



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информатики и прикладной математики
Кафедра прикладной математики и экономико-математических методов

ОТЧЁТ
по дисциплине:
«Имитационное моделирование»

Студента: Бронникова Егора Игоревича

Курс: 4

Группа: ПМ-1901

Форма обучения: очная

Форма представления на кафедру выполненных заданий:
отчёт в электронной форме

Оценка по результатам текущего
контроля (КТ№3)

Санкт-Петербург
2022

Содержание

| | |
|--|-----------|
| Пешеходная библиотека | 2 |
| Пешеходная библиотека. Первая модель | 2 |
| Проект «Станция метро» | 4 |
| Внешние данные. Табличные функции | 7 |
| Чтение из Excel – фигуры | 7 |
| Фигуры из презентации в файл MS Excel | 9 |
| Системная динамика | 11 |
| Модель развития социального стресса | 11 |
| Агентные модели. Дискретное пространство | 14 |
| «Жизнь» Конвея | 14 |
| Модель сегрегации Шеллинга в AnyLogic | 16 |
| Модель сегрегации Шеллинга в Python | 19 |
| Диаграммы состояний и события | 23 |
| Светофор | 23 |
| SIR – агентная модель | 25 |
| Ноутбук + зарядка | 28 |
| SIERD + вакцинация | 30 |
| Диаграмма действий | 32 |
| Сумма ряда – экспонента | 32 |
| Сумма ряда – синус | 34 |
| Агентные модели | 36 |
| Подготовка к зачёту | 36 |
| Связи агентов – добавление, удаление, список, количество | 39 |
| Работа с ГИС-картами | 44 |
| Доставка с покупками | 44 |

Пешеходная библиотека

Пешеходная библиотека. Первая модель

Задание:

Разработать модель движения пассажиров в наземном павильоне метро.

Решение:

Перед тем, как пройти к поездам метро, пассажиры проходят через турникеты, проверяющие наличие проездного документа. Некоторые пассажиры должны будут вначале приобрести жетоны или проездные билеты в кассе. (Рисунок 1)

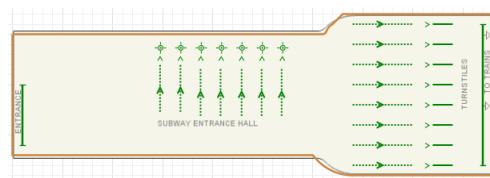


Рис. 1: Разметка пространства наземного павильона метро

Далее была разработана основная логика модели. (Рисунок 2)

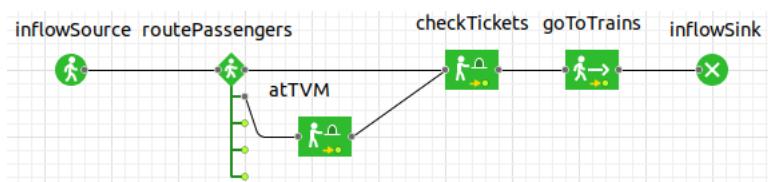


Рис. 2: Описание логики модели

Агенты поступают в модель согласно заданной интенсивности, затем у них есть выбор либо пойти в кассы и купить билет, либо сразу пойти к турникетам. Выбор осуществляется на основе заданной вероятности в 70%. (Рисунок 3)

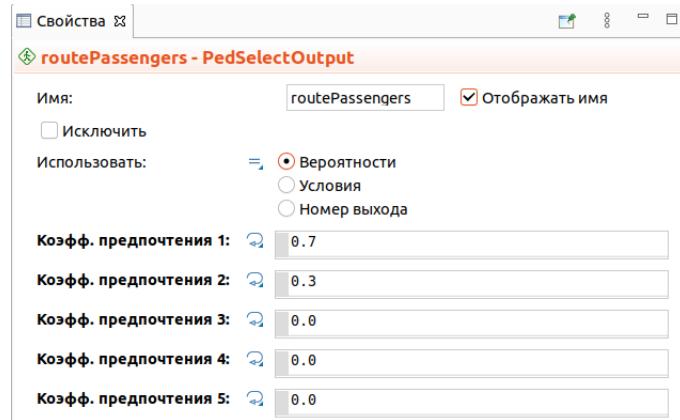


Рис. 3: Принятие решения в модели

Блоки обслуживания агентов задаются идентичным образом как это было в Библиотеке моделирования процессов. Выход из модели осуществляется путём достижения линии выхода.

Для отслеживания возникновения мест с большим скоплением агентов используем карту плотности, тогда модель будет выглядеть следующим образом. (Рисунок 4)

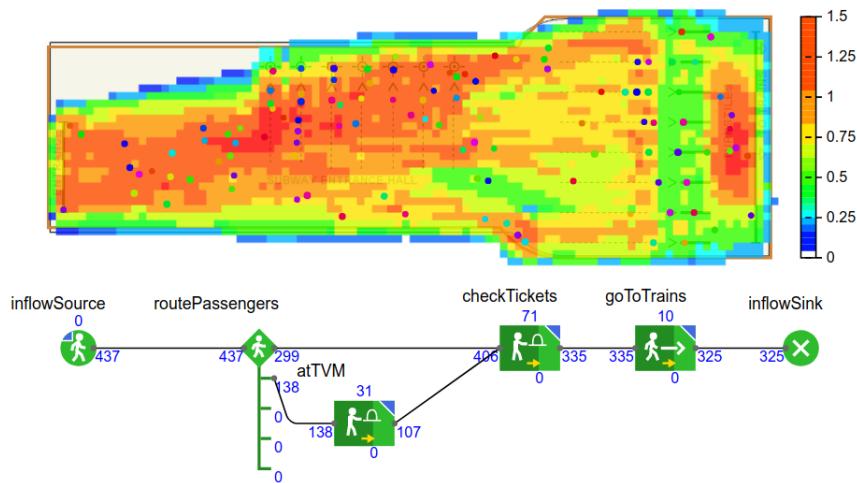


Рис. 4: Принятие решения в модели

Таким образом, на примере модели метро, нами был рассмотрен инструментарий пешеходной библиотеки.

Проект «Станция метро»

Задание:

Построить и проанализировать пешеходную имитационную модель станции метро Озерки.

Решение:

Имеется следующая схема станции метро Озерки. (Рисунок 5)

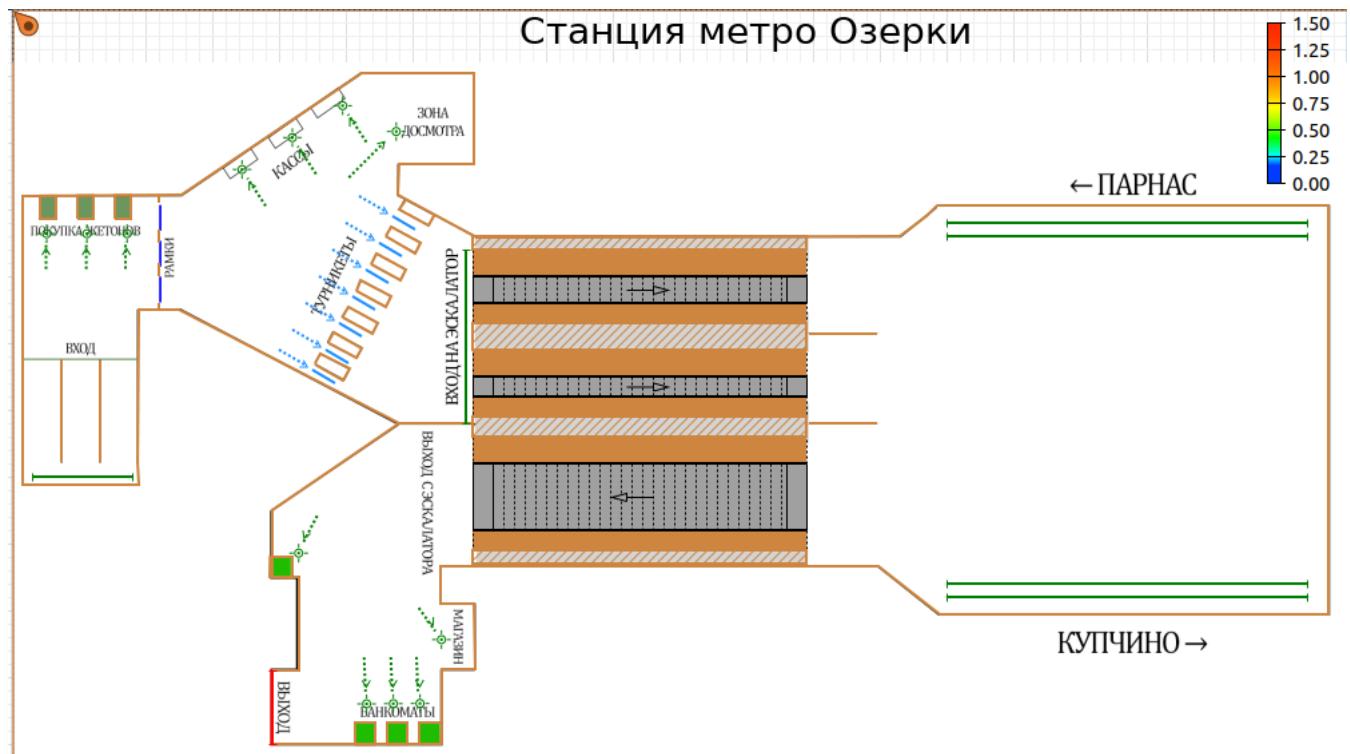


Рис. 5: Схема станции метро Озерки

При входе пассажиры могут купить жетоны либо в специальных автоматах, либо пойти на кассу, также их могут остановить на досмотр. После чего пассажиры проходят через турникеты и спускаются по эскалатору, далее они выбирают направление движения и садятся на поезд.

Соответственно, пассажиры, которые прибывают на станцию метро с других направлений, могут пройти по эскалатору на верх. После того как они поднялись, у них есть выбор пойти в группу банкоматов, пойти в «непопулярный» банкомат или зайти в магазин, также в магазине они смотрят на товар и в случае, если там нет нужного им продукта покинуть магазин или купить что-то. После всех данных альтернатив пассажиры покидают станцию метро.

В соответствии с описанием данная модель была реализована в среде моделирования *AnyLogic* (Рисунок 6).

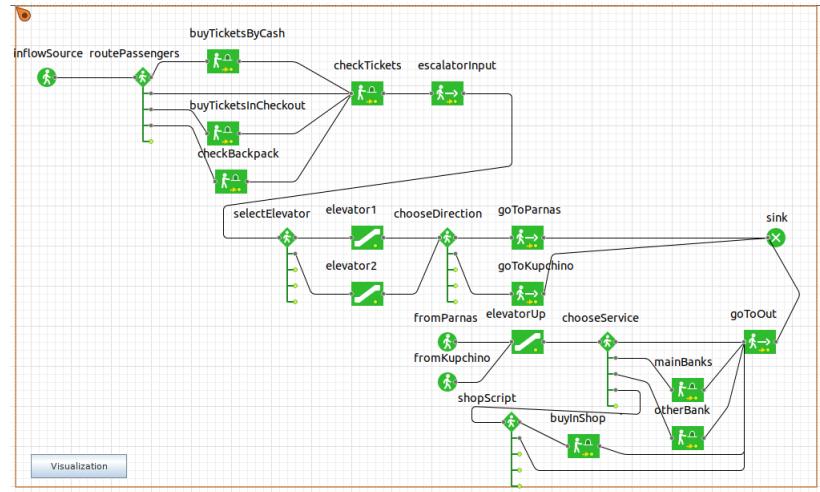


Рис. 6: Модель в среде *AnyLogic*

Данная модель не имеет модификаций с изменением направления эскалатора и имеет статический поток интенсивности пассажиров.

