**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ**

**УНИВЕРСИТЕТ»**

**(СПбГЭУ)**

Факультет информатики и прикладной математики

Кафедра прикладной математики и экономико-математических методов

**КУРСОВАЯ РАБОТА**

по дисциплине:

«**Языки и методы программирования**»

Тема «Реализация агентно-ориентированных моделей на языке Python»

Направление 01.03.02 Прикладная математика и информатика

Направленность Прикладная математика и информатика в экономике и управлении

Обучающийся Захаров Кирилл Александрович

Группа ПМ1801 Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил Гниденко И. Г.

Должность к.э.н., доцент

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2019

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc28086655)

[1. МОДЕЛЬ ШЕЛЛИНГА 5](#_Toc28086656)

[2. РЕАЛИЗАЦИЯ АГЕНТНО-ОРИЕНТИРОВАННЫХ МОДЕЛЕЙ 6](#_Toc28086657)

[2.1. Дискретная стохастическая модель Шеллинга 6](#_Toc28086658)

[2.2. Непрерывная стохастическая модель Шеллинга 10](#_Toc28086659)

[2.3. Модель Шеллинга для социальных сетей 14](#_Toc28086660)

[ЗАКЛЮЧЕНИЕ 20](#_Toc28086661)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc28086662)

[ПРИЛОЖЕНИЕ А 22](#_Toc28086663)

[ПРИЛОЖЕНИЕ Б 24](#_Toc28086664)

[ПРИЛОЖЕНИЕ В 26](#_Toc28086665)

[ПРИЛОЖЕНИЕ Г 28](#_Toc28086666)

# **ВВЕДЕНИЕ**

Масштабные социальные процессы, нередко формирующиеся спонтанно под влиянием непредсказуемых факторов, как это наблюдается сейчас в Европейских странах в связи с массовым притоком мигрантов, требуют разработки новых инструментов мониторинга и прогнозирования, позволяющих не только оперировать большим количеством данных, но и отражать сложную динамику, определяемую действиями отдельных людей. Таким инструментом является агент-ориентированное моделирование, возможности которого в данный момент усилены за счет использования суперкомпьютерных вычислительных ресурсов.

В данной работе рассматривается применение агентно-ориентированного подхода при моделировании социальных процессов, описывается модель расовой сегрегации Томаса Шеллинга, разработанная в 1969 году [1].

Агентно-ориентированные модели включают в себя агентов, предназначенных для моделирования людей и других объектов, которые собирают информацию о мире, принимают решения и предпринимают действия.

Агенты обычно располагаются в пространстве или в сети и взаимодействуют друг с другом локально. У них есть несовершенная или неполная информация о мире. Также, агент-ориентированные модели включат в себя случайность как среди агентов, так и в мире в целом, иначе они называются стохастическими.

Агентно-ориентированные модели полезны для моделирования динамики систем, которые не прибывают в равновесии. В частности, они особенно полезны для понимания отношений между отдельными решениями и поведением системы.

Цель курсовой работы, реализовать модель Шеллинга, модифицировать ее, применить относительно социальных сетей и показать ее удобность для прогнозирования социальных процессов в интернете.

Задачи, которые стояли для достижения цели:

1. реализация дискретной стохастической модели Шеллинга;
2. реализация непрерывной стохастической модели Шеллинга;
3. изучение поведения людей в социальных сетях, подбор параметров;
4. модификация и применение непрерывной стохастической модели Шеллинга в социальных сетях;
5. эксперименты с моделью.

Для достижения целей, используется литература связанная с агентным моделированием и социологические исследования, позволяющие совершить подбор параметров для социальных сетей.

# **1. МОДЕЛЬ ШЕЛЛИНГА**

Модель расовой сегрегации впервые была представлена Томасом Шеллингом в 1969 году. Модель мира Шеллинга – это сетка, в которой каждая ячейка обозначает дом. Дома заняты двумя видами агентов, помеченных, например, красным или синим, в примерно равных количествах (в предложенной нами модели у нас не два типа агентов, а три). Около 10% домов при этом пусты.

В любой момент времени агент может быть счастлив или несчастен, в зависимости от других агентов по соседству, где «окрестность» каждого дома – это набор из восьми соседних ячеек, которые окружают дом. В одной версии модели агенты счастливы, если у них есть как минимум два соседа, таких же как они, и несчастны, если у них есть один сосед или ноль соседей. В предложенной нами модели пользователь может самостоятельно выбирать предел количества соседей.

Моделирование продолжается, выбирая агента наугад и проверяя, счастлив ли он. Если да, ничего не происходит; если нет, агент выбирает одну из незанятых ячеек случайным образом и перемещается.

Данная модель приводит к некоторой сегрегации. Из случайной начальной точки кластеры похожих агентов образуются почти сразу. Кластеры растут и усиливаются со временем, пока не появится небольшое количество крупных кластеров, а большинство агентов не будет жить в однородных окрестностях.

Далее будут приведены примеры применения агент-ориентированного подхода при моделировании социальных процессов, основанного на базовой модели Шеллинга, а также будут подробно разобраны коды соответствующих моделей.

# **2. РЕАЛИЗАЦИЯ АГЕНТНО-ОРИЕНТИРОВАННЫХ МОДЕЛЕЙ**

## **2.1. Дискретная стохастическая модель Шеллинга**

Данная модель является классической моделью Шеллинга, лишь с одним отличием: здесь используется три типа агентов. В дискретной модели координаты каждого агента задаются парой значений *(x, y)*, где *x* и *y* – это целочисленные координаты. Также эти координаты выбираются случайным образом, но используя классические функции в python *random.randint* для генерации случайных чисел нельзя гарантировать, что эти числа не будут повторяться, и также не возможно было отбирать пустые области [2]. Поэтому в данной модели сначала генерируется список всех координат среды, а затем случайным образом выбирается пара значений и присваивается к данному агенту, на итерации котором находится цикл. Эти координаты сразу удаляются из списка всех координат, тем самым гарантируется однозначное присваивание.

Для начала нужно импортировать необходимые библиотеки, которые представлены ниже.

#!pip install ipywidgets

import ipywidgets as widgets

from ipywidgets import interact, interactive

import numpy as np

import random

from math import sqrt

import matplotlib.pyplot as plt

from IPython.display import display

%matplotlib inline

import copy

После того, как все необходимые для работы библиотеки будут установлены, можно переходить к описанию единственного и самого главного класса – класса *Agent*. Исходный код находится в Приложении А.

Рассмотрим подробнее процесс его работы. На вход классу поступают три значения: тип агента и координаты *x* и *y*. При помощи функции *get\_distance* вычисляется евклидово расстояние между текущим агентом и агентом, который подается на вход данной функции. Далее располагается одна из самых важных функций в классе, определяющая состояние агента, счастлив он или нет. Для этого запускается цикл по всем агентам, которые передаются как аргумент к функции *happy*, вычисляется расстояние до агентов, не совпадающих с текущем. Далее создается список из этих расстояний и сортируется в порядке возрастания. Тем самым, когда будут отбираться ближайшие соседи, нужно будет взять лишь первые элементы списка *distances*. На выходе функция *happy* выдает значение *true* или *false*. Это зависит есть ли в окрестности текущего агента, количество агентов того же типа большее чем требуемое число *require\_same\_type.*

Функция *update* выполняет основную работу по перемещению агента по среде. На вход подаются список всех агентов и позиции всех координат *pos\_coord*. Далее создается копия списка координат для возможности его редактирования. Затем запускается цикл, который удаляет из всех координат позиции агентов. Тем самым создается новый список *pos\_coord\_loc*, в котором теперь хранятся координаты пустых ячеек. Последний здесь элемент, это цикл *while*, который запускается и работает, пока текущий агент не счастлив и индекс *i* не равен длине списка *pos\_coord\_loc-1*, чтобы избежать ошибки, когда будет браться элемент с индексом *i* из списка *pos\_coord\_loc.* Индекс *i* – это координата из списка пустых клеток в среде. В теле цикла происходит изменение положения агентов по среде, соответствуя переезду человека на новое место, если он не удовлетворен текущем местом жительства. Тем самым, наблюдается эффект сегрегации, то есть кластеризация агентов по каждому типу.

Далее, для более ясного понимания процесса работы необходимо сделать визуализацию с помощью библиотеки *matplotlib*. За это отвечает функция *plot\_distribution*, которая получает на вход два аргумента: *agents* и *cycle\_num*. Первый отвечает за количество агентов, второй отвечает за количество итераций, которые предстоит пройти циклу. Соответственно, количеству итераций будет равно количество графиков, которые получаются на выходе работы функции. Пример работы данной функции представлен на рисунке 1 (график слева – генерация агентов, график справа – агенты разбились на небольшие кластеры).

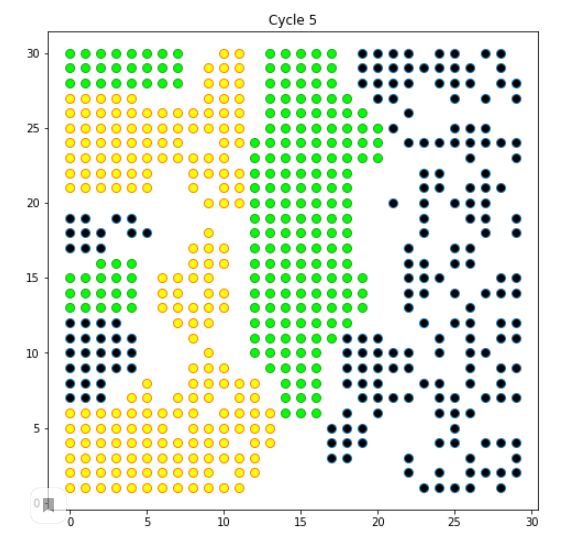
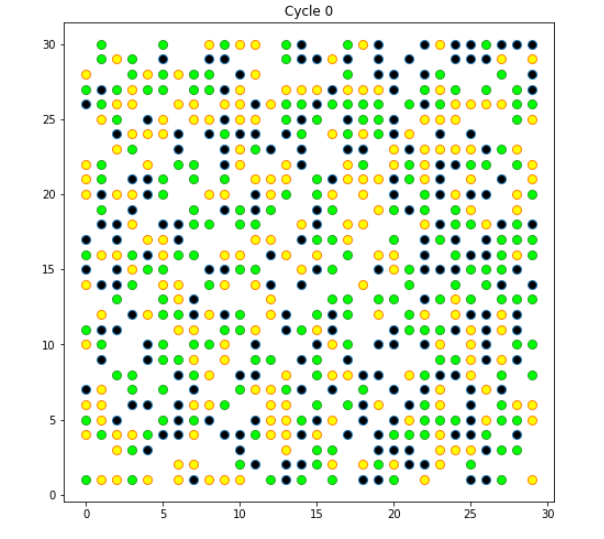


Рисунок 1 – Пример работы функции *plot\_distribution*

Рассмотрим более подробно процесс работы данной функции. Сперва для трех типов агентов создаются по два пустых списка, которые будут соответствовать их координатам. Цикл проходит по всем агентам и определяет тип агента (0, 1 или 2) и добавляет координаты агента в соответствующие списки. Далее происходит настройка размера графика, установка нижнего и верхнего заголовков и с помощью метода *show* происходит генерация графиков.

Для более удобного взаимодействия с моделью было принято решение подключить пользовательский интерфейс. За него отвечает библиотека *ipywidgets*. Из нее необходимо импортировать две функции: *interac*t и *interactive*.

Рассмотрим создание пользовательского интерфейса. С помощью класса *IntSlider* создаем ползунок (в данном случае для регулирования размера), присваивая его переменной *size*, а также присваиваем переменной *num\_of\_type\_0* объект класса *IntText*, который необходим нам для задания количества агентов нулевого типа (оставшиеся виджеты создаются аналогичным образом). Код представлен ниже.

size = widgets.IntSlider(

value=30,

min=10,

max=50,

description='Size')

display(size)

num\_of\_type\_0 = widgets.IntText(

value=200,

description='Type 0')

display(num\_of\_type\_0)

num\_of\_type\_1 = widgets.IntText(

value=200,

description='Type 1')

display(num\_of\_type\_1)

num\_of\_type\_2 = widgets.IntText(

value=200,

description='Type 2')

display(num\_of\_type\_2)

num\_neighbors = widgets.IntSlider(

value = 8,

min=6,

max=10,

description='Neighbours')

display(num\_neighbors)

require\_same\_type = widgets.IntSlider(

value=5,

min=3,

max=6,

description="Same type")

display(require\_same\_type)

button = widgets.Button(description="Simulate", button\_style='info')

output = widgets.Output()

display(button, output)

def on\_button\_clicked(a):

function()

button.on\_click(on\_button\_clicked)

На рисунке 2 показан получившейся пользовательский интерфейс.

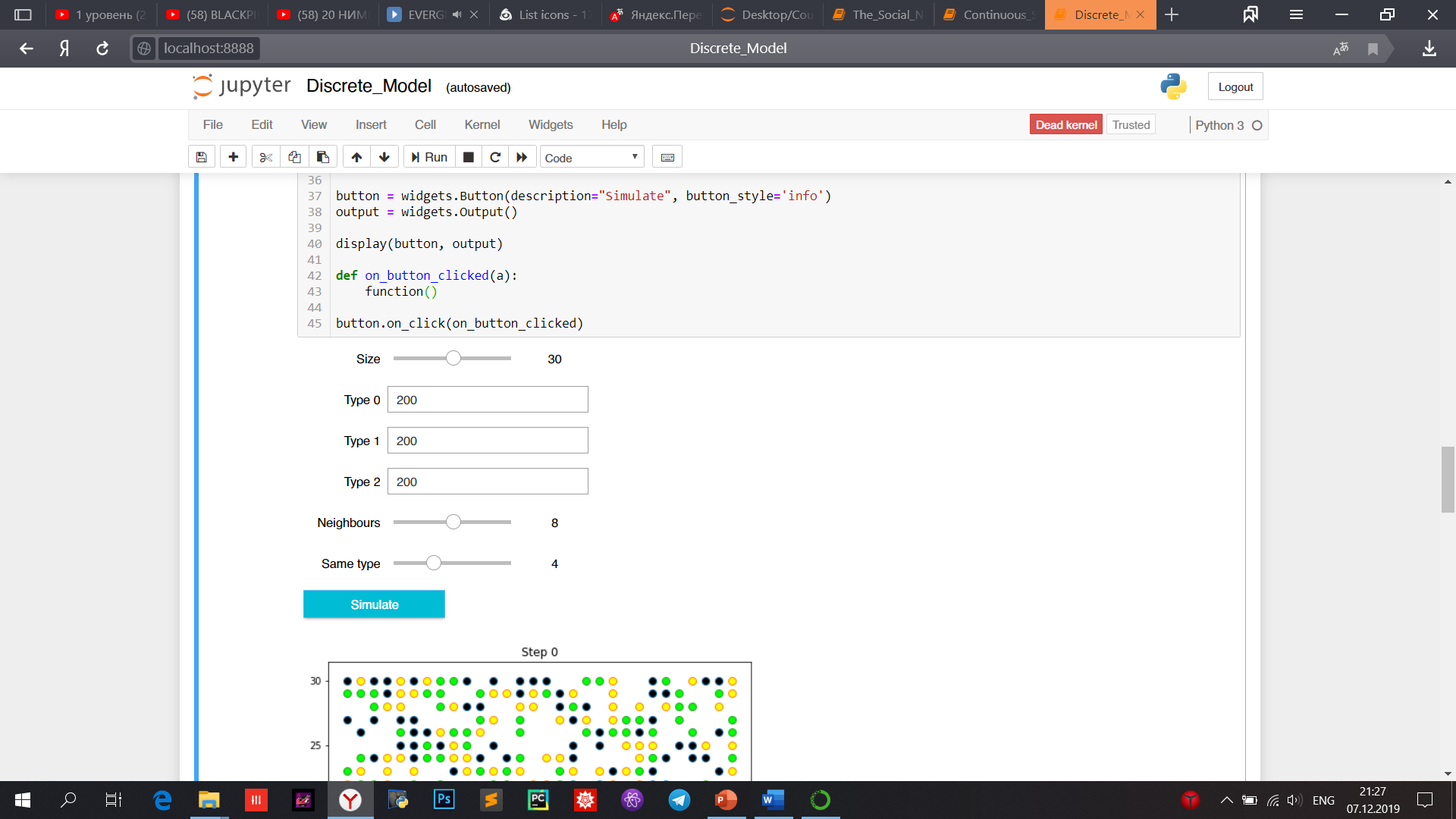


Рисунок 2 – Пользовательский интерфейс

Соответственно, при выборе нужных пