (mode)
    {
        case LEFT:
            --xAxis;
            break;
        case RIGHT:
            ++xAxis;
            break;
        case UP:
            --yAxis;
            break;
        case DOWN:
            ++yAxis;
            break;
        default:

```

```
        break;

    }

}

public void setMode(Mode mode)
{
    this.mode = mode;
}

public void setDirection(Mode direction)
{
    this.direction = direction;
}

public void setX(int x)
{
    this.xAxis = x;
}

public void setY(int y)
{
    this.yAxis = y;
}

public void setDestPosX(int x)
{
    this.destinationPosX = x;
}

public int getX()
{
    return this.xAxis;
}

public int getY()
{
    return this.yAxis;
}
```

```

    public int getDestPosX()
    {
        return this.destinationPosX;
    }

    public int getCurrentFloor()
    {
        return this.currentFloor;
    }

    public int getSourceFloor()
    {
        return this.sourceFloor;
    }

    public int getDestinationFloor()
    {
        return this.destinationFloor;
    }

    public int getWeight()
    {
        return this.weight;
    }

    public enum Mode {LEFT, RIGHT, UP, DOWN, WAIT};
}

```

Листинг 4 – Класс Elevator

```

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class Elevator

```

```

{
    private int xAxis;
    private int yAxis;
    private int width;
    private int height;
    private int doorWidth;
    private int floor;

    private boolean isOpen;
    private boolean isClose;

    private Mode mode;
    private Mode direction;
    private List<Passenger> passengers;

    public boolean test1;
    public boolean test2;
    public boolean test3;
    public boolean test4;
    public boolean test5;
    public boolean test6;
    public boolean test7;

    public Elevator(int xAxis, int yAxis)
    {
        this(xAxis, yAxis, 100, 100, 0);
    }

    public Elevator(int xAxis, int yAxis, int width, int height, int
floor)
    {
        this.xAxis = xAxis;
        this.yAxis = yAxis;
        this.width = width;
        this.height = height;
        this.doorWidth = width / 2;
        this.floor = floor;
        this.isOpen = false;
    }
}

```

```

        this.isClose = true;
        this.mode = Mode.WAIT;
        this.direction = Mode.UP;
        this.passengers = Collections.synchronizedList(new
ArrayList());

        test1 = true;
        test2 = false;
        test3 = false;
        test4 = false;
        test5 = false;
        test6 = false;
        test7 = false;
    }

    public void draw(Graphics g)
    {
        for (Passenger passenger : passengers)
        {
            passenger.draw(g);
        }

        g.drawRect(xAxis, yAxis, width, height);
        g.setColor(Color.WHITE);
        g.fillRect(xAxis, yAxis, doorWidth, height);
        g.fillRect(xAxis + width - doorWidth, yAxis, doorWidth,
height);
        g.setColor(Color.BLACK);
        g.drawRect(xAxis, yAxis, doorWidth, height);
        g.drawRect(xAxis + width - doorWidth, yAxis, doorWidth,
height);
        step();
    }

    public void step()
    {
        switch (mode)
        {

```

```

case UP:

    --yAxis;

    for (Passenger passenger : passengers)
    {
        passenger.setY(passenger.getY() - 1);
    }

    if (yAxis % 100 == 0)
    {
        ++floor;

        if (floor == 4)
        {
            direction = Mode.DOWN;
        }
    }

    break;

case DOWN:

    ++yAxis;

    for (Passenger passenger : passengers)
    {
        passenger.setY(passenger.getY() + 1);
    }

    if (yAxis % 100 == 0)
    {
        --floor;

        if (floor == 0)
        {
            direction = Mode.UP;
        }
    }

```



```

        }

        break;

    case OPEN:

        if (doorWidth > 0)
        {
            --doorWidth;
        }
        else if (doorWidth == 0)
        {
            isOpen = true;
            isClose = false;
        }

        break;

    case CLOSE:

        if (doorWidth < width / 2)
        {
            ++doorWidth;
        }
        else if (doorWidth == width / 2)
        {
            isOpen = false;
            isClose = true;
        }

        break;

    default:

        break;

    }
}

```

```

public void alight(List<Passenger> passengers)
{
    Iterator iterator = this.passengers.iterator();

    while (iterator.hasNext())
    {
        Passenger temp = (Passenger) iterator.next();

        if (temp.getDestinationFloor() == floor)
        {
            passengers.add(temp);
            iterator.remove();
        }
    }
}

public void board(List<Passenger> passengers)
{
    int totalWeight = 0;

    for (Passenger passenger : passengers)
    {
        totalWeight += passenger.getWeight();
    }

    while (this.passengers.size() < 10 && !passengers.isEmpty() &&
totalWeight < 700)
    {
        this.passengers.add(passengers.get(0));
        passengers.remove(0);
    }

    int x = 100 / (this.passengers.size() + 1);

    for (int i = 0, j = xAxis - 5 + x; i < this.passengers.size();
i++, j += x)
    {
        this.passengers.get(i).setDestPosX(j);
    }
}

```

```

    }

}

public boolean isOpen()
{
    return this.isOpen;
}

public boolean isClose()
{
    return this.isClose;
}

public void setMode(Mode mode)
{
    this.mode = mode;
}

public void setDirection(Mode direction)
{
    this.direction = direction;
}

public int getFloor()
{
    return this.floor;
}

public List<Passenger> getPassengers()
{
    return this.passengers;
}

public Mode getMode()
{
    return this.mode;
}

```

```

    public Mode getDirection()
    {
        return this.direction;
    }

    public int getX()
    {
        return this.xAxis;
    }

    public int getY()
    {
        return this.yAxis;
    }

    public enum Mode {UP, DOWN, OPEN, CLOSE, WAIT};
}

```

Листинг 5 – Класс Floor

```

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class Floor
{
    private int x1;
    private int x2;
    private int x3;
    private int x4;
    private int y;
    private int l;
    private int floor;
    private List<Passenger> passengers;
    private List<Passenger> departing;
}

```

```

    public Floor(int x1, int x2, int x3, int x4, int y, int l, int
floor)
    {
        this.x1 = x1;
        this.x2 = x2;
        this.x3 = x3;
        this.x4 = x4;
        this.y = y;
        this.l = l;
        this.floor = floor;
        this.passengers      =      Collections.synchronizedList(new
ArrayList());
        this.departing      =      Collections.synchronizedList(new
ArrayList());
    }

    public void draw(Graphics g)
    {
        g.drawLine(0, y, x1, y);
        g.drawLine(x2, y, x3, y);
        g.drawLine(x4, y, l, y);

        g.setColor(Color.LIGHT_GRAY);

        g.fillRect(222, y - 50, 8, 10);
        g.fillRect(372, y - 50, 8, 10);

        g.drawLine(x1, y, x2, y);
        g.drawLine(x3, y, x4, y);

        g.setColor(Color.BLACK);

        g.drawString("Floor " + (floor + 1), 30, y - 45);

        for (int i = 0, j = 400; i < passengers.size(); i++, j += 50)
        {
            passengers.get(i).setDestPosX(j);
            passengers.get(i).draw(g);
        }
    }

```

```

    }

    Iterator iterator = departing.iterator();

    while (iterator.hasNext())
    {
        Passenger temp = (Passenger) iterator.next();

        temp.setDestPosX(775);
        temp.setDirection(Passenger.Mode.RIGHT);
        temp.draw(g);

        if (temp.getX() == temp.getDestPosX())
        {
            iterator.remove();
        }
    }
}

public void addPassenger(Passenger passenger)
{
    int k = (!passengers.isEmpty()) ?
passengers.get(passengers.size() - 1).getX() + 750 : 750;

    passenger.setX(k);
    passengers.add(passenger);
}

public int getFloor()
{
    return this.floor;
}

public List<Passenger> getPassengers()
{
    return this.passengers;
}

```

```
public List<Passenger> getDeparting()  
{  
    return this.departing;  
}  
}
```