Также в соседнем окне была построена визуализация модели и тепловая карта, которая соответствует плотности различных участков станции. (Рисунок 7)

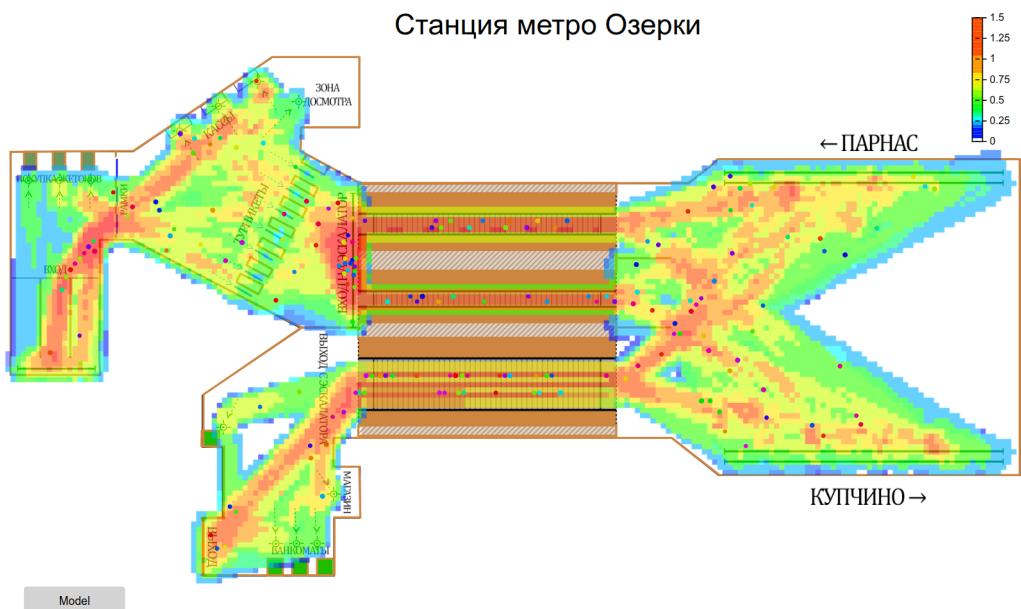


Рис. 7: Модель в среде *AnyLogic*

На данной тепловой карте можно видеть, что если средняя интенсивность пассажиропотока составляет 2000 человек в час, то «узким горлышком» на станции служит вход до рамок металлодетектора и входа на спуск по эскалатору.

Таким образом, нами была построена модель станции метро Озерки и была проанализирована зависимость плотности от различных факторов.

Внешние данные. Табличные функции

Чтение из Excel – фигуры

Задание:

Построить фигуры, считав данные о типах фигур и их параметрах из файла ФигурыExcel.xlsx (для отрезков указаны координаты концов, для прямоугольников – координаты левого верхнего угла).

Решение:

Для того, чтобы связать Excel файл с моделью, необходимо использовать блок – файл Excel. Для загрузки была создана кнопка *Load*. (Рисунок 8)

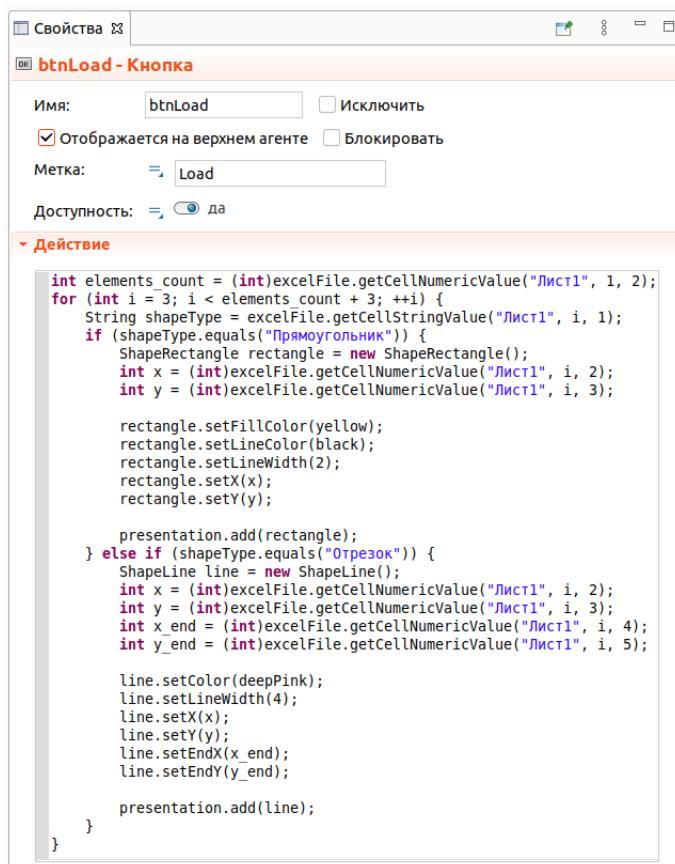


Рис. 8: Логика кнопки для загрузки данных из файла

При нажатии на данную кнопку происходит отрисовка фигур заданных типов с заданными параметрами. (Рисунок 9)



Рис. 9: Результат нажатия на кнопку *Load*

Таким образом, по данным из excel файла нами были построены необходимые фигуры, тем самым нами был рассмотрен процесс чтения данных из excel файла.

Фигуры из презентации в файл MS Excel

Задание:

Поместить в область просмотра несколько фигур из палитры Презентация. Запишите в файл Excel данные о размещённых фигурах – тип фигуры и параметры.

Решение:

Для того, чтобы связать Excel файл с моделью, необходимо использовать блок – файл Excel. Далее были расставлены элементы презентации. Для выгрузки была создана кнопка *Save*. (Рисунок 10)

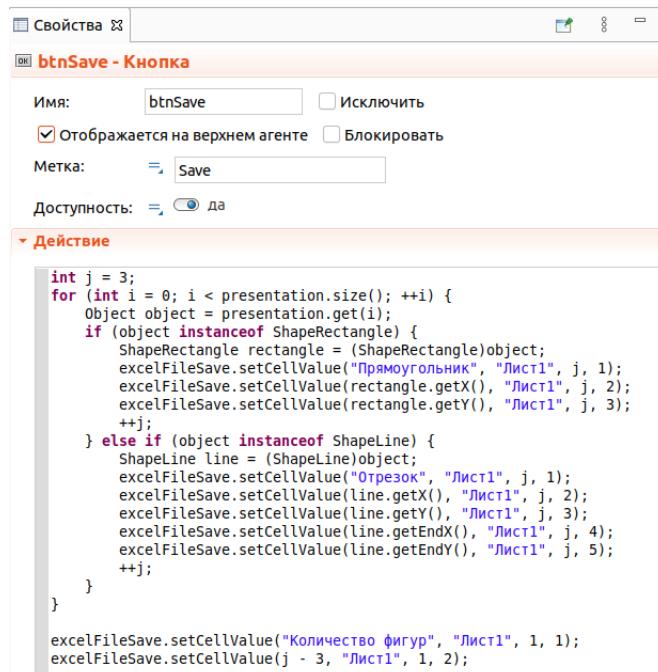


Рис. 10: Логика кнопки для выгрузки данных из файла

При нажатии на данную кнопку происходит сохранение фигур заданных типов с заданными параметрами. (Рисунок 11)



Рис. 11: Результат нажатия на кнопку *Save*

| | A | B | C | D | E | F |
|----|------------------|-----|-----|-----|-----|---|
| 1 | Количество фигур | 18 | | | | |
| 2 | | | | | | |
| 3 | Прямоугольник | 50 | 350 | | | |
| 4 | Прямоугольник | 200 | 350 | | | |
| 5 | Прямоугольник | 200 | 430 | | | |
| 6 | Прямоугольник | 50 | 430 | | | |
| 7 | Прямоугольник | 125 | 390 | | | |
| 8 | Отрезок | 80 | 380 | 125 | 390 | |
| 9 | Отрезок | 155 | 390 | 200 | 380 | |
| 10 | Отрезок | 80 | 430 | 125 | 420 | |
| 11 | Отрезок | 155 | 420 | 200 | 430 | |
| 12 | Прямоугольник | 50 | 50 | | | |
| 13 | Отрезок | 100 | 100 | 125 | 125 | |
| 14 | Прямоугольник | 125 | 125 | | | |
| 15 | Прямоугольник | 200 | 50 | | | |
| 16 | Прямоугольник | 50 | 200 | | | |
| 17 | Прямоугольник | 200 | 200 | | | |
| 18 | Отрезок | 200 | 100 | 175 | 125 | |
| 19 | Отрезок | 100 | 200 | 125 | 175 | |
| 20 | Отрезок | 200 | 200 | 175 | 175 | |
| 21 | | | | | | |

Рис. 12: Получившийся набор данных

Таким образом, по данным из презентации нами были сформированы данные excel файла, тем самым нами был рассмотрен процесс экспорта данных в excel файл.

Системная динамика

Модель развития социального стресса

Задание:

Реализовать и проанализировать модель развития социального стресса.

Решение:

Имеется три фазы развития психологического стресса:

$$\begin{cases} \frac{dN_1}{dt} = -\alpha VN_1 N_2 - pVN_1 + qN_3 \\ \frac{dN_2}{dt} = \alpha VN_1 N_2 + pVN_1 - \beta N_2 \\ \frac{dN_3}{dt} = \beta N_2 - qN_3 \\ \frac{dV}{dt} = (cN_1 - r - m_0)V \end{cases}$$

V – степень «эмоциональной выраженности» информации, обусловленной стрессовым фактором.

Механизмы психологического давления:

1. $r = m \cdot N_3$
2. $r = m \cdot N_2$
3. $r = m \cdot (N_2 + N_3)$

Покажем поведение модели при начальных значениях $N_1(0) = 995$, $N_2(0) = 5$, $V(0) = 1$, $\alpha = 0.0005$, $q = 0.05$, $b = 1/15$, $c = 0.0001$, $m = 0.00005$, $m_0 = 0.05$, $H = 1000$, $p = 0.017$. (Рисунок 13)

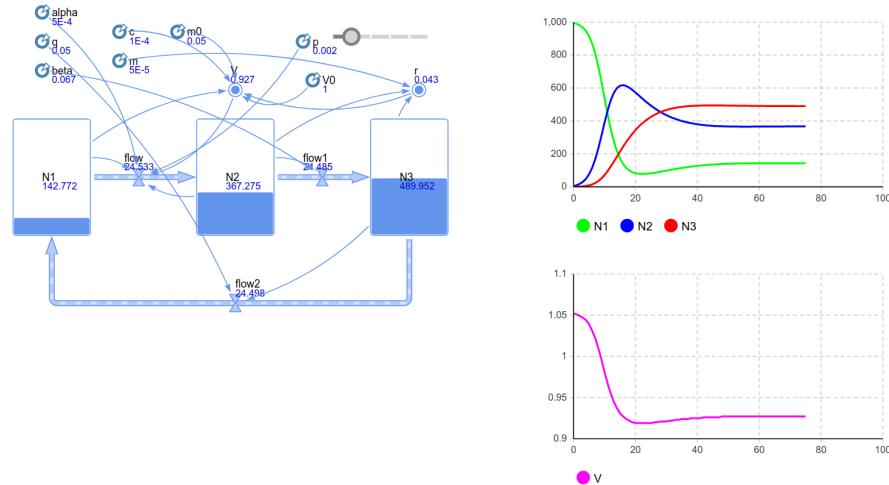


Рис. 13: Модель развития социального стресса при $p = 0.0017$

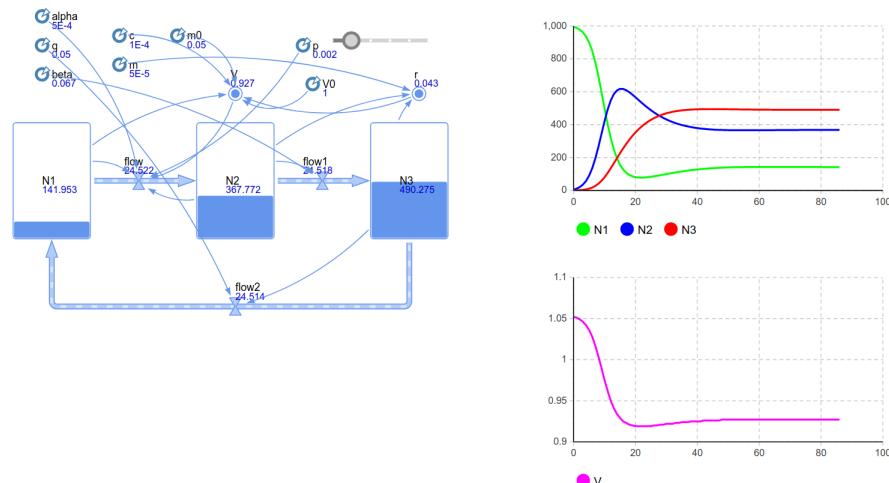


Рис. 14: Модель развития социального стресса при $p = 0.00246$

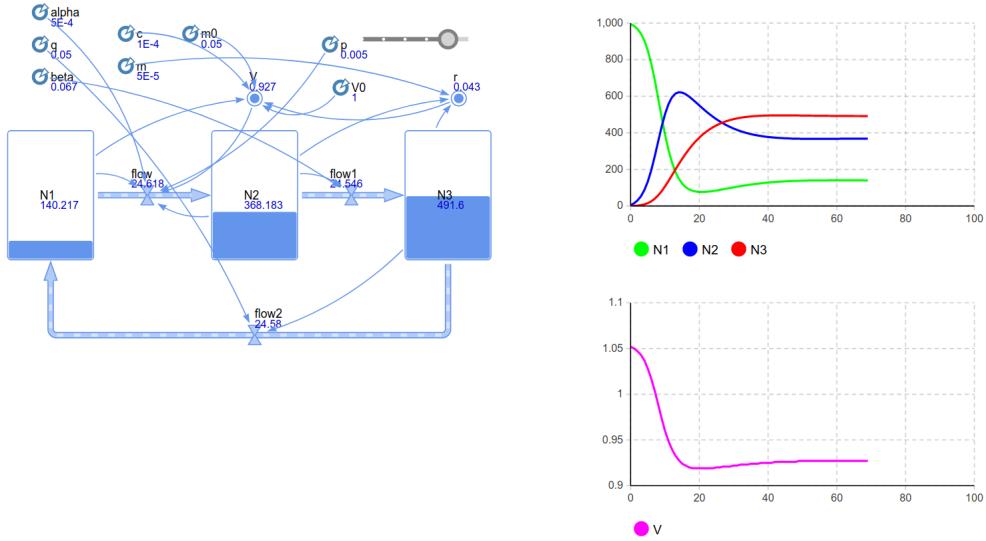


Рис. 15: Модель развития социального стресса при $p = 0.00534$

Из графиков можно видеть, что чем больше значение p , тем интенсивнее происходят изменения в составе групп. Также данная модель с течением времени приходит к стационарному состоянию из-за того, что V примет значение, при котором численность групп будет оставаться на том же уровне.

Таким образом, можно вывести закономерность, связанную с изменением численности групп подверженных стрессу. С ростом степени «эмоциональной выраженности» информации, обусловленной стрессовыми факторами, увеличивается численность людей в генерализированной стадии, с ростом численности данной группы увеличивается численность людей в восстановительной стадии, с ростом численности данной группы уменьшается степень «эмоциональной выраженности». С уменьшением степени «эмоциональной выраженности» возрастает численность людей в начальной стадии и уменьшается численность оставшихся групп.

Таким образом, была реализована модель развития социального стресса, был проведён численный анализ и были проанализированы взаимосвязи между переменными и параметрами модели.

Агентные модели. Дискретное пространство

«Жизнь» Конвея

Задание:

Реализовать модель «Game Of Life»

Решение:

Сначала была создана популяция размером – 2500 агентов, заданная в дискретном пространстве и имеющую тип соседства – Мурово. (Рисунок 16)

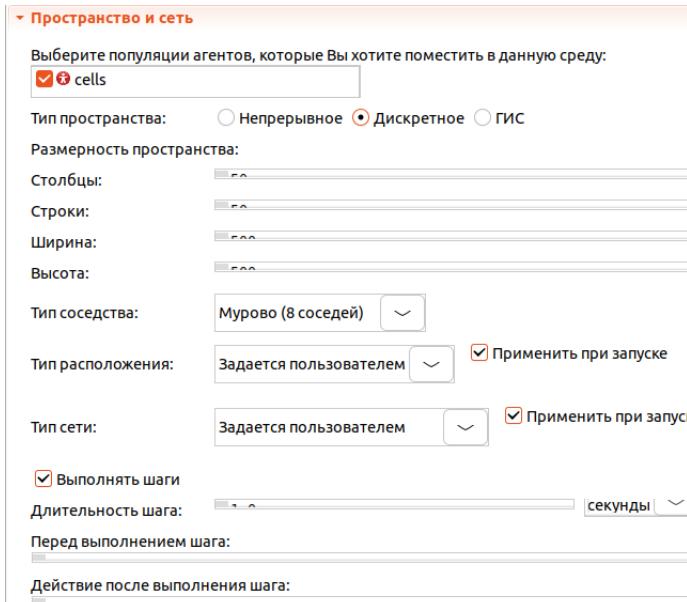


Рис. 16: Настройка популяции агентов

Цель данной модели заключается в том, что изначально каждой клетке присваивается состояние – жива или мёртва. Далее это состояние изменяется, если рядом с живой клеткой находится меньше двух или больше трёх живых клеток, то она умирает. Мёртвая клетка, рядом с которой находится ровно три живые клетки, оживает.

Для того чтобы реализовать данный алгоритм нужно перейти в популяцию агентов и задать действия перед выполнением шага и задать действие на каждом шаге. (Рисунок 17)

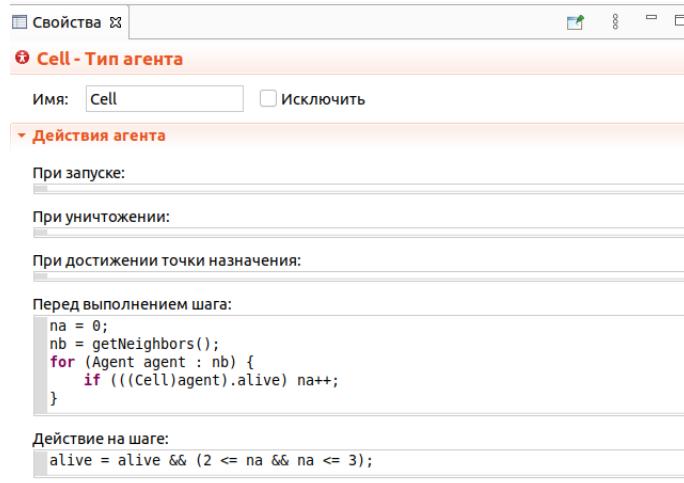


Рис. 17: Реализация алгоритма Игры в жизнь

В результате получается модель, в которой в соответствии с описанным алгоритмом меняются состояния клеток. Также можно поменять состояние клетки шёлкнув по ней. (Рисунок 18)

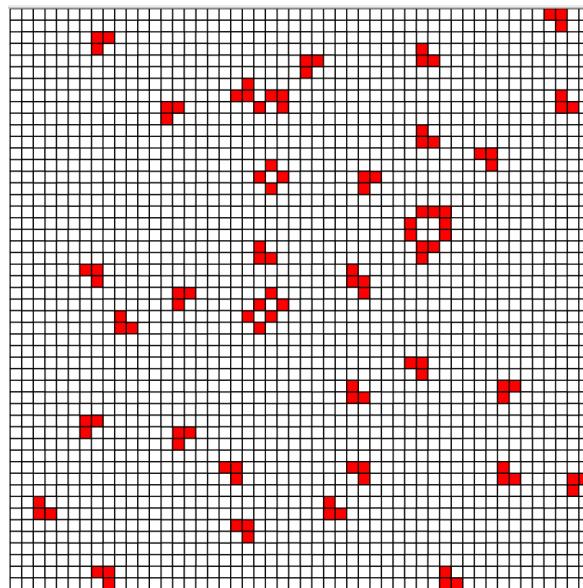


Рис. 18: Работа алгоритма Игры в жизнь

Таким образом, была реализована модель «жизнь» Конвея, на примере которой мы познакомились с агентными моделями в дискретном пространстве.

Модель сегрегации Шеллинга в AnyLogic

Задание:

Реализовать модель сегрегации Шеллинга в AnyLogic.

Решение:

Сначала была создана популяция размером – 9000 агентов, заданная в дискретном пространстве и имеющую тип соседства – Мурово. (Рисунок 19)

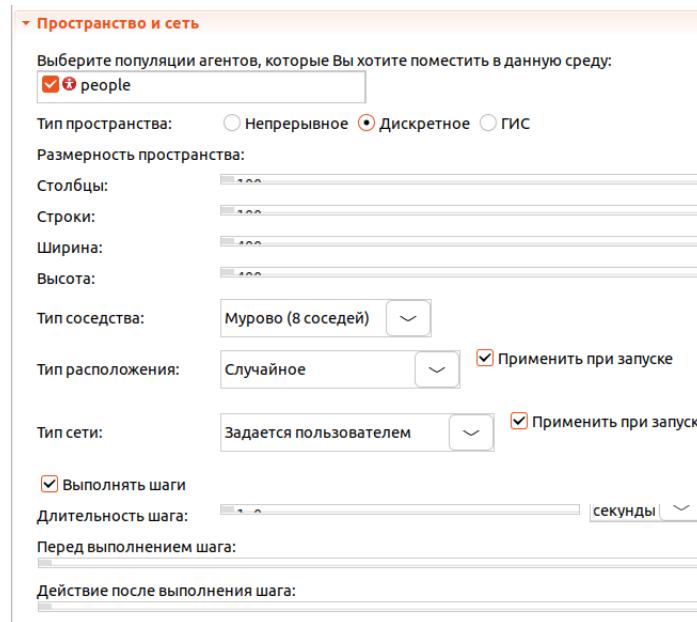


Рис. 19: Настройка популяции агентов

Цель данной модели заключается в том, чтобы промоделировать поведение людей. Существует несколько правил, по которым действуют люди в данной модели:

1. агент, который имеет только 1 агента соседа, переедет, если это сосед отличного цвета;
2. агент, имеющих 2 соседей, не будет переезжать, если хотя бы один из них имеет тот же цвет, что и он сам;
3. агент, проживающий по соседству от 3 до 5 человек, не будет переезжать, если хотя бы два из них будут цвета, что и он сам;
4. агент с соседями от 6 до 8 человек не будет переезжать, если хотя бы 3 из них будут одного цвета.

Также стоит сказать, что изначально агентам задаются случайные цвета с равной вероятностью. Таким образом, агент будет иметь два параметра: цвет и состояние счастья, которое принимает тип boolean. (Рисунок 20)

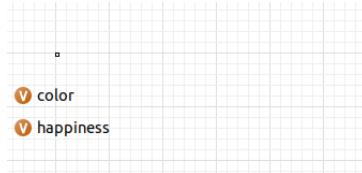


Рис. 20: Переменные и параметры агентов

Также в данной модификации модели рассматривается вариант при котором, агенты могут положительно взаимодействовать не только с соседями своего цвета, но и соседями других цветов. (Рисунок 21)

People - Тип агента

```
Перед выполнением шага:
int near = 0;
Agent[] neighbors = getNeighbors();

if (neighbors == null) {
    happiness = true;
    return;
}

for (Agent agent : neighbors) {
    if (((People)agent).color == color) {
        ++near;
    }
}

// Зелёные дружат с синими
if (get_Main().radio.getValue() == 1) {
    if ((color == Color.green && ((People)agent).color == Color.blue)
        || (color == Color.blue && ((People)agent).color == Color.green)) {
        ++near;
    }
}

// Красные дружат с зелёными
if (get_Main().radio.getValue() == 2) {
    if ((color == Color.red && ((People)agent).color == Color.blue)
        || (color == Color.blue && ((People)agent).color == Color.red)) {
        ++near;
    }
}

// Красные дружат с зелёными
if (get_Main().radio.getValue() == 3) {
    if ((color == Color.red && ((People)agent).color == Color.green)
        || (color == Color.green && ((People)agent).color == Color.red)) {
        ++near;
    }
}

if (color == Color.red) {
    happiness = (near >= get_Main().intoleranceLevelRed * neighbors.length);
} else if (color == Color.green) {
    happiness = (near >= get_Main().intoleranceLevelGreen * neighbors.length);
} else if (color == Color.blue) {
    happiness = (near >= get_Main().intoleranceLevelBlue * neighbors.length);
}
```

Рис. 21: Реализация алгоритма взаимодействия агентов

Для сбора статистики кто среди агентов счастлив, а кто нет, была создана круговая диаграмма, которая отражает данные показатели. (Рисунок 22)

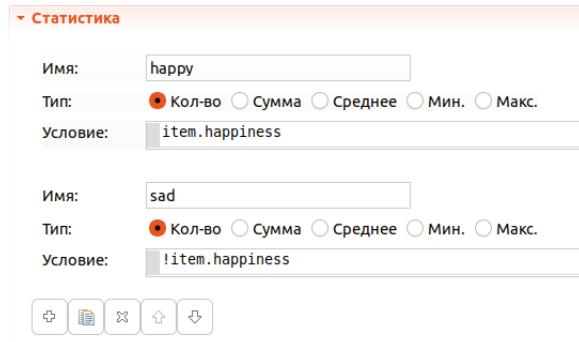


Рис. 22: Реализация алгоритма взаимодействия агентов

Также можно промоделировать различные сценарии взаимодействия агентов. (Рисунок 23)

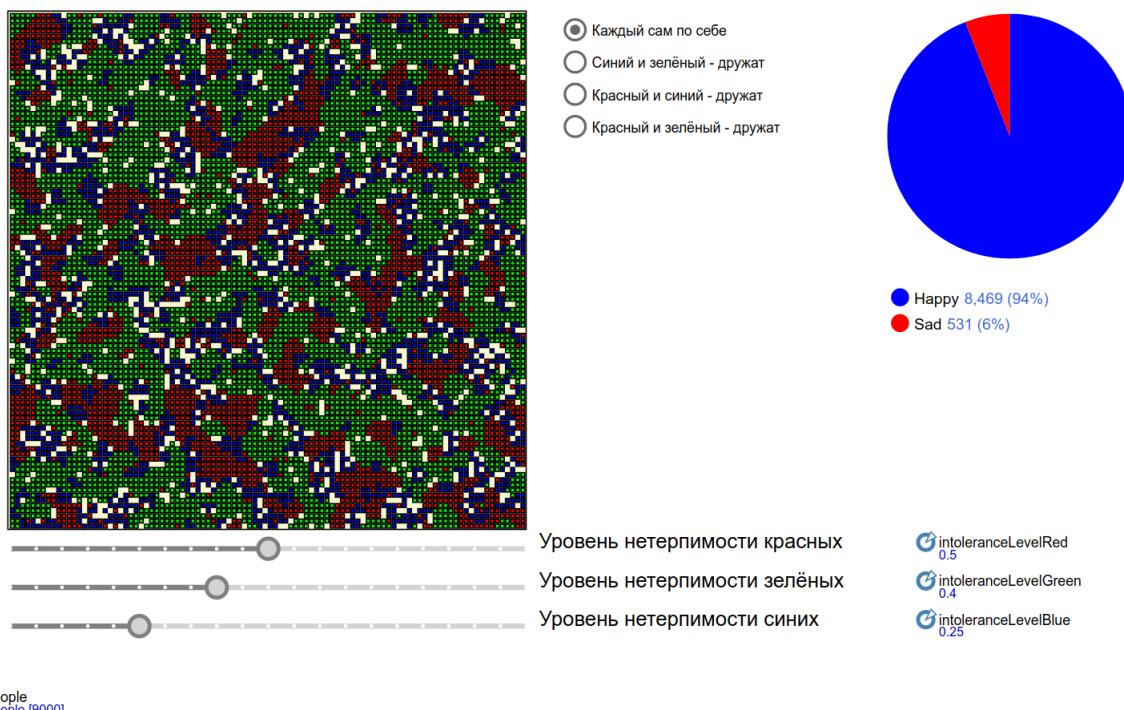


Рис. 23: Результат работы модели

Таким образом, была реализована модель сегрегации Шеллинга при различных условиях взаимодействия агентов.

Модель сегрегации Шеллинга в Python

Задание:

Реализовать модель сегрегации Шеллинга в Python.

Решение:

Суть данного алгоритма была описана в предыдущем разделе. Здесь же будет рассмотрена просто реализация предыдущей задачи на языке программирования Python.

Для начала был создан класс *Person*, который содержит информацию о члене популяции. У него имеется два поля: тип и счастлив ли данный агент или нет. (Рисунок 24)

```
class Person(object):
    def __init__(self, *args, types_count: int, **kwargs) -> None:
        self.type = random.choice([x for x in range(1, types_count + 1)])
        self.happiness = True

    def __repr__(self) -> None:
        return f"Person(type={self.type}, self.happiness={self.happiness})"
```

Рис. 24: Класс члена популяции *Person*

Далее был создан класс популяции – *Population*, в котором изначально генерировалась популяция агентов и задавалась начальная конфигурация системы. (Рисунок 25)

```
class Population(object):
    def __init__(self, *args,
                 dimention: int,
                 population_size: int,
                 types_count: int,
                 tolerance_levels: List[float],
                 **kwargs) -> None:
        self.dimention = dimention
        self._population_size = population_size
        self.types_count = types_count
        self._tolerance_levels = tolerance_levels

        self._population = [Person(types_count=types_count) for _ in range(population_size)]
        self.grid = np.zeros((dimention, dimention), dtype=Person)
        self._empty_cells = []

        self.simulation_results = []

    def _initial_empty_cells(self) -> None:
        for i in range(self.dimention):
            for j in range(self.dimention):
                if self.grid[i, j] == 0:
                    self._empty_cells.append((i, j))

    def _initial_configuration(self) -> None:
        grid_positions = [(i, j) for i in range(self.dimention) for j in range(self.dimention)]
        people_positions = random.sample(grid_positions, self._population_size)
        for i in range(len(self._population)):
            self.grid[people_positions[i]] = self._population[i]
        self._initial_empty_cells()
        self.simulation_results = [deepcopy(self.grid)]
```

Рис. 25: Класс популяции *Population*

Далее были созданы функции для перемещения агента в системе, изменения уровня счастья конкретного агента и собственно основной функции симуляции. (Рисунок 26)

```

def _get_neighbors_position(self, i: int, j: int) -> None:
    function = lambda element: not (element[0] < 0 or element[0] > self.dimention - 1 or \
                                    element[1] < 0 or element[1] > self.dimention - 1)
    return list(filter(function, [(i - 1, j + 1), (i, j + 1), (i + 1, j + 1), (i - 1, j), (i + 1, j), (i - 1, j - 1), (i, j - 1), (i + 1, j - 1)]))

def _agent_happy(self, agent: Person, i: int, j: int) -> None:
    near = 0
    neighbors_positions = self._get_neighbors_position(i, j)
    neighbors = [self.grid[neighbor] for neighbor in neighbors_positions]
    for neighbor in neighbors:
        if neighbor == 0 or neighbor.type == agent.type:
            near += 1
    agent.happiness = near >= self._tolerance_levels[agent.type - 1] * len(neighbors)

def _jump_to_random_empty_cell(self, i: int, j: int) -> None:
    agent = self.grid[i, j]
    new_position = random.choice(self._empty_cells)
    self._empty_cells.remove(new_position)
    self.grid[new_position] = Person(types_count=self.types_count)
    self.grid[new_position].type = agent.type
    self.grid[new_position].happiness = agent.happiness
    self.grid[i, j] = 0
    self._empty_cells.append((i, j))

def simulation(self, iterations_count: int) -> None:
    self._initial_configuration()
    for _ in range(iterations_count):
        for i in range(self.dimention):
            for j in range(self.dimention):
                if self.grid[i, j] != 0:
                    agent = self.grid[i, j]
                    self._agent_happy(agent, i, j)
                    if not agent.happiness and random.random() > 0.7:
                        self._jump_to_random_empty_cell(i, j)
    self.simulation_results.append(deepcopy(self.grid))

```

Рис. 26: Функции для перемещения агентов в системе и симуляции

Также были созданы вспомогательные функции для отрисовки текущего состояния системы и для анимации получившихся результатов. (Рисунок 27)

```

def plot_population(population: Population, *, colors: List[str]) -> None:
    paint_grid = [[population.grid[i, j].type if population.grid[i, j] != 0 else 0
                  for j in range(population.dimention)]
                  for i in range(population.dimention)]
    cmap = col.ListedColormap(colors)
    plt.figure(figsize=(10,10))
    plt.pcolormesh(paint_grid, edgecolor="black", cmap=cmap)
    plt.show()

```

Рис. 27: Функции для отрисовки текущего состояния системы

```

def draw_animation(population: Population, *, colors: List[str], step=1) -> HTML:
    fig, ax = plt.subplots(figsize=(10, 10))

    cmap = col.ListedColormap(colors)
    paint_grids = []

    for k in range(kIterations):
        paint_grids.append([[population.simulation_results[k][i, j].type if population.simulation_results[k][i, j] != 0 else 0
                            for j in range(population.dimention)]
                            for i in range(population.dimention)])

    def animate(i):
        return ax.pcolormesh(paint_grids[i], edgecolor="black", cmap=cmap)

    anim = animation.FuncAnimation(fig, animate, frames=range(0, kIterations, step), blit=False)
    plt.close()

    return HTML(anim.to_jshtml())

```

Рис. 28: Функции для анимации процесса симуляции

Ещё были реализованы функции для сбора статистики по популяции, а именно отслеживание количества счастливых агентов. (Рисунок 29)

```

def _stats(population: Population) -> Tuple[int, int, int]:
    happy, sad = 0, 0

    for i in range(population.dimention):
        for j in range(population.dimention):
            if population.grid[i, j] != 0:
                if population.grid[i, j].happiness:
                    happy += 1
                else:
                    sad += 1

    return (happy, sad, happy + sad)

def pie_population(population: Population) -> None:
    happy, sad, _ = _stats(population)
    plt.figure(figsize=(8, 8))
    plt.pie([happy, sad], autopct="%1.1f%%", labels=["Happy", "Sad"])
    plt.title("Соотношение счастья")
    plt.legend()
    plt.show()

```

Рис. 29: Функции сбора статистики по популяции

Визуализация различных уровней счастья агентов в модели, построенной при помощи Python, сходна с той, что была получена при использовании среды AnyLogic. (Рисунок 30)

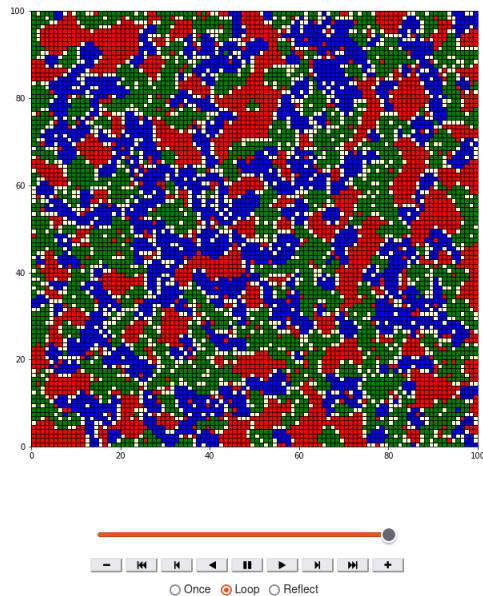


Рис. 30: Визуализация полученных результатов

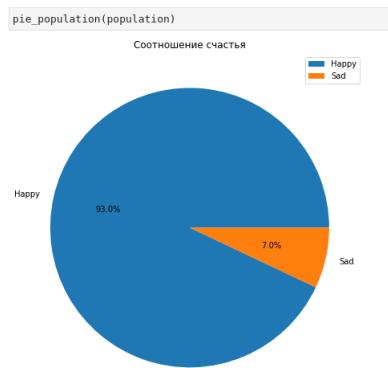


Рис. 31: Статистика по полученным результатам

Также были промоделированы различные другие ситуации, допустим когда рассматривается всего две типа людей или когда изменяется уровень толерантности группы.

Таким образом, была реализована модель сегрегации Шеллинга при различных условиях взаимодействия агентов.

Диаграммы состояний и события

Светофор

Задание:

Реализовать модели работы светофоров при помощи диаграммы состояний.

Решение:

Необходимо промоделировать работу двух светофоров – дорожного и пешеходного. Сначала стоит нарисовать данные объекты. (Рисунок 32)

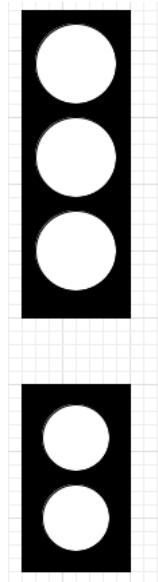


Рис. 32: Визуальное отображение светофоров

Далее была создана диаграмма действий, в которой реализована основная логика светофора. (Рисунок 33)

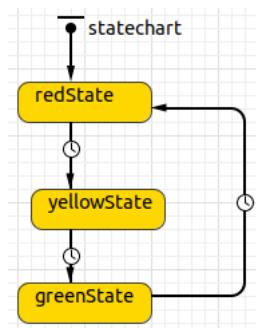


Рис. 33: Диаграмма состояний светофоров

Можно сказать что данная логика реализована относительно дорожного светофора.

Все переходы модели осуществляются по таймауту. Рассмотрим логику работы светофоров, когда на дорожном светофоре горит красный свет. Мы назначаем дорожному светофору красный свет, а пешеходному назначаем зелёный. (Рисунок 34)

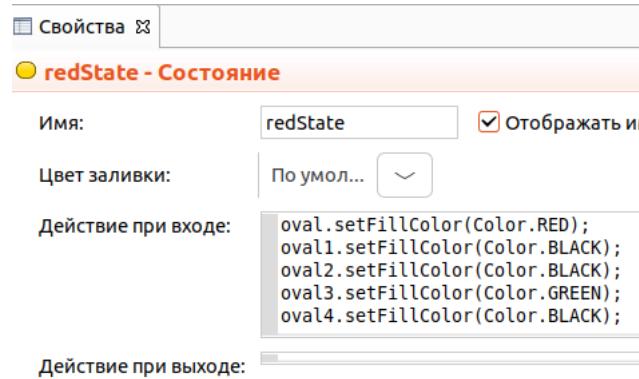


Рис. 34: Пример логики одного из состояний

Аналогично было проделано для жёлтого и зелёного состояний.

Таким образом, при помощи диаграммы состояний была промоделирована работа изменения сигналов дорожного и пешеходного светофоров.

SIR – агентная модель

Задание:

Реализовать и проанализировать модель распространения инфекционного заболевания посредством агентного моделирования.

Решение:

Изначально необходимо создать популяцию людей в размере – 1000 человек, в непрерывном пространстве, имеющую упорядоченный тип расположения и случайный тип сети с двумя связями. (Рисунок 35)

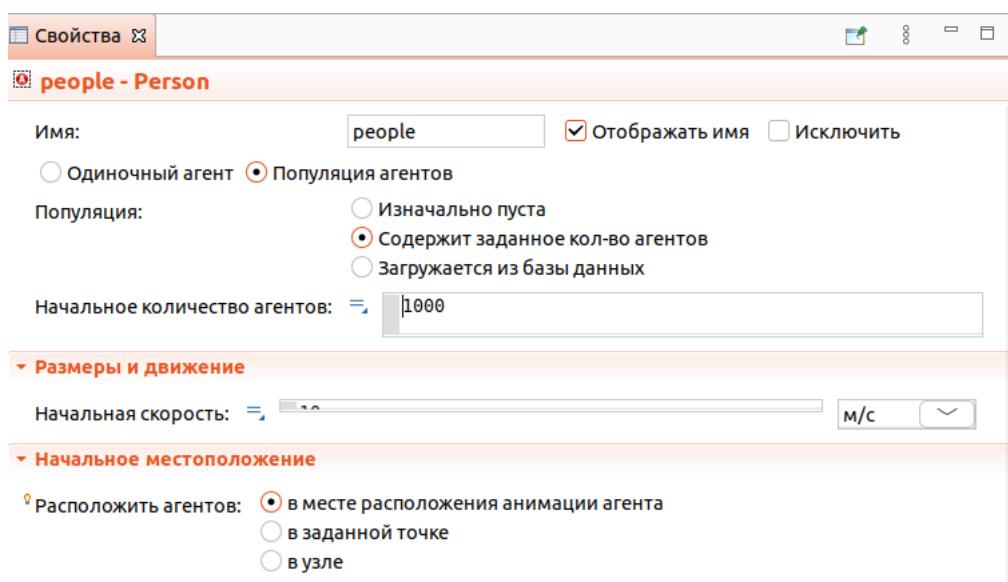


Рис. 35: Настройка агентов модели

Далее для всех агентов указываются количество дней, требуемое на восстановление, количество контактов и вероятность заразиться. После задания необходимых параметров модели было решено перейти описанию диаграммы состояний для конкретного агента в популяции. (Рисунок 36)

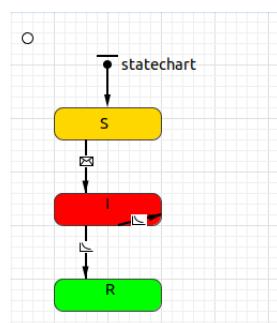


Рис. 36: Описание диаграммы состояний конкретного агента в популяции

В приведённой выше диаграмме человек может находиться в состояниях:

1. когда человек здоров и может заразиться;
2. когда человек заразился и болеет;
3. когда человек выздоровел и уже не может заразиться.

Переход из первого состояния во второе происходит на основе получения сообщения, отправка сообщения заражённым агентом осуществляется с интенсивностью, равной вероятности заразиться умноженной на количество контактов. Переход из состояния, когда человек болеет в состояние выздоровления происходит согласно интенсивности, равной единице делённой на количество дней, требуемых на восстановления после заболевания данной болезнью.

Для начала распространения инфекции необходимо инфицировать нескольких агентов в начале симуляции. (Рисунок 37)

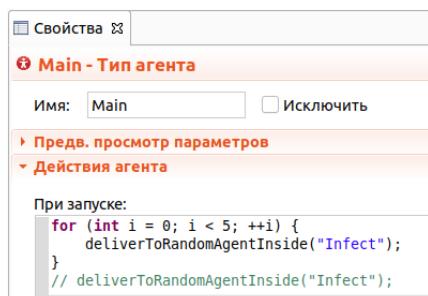


Рис. 37: Инфицирование нескольких агентов на начальной стадии

Также был проведёт сбор статистики по числу агентов, которые находятся в рассмотренных выше состояниях для того, чтобы на графиках проанализировать текущую ситуацию модели и как данная модель отличается от рассмотренной ранее. (Рисунок 38)

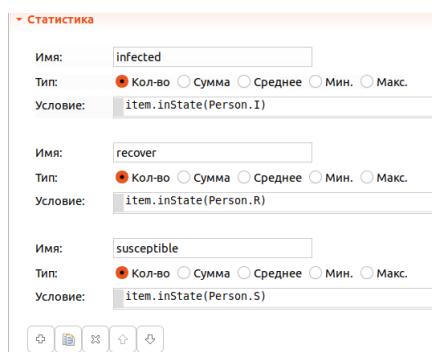


Рис. 38: Сбор статистики по каждому агенту

Если запустить данную модель, то получим схожую картину, которая была когда мы рассматривали системную-динамику. (Рисунок 39)

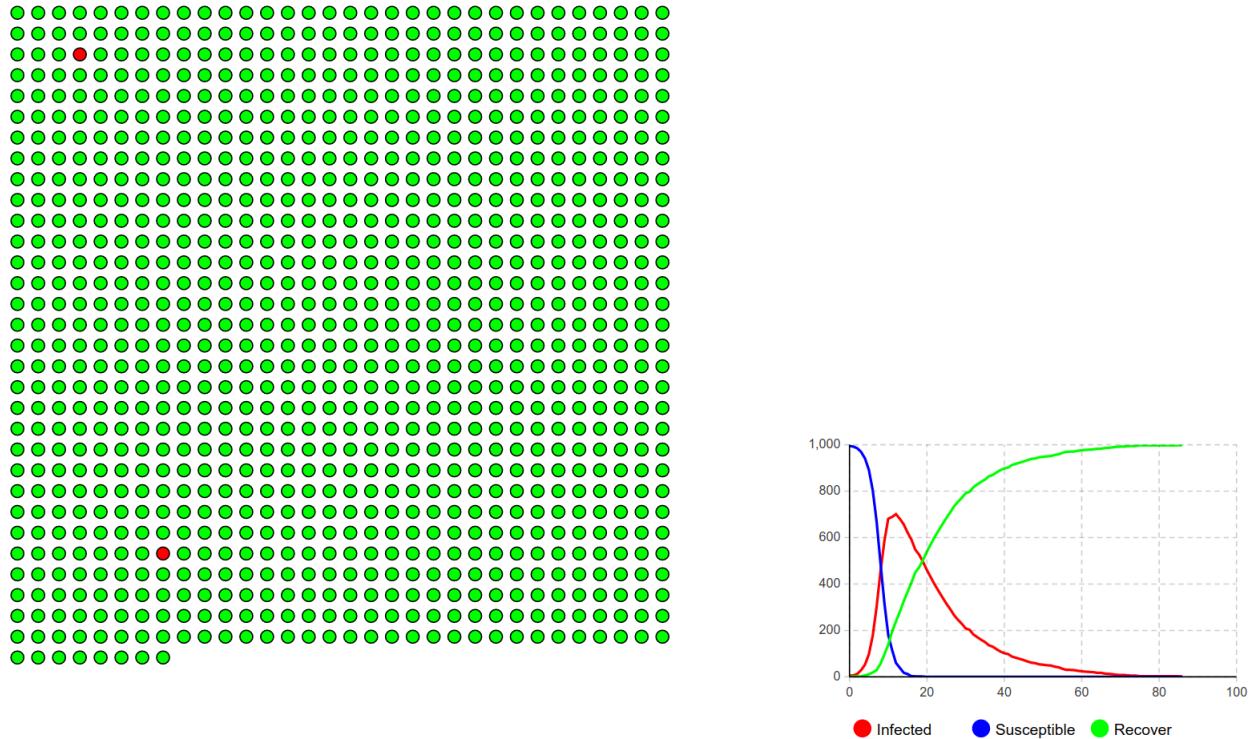


Рис. 39: Процесс моделирования

Таким образом, посредством агентного моделирования была реализована модель распространения инфекций.

Ноутбук + зарядка

Задание:

Реализовать модель работы ноутбука при условии, что присутствует возможность зарядки.

Решение:

Для начала стоит с помощью элементов презентации нарисовать мышку и сам ноутбук. (Рисунок 40)

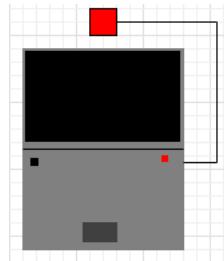


Рис. 40: Визуализация ноутбука и мышки

Далее была описана логика работы и построены диаграммы состояний. (Рисунок 41)

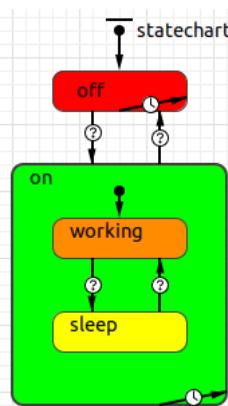


Рис. 41: Диаграммы состояний

Всего существуют два глобальных состояний: ноутбук включен и ноутбук выключен. Пока ноутбук включен, то у него постепенно расходуется заряд, если заряд ноутбука заканчивается, то он переходит в состояние выключения. Для отслеживания текущего состояния переменных у двух состояний есть внутренние переходы по таймауту, которые обновляются каждую секунду. Как только ноутбук подключается к зарядке, то им снова можно пользоваться.

Также в глобальном состоянии, когда ноутбук включен есть два внутренних: работает и в состоянии сна. Если не нажимать на кнопки на клавиатуре или на тачпад, то ноутбук перейдет в состояния сна и продолжит потреблять зарядку.

Ещё у ноутбука есть кнопка включения и выключения, чтобы пользователь мог сам переключаться между состояниями.

Реализация логики одного из внутренних переходов ноутбука представлена ниже. (Рисунок 42)

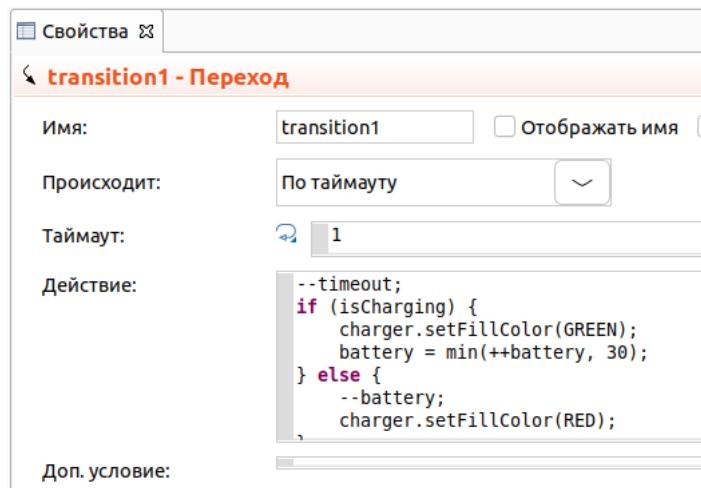


Рис. 42: Реализация логики внутреннего перехода в ноутбуке

Таким образом, была реализована модель, имитирующая работу ноутбука с зарядкой.

SIERD + вакцинация

Задание:

Реализовать модель распространения заболевания с добавлением новых состояний: носитель заболевания, умершие и вакцинированные.

Решение:

Данная модель реализуется похожую логику на ту которая была в модели SIR, поэтому в данном разделе будут рассмотрены только модификации.

В качестве новых параметров добавились вероятность вакцинации, вероятность выжить и время, которое должно пройти с момента вакцинации, чтобы вакцина успела действовать.

Диаграмма состояний в таком случае будет изменена и примет следующий вид. (Рисунок 43)

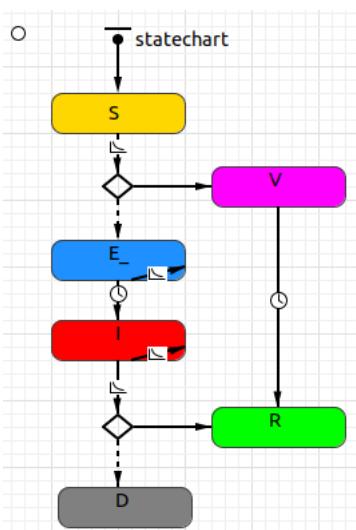


Рис. 43: Диаграмма состояний

Теперь из состояния когда человек полностью здоров можно перейти в состояние, когда человек вакцинирован или является носителем. Данный переход происходит с вероятностью, для которой был введён отдельный параметр. Если агент вакцинировался, то он сразу же попадает в состояние – выздоровевший.

Если же человек стал носителем, то дальше через некоторое время он заболевает, и в процессе пока он является носителем он также может заражать других здоровых людей. Далее из состояния больного агент может перейти в состояние умершего или выздоровевшего с заданной вероятностью. Для

этого был введён специальный параметр, который рассмотрен чуть выше.

В результате работы данной модели был получен график, который похож, на тот, что мы рассматривали на системной динамике. (Рисунок 44)

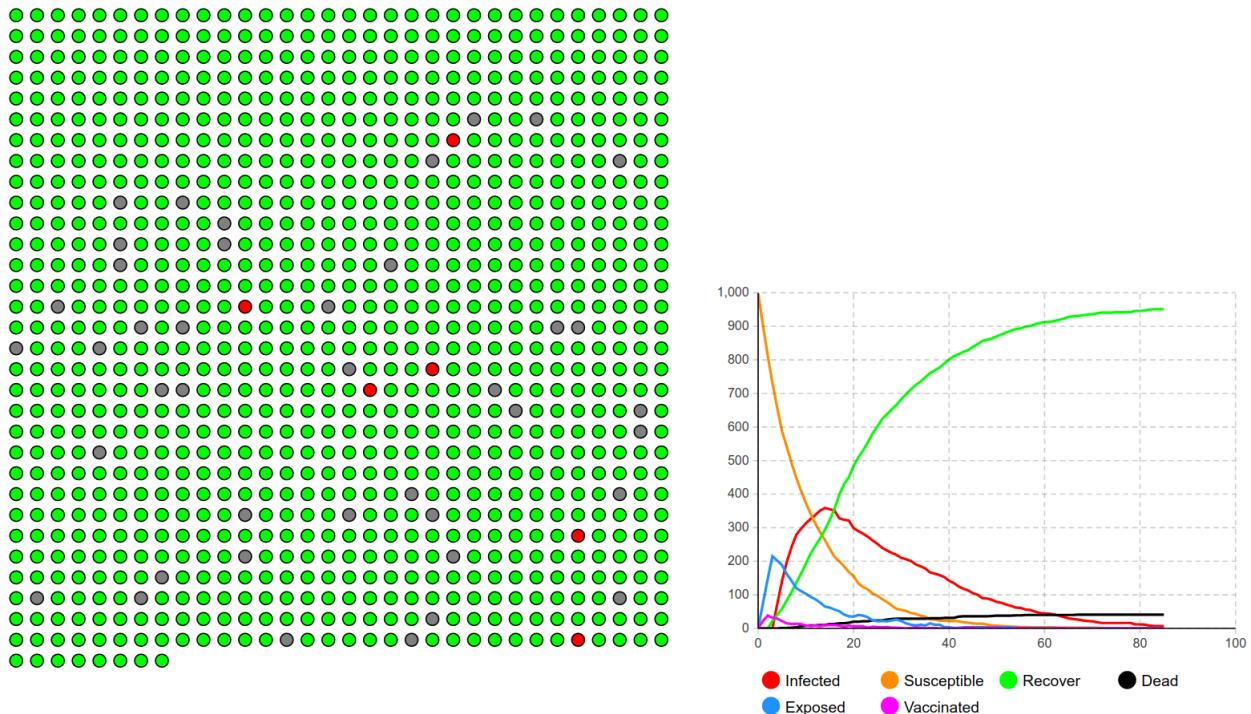


Рис. 44: Результат модели SIERD

Таким образом, посредством агентного моделирования была построена модель распространения заболевания с учётом вакцинации.

Диаграмма действий

Сумма ряда – экспонента

Задание:

Используя диаграмму действий, реализовать алгоритм вычисления суммы ряда экспоненты.

Решение:

Для вычисления экспоненты воспользуемся суммой ряда Тейлора:

$$e^x \approx \sum_{k=0}^N \frac{x^k}{k!}$$

В соответствии с данной формулой была построена рекуррентная формула для вычисления данной суммы. (Рисунок 45)

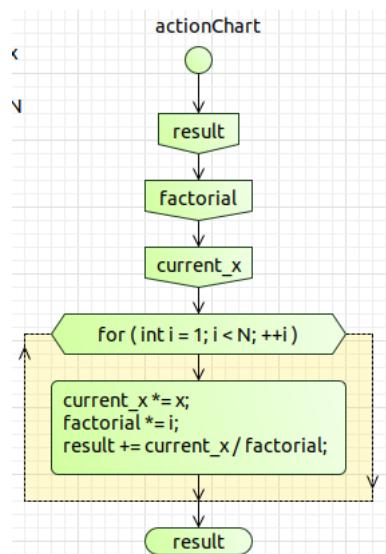


Рис. 45: Диаграмма действий для суммы ряда экспоненты

На вход данному алгоритму поступает количество итераций и степень x . (Рисунок 46)

| | |
|----|-------------------|
| 1 | \textcircled{X} |
| 10 | \textcircled{Y} |

Power: 1
Precision: 10
Result: 2.7182815255731922

Рис. 46: Результаты вычисления ряда экспоненты

Результаты расчёта совпали с результатами Wolfram Mathematica.

Таким образом, на примере создания функции для нахождения суммы ряда экспоненты был освоен процесс работы с диаграммами действий.

Сумма ряда – синус

Задание:

Используя диаграмму действий, реализовать алгоритм вычисления суммы ряда синуса.

Решение:

Для вычисления экспоненты воспользуемся суммой ряда Тейлора:

$$\sin x \approx \sum_{k=0}^N (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

В соответствии с данной формулой была построена рекуррентная формула для вычисления данной суммы. (Рисунок 47)

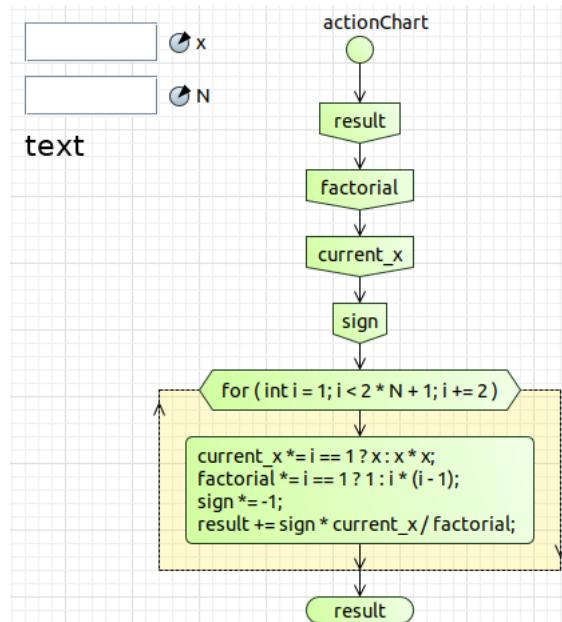


Рис. 47: Диаграмма действий для суммы ряда синуса

На вход данному алгоритму поступает количество итераций и степень x . (Рисунок 48)

3.14 $\textcircled{X}_{3.14}$
200 \textcircled{N}_{200}
Power: 3.14
Precision: 200
Result: 0.0015926529164867666

Рис. 48: Результаты вычисления ряда синуса

Результаты расчёта совпали с результатами Wolfram Mathematica.

Таким образом, на примере создания функции для нахождения суммы ряда синуса был освоен процесс работы с диаграммами действий.

Агентные модели

Подготовка к зачёту

Задание:

Реализовать модель подготовки к зачёту для одного студента.

Для подготовки к зачёту дано N вопросов.

Студент использует следующую схему подготовки.

1. Сначала он прорабатывает теоретический материал по очередному вопросу (и старается запомнить). Среднее время для этого по каждому вопросу одной темы колеблется в диапазоне от 30 до 40 минут (распределение равномерное).
2. Затем студент по памяти воспроизводит материал изученного вопроса.
3. Если материал не усвоен, то требуется повторение.
4. После однократной повторной проработки вопроса студент переходит к следующему вопросу.

Процесс продолжается до тех пор, пока не будут подготовлены все вопросы.

Степень усвоения изученного материала задается случайно (треугольное распределение от 5 до 100, мода - 60).

Решение:

Сначала необходимо создать популяцию вопросов. (Рисунок 49)

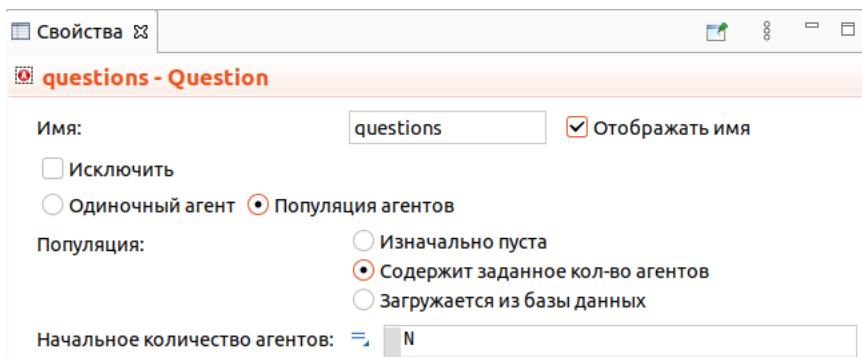


Рис. 49: Настройка популяции

Каждый из вопросов имеет несколько состояний. (Рисунок 50)

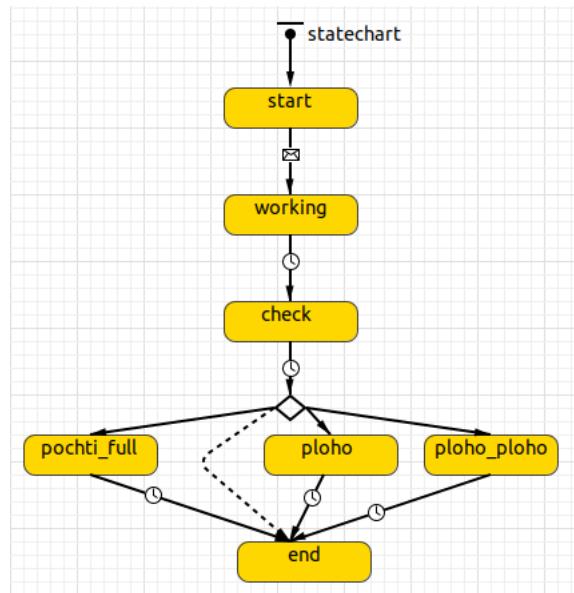


Рис. 50: Диаграмма состояний одного агента популяции

Переход между состояниями осуществляется в соответствии с правилами описанными в условии. Так как студент начинает подготовку вопроса при получении сообщения, то необходимо перед началом симуляции отправить сообщение о начале подготовки к первому вопросу. (Рисунок 51)

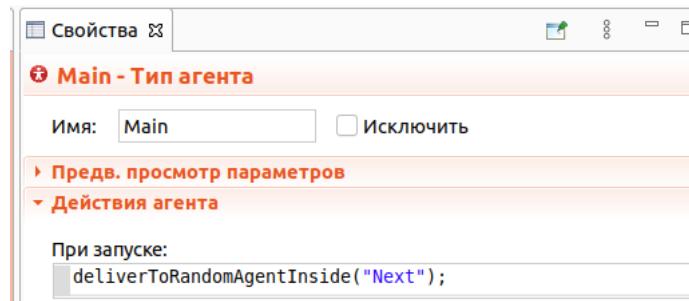


Рис. 51: Отправка сообщения первому агенту из популяции вопросов

Если запустить симуляцию, то будет следующий результат. (Рисунок 52)

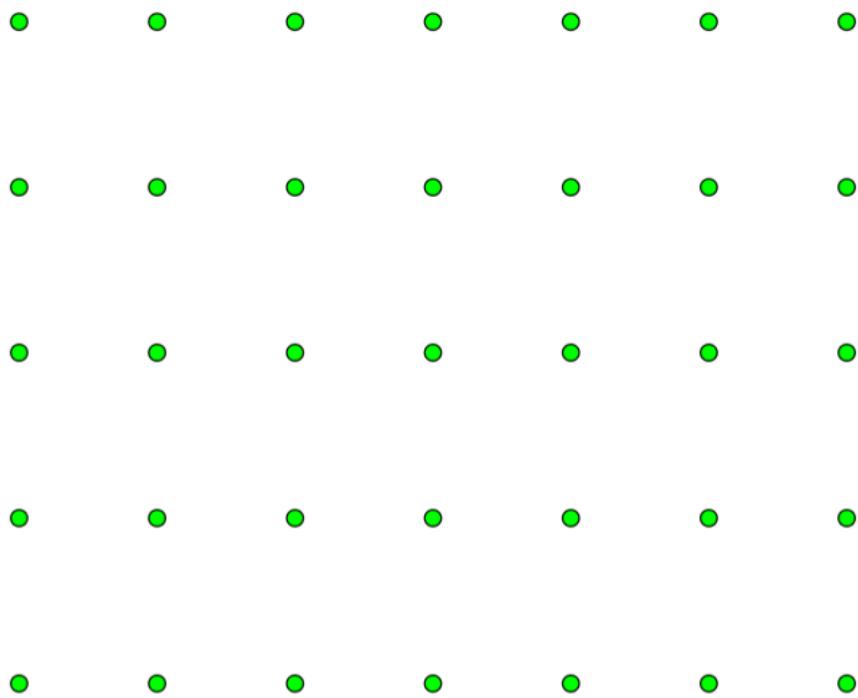


Рис. 52: Результат работы модели

Таким образом, была получена модель подготовки студента к зачёту.

Связи агентов – добавление, удаление, список, количество

Задание:

1. Создать популяцию агентов. Отобразить агенты и их связи. Для визуализации агентов использовать круг и текст с номером агента в популяции.
2. Предусмотреть:
 - добавление в популяцию нового агента отображение обновленной популяции;
 - добавление/удаление связи между случайными агентами;
 - добавление/удаление связи между агентами с заданными номерами. Номера задавать с помощью выпадающих списков.
3. Выделить цветом агента с максимальным количеством связей, вывести количество связей.
4. Выделить цветом агентов, связанных с заданным.
5. Предусмотреть удаление сделанных цветом отметок.

Решение:

Для начала необходимо создать популяцию агентов. У агентов будет их порядковый номер, текстовая метка которая отражает данную информацию и текстовая метка, которая показывает сколько других агентов связано с текущим. (Рисунок 53)

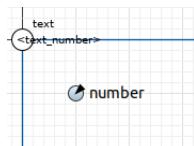


Рис. 53: Параметры популяции

Добавление новых агентов осуществляется по кнопке, также в данной кнопке осуществляется добавление нового агента в ComboBox. (Рисунок 54)

```
▼ Действие
addHumans(humans.size() + 1);
applyLayout();

String[] items = new String[humans.size()];
for (int i = 1; i <= humans.size(); ++i) {
    items[i - 1] = String.valueOf(i);
}
comboBox.setItems(items);
comboBox1.setItems(items);
```

Рис. 54: Реализация логики добавления нового агента

В самом начале при запуске модели осуществляется добавление всех сгенерированных агентов в ComboBox. (Рисунок 55)

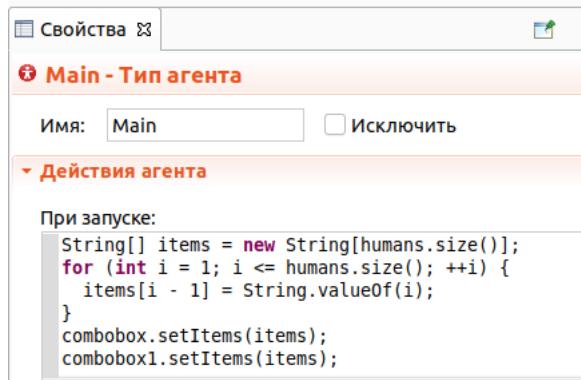


Рис. 55: Задание начальных значений выпадающих списков

Добавление связи между случайными агентами также осуществляется через кнопку. В данном алгоритме делается 100 раз попытка найти подходящую случайную вершину, то есть такую, которая не является текущей вершиной или вершиной-соседом с которым уже есть связь. Если за 100 итераций, такая вершина нашлась, то с ней возникает связь. (Рисунок 56)

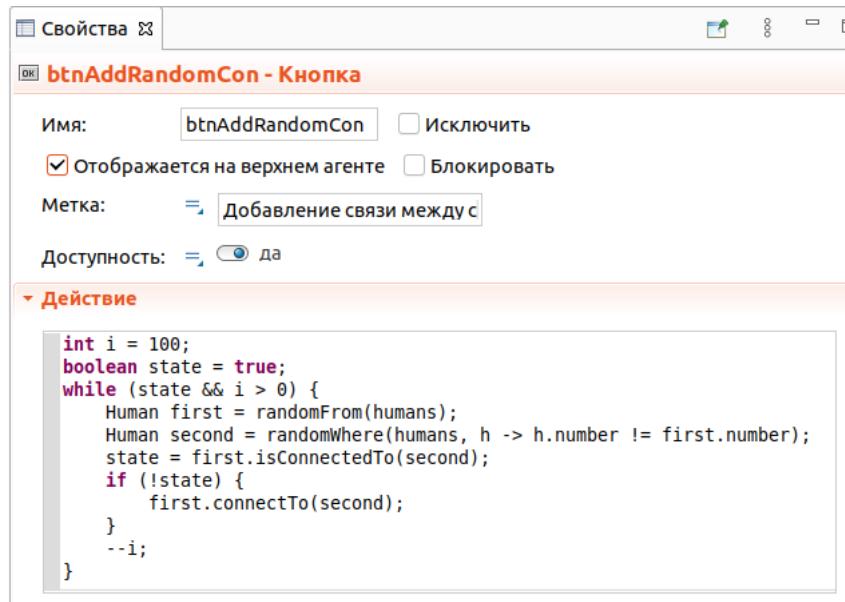


Рис. 56: Добавление связи между случайными агентами

Удаление связи между случайными агентами также реализовано по кнопке. В данном алгоритме делается 100 раз попытка найти подходящую случайную вершину, то есть такую, которая не является текущей вершиной или вершиной-соседом с которым нет связи. Если за 100 итераций, такая вершина нашлась, то связь с ней удаляется. (Рисунок 57)

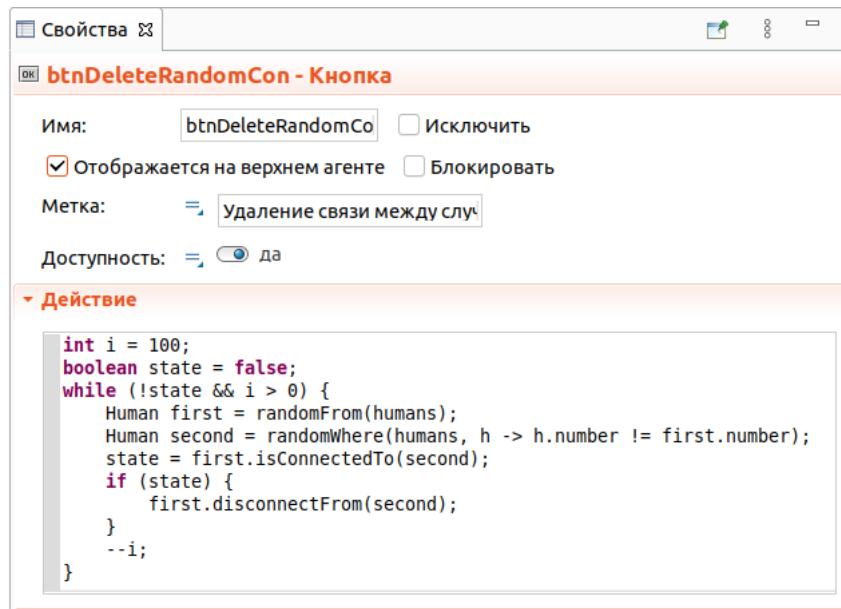


Рис. 57: Удаление связи между случайными агентами

Была реализована кнопка, по которой добавляется связь между двумя заданными вершинами. Вершины задаются через ComboBox. Если связь вершин является недопустимой, то ничего не происходит. (Рисунок 58)

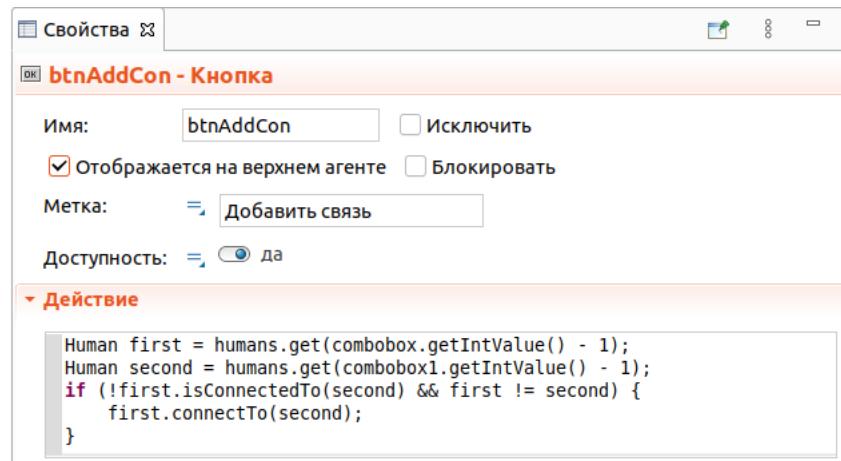


Рис. 58: Добавление связи между заданными вершинами

По аналогии с добавлением связи между двумя заданными вершинами, была реализована кнопка удаление связи между двумя заданными вершинами. (Рисунок 59)

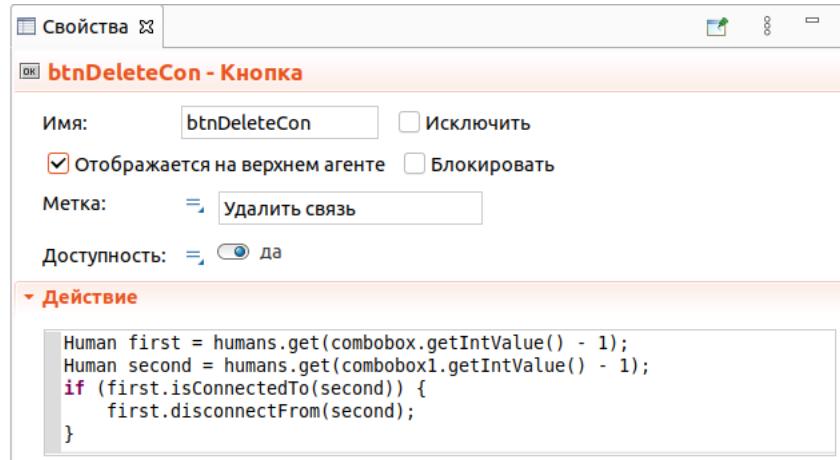


Рис. 59: Удаление связи между заданными вершинами

Далее в событии реализовано обновление агента с максимальным числом связей, он красился в красный, а всего его соседи красились в зелёный, остальные агенты в белый. (Рисунок 60)

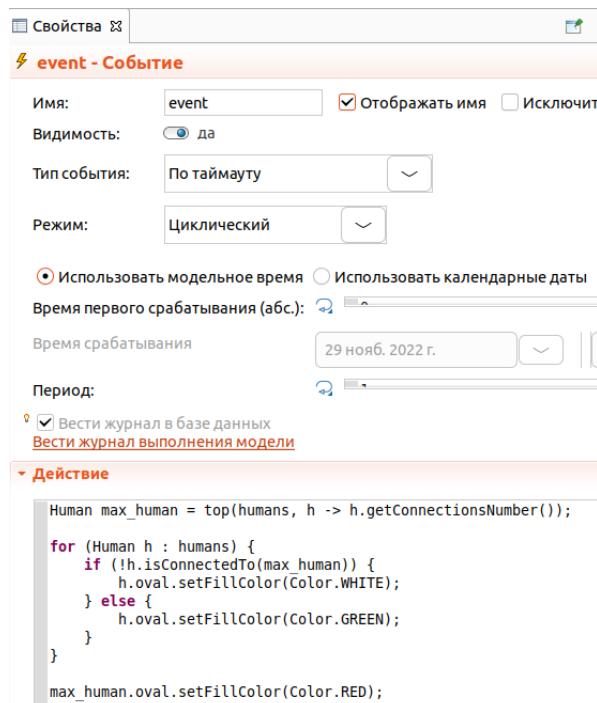


Рис. 60: Реализация события

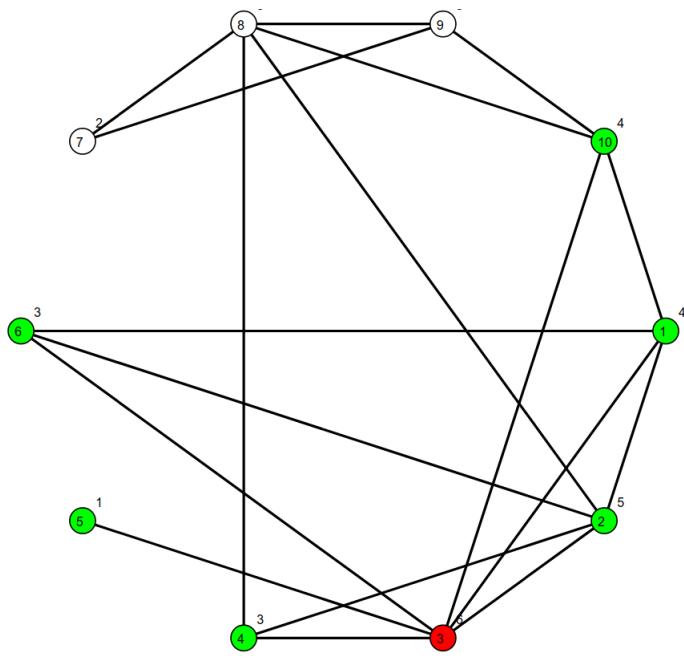


Рис. 61: Визуализация работы модели

Таким образом, в ходе реализации модели были разобраны основные инструменты для работы со связями агентов.

Работа с ГИС-картами

Доставка с покупками

Задание:

Реализовать модель цепочки поставок ретейлерам, учитывающую то, у ретейлеров имеется покупатели, приобретающие некое количество товаров с использованием ГИС-карт.

Решение:

Для начала необходимо разместить ГИС-карту и выделить регион – Франция, так как местоположения ретейлеров и дистрибуторов находятся именно в этом регионе. (Рисунок 62)

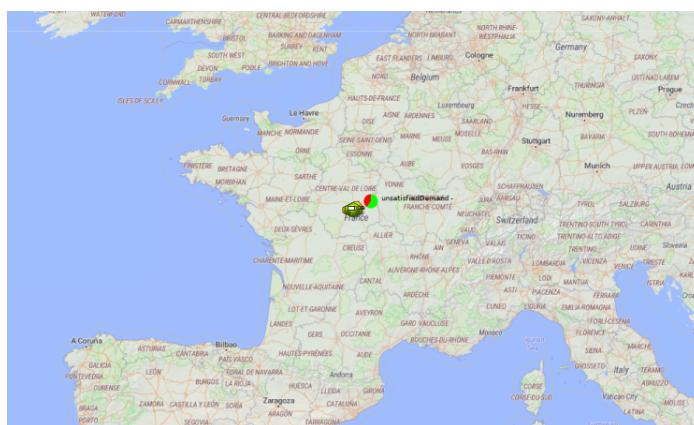


Рис. 62: Разметка ГИС-карты

Также имеются данные о расположении дистрибутеров и количестве грузовиков, имеющихся в наличии у каждого из дистрибутеров и данные о местоположениях ретейлеров находятся в таблицах, которые импортируются и используются в дальнейшем.

Далее создаются несколько популяций агентов: ретейлеров, дистрибутеров, грузовиков и заказов. (Рисунок 63)

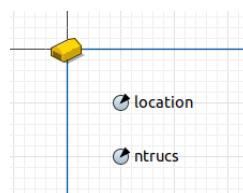


Рис. 63: Популяция дистрибутеров

У дистрибутеров имеется два параметра: позиция и количество грузовиков, которые у них имеются.

У агентов типа «заказ» имеется также два параметра: количество товара заказа и ретейлер, который оформил данный заказ. (Рисунок 64)

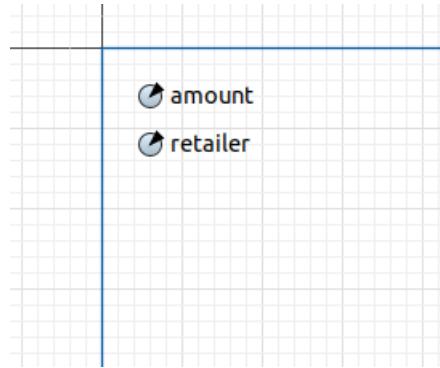


Рис. 64: Популяция заказов

Грузовики имеют несколько состояний: находятся у дистрибутера, двигаются к ретейлеру, разгружаются у ретейлера, двигаются к дистрибутеру. (Рисунок 65)

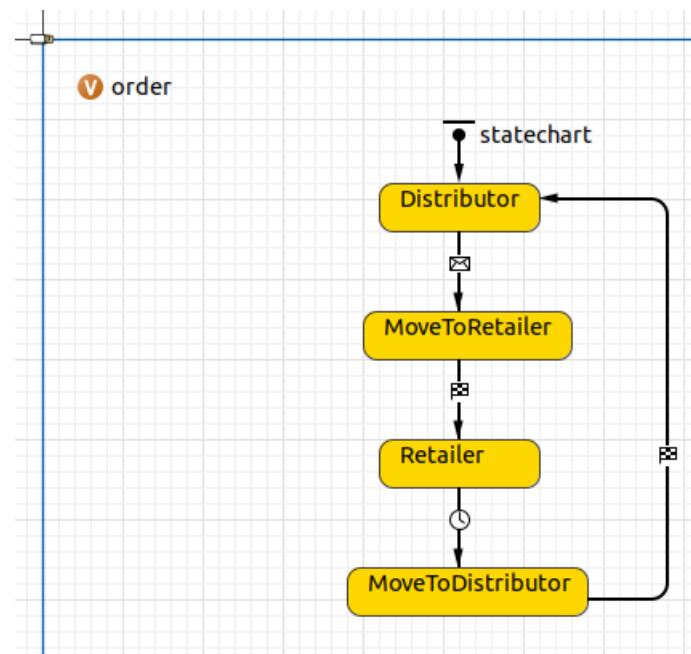


Рис. 65: Популяция грузовиков

Грузовик начинает свое движение при получении сообщения. Отправка данного сообщения осуществляется ретейлером, когда достигнут критический запас при помощи событий. (Рисунок 66)

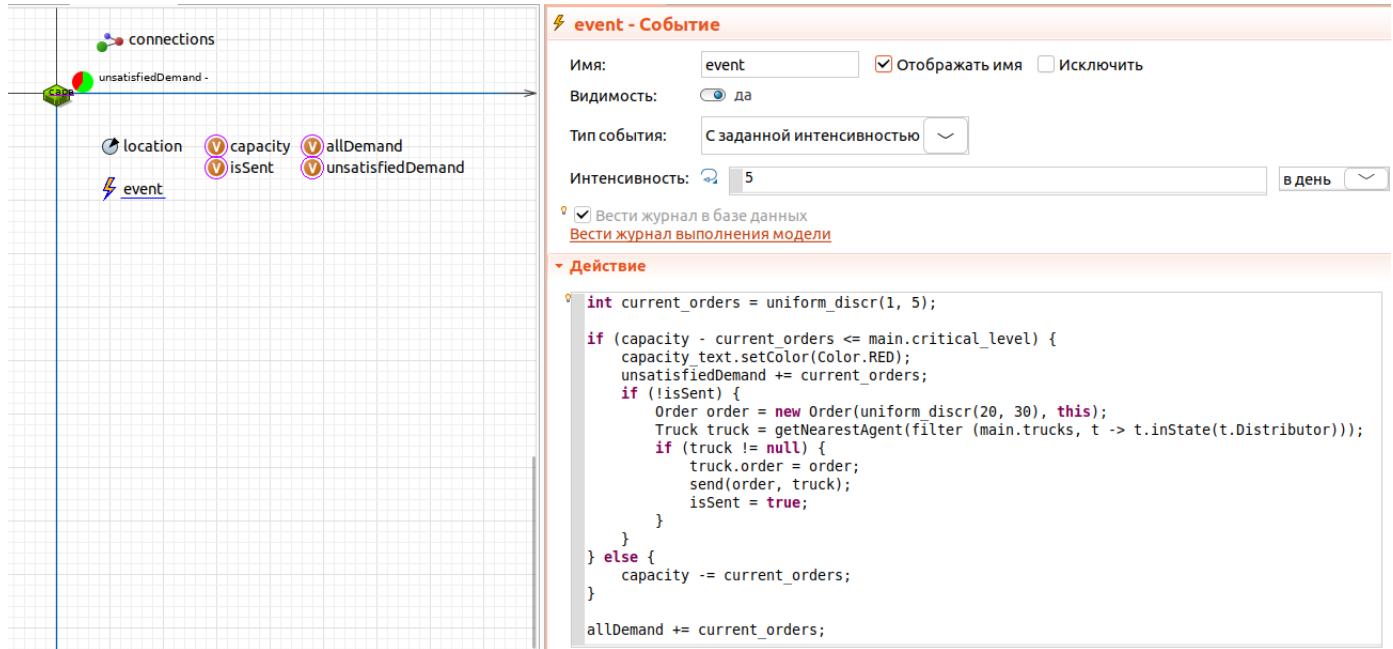


Рис. 66: Популяция заказов и логика основного события

Также у ретейлеров имеется круговая диаграмма, которая отражает количество удовлетворённого и неудовлетворённого спроса, а также имеется метка, которая отражает актуальное число запасов данного ретейлера. (Рисунок 67)

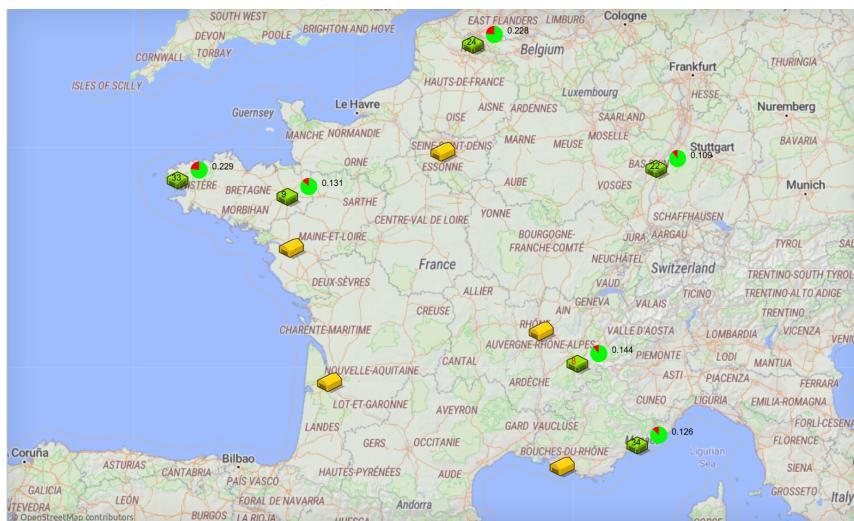


Рис. 67: Запуск симуляции модели

Таким образом, была реализована модель доставки с покупками, на при-

мере которой мы ознакомились с основными механиками работы с ГИС-картами.

Проект «Доставка товаров на примере маркетплейса Ozon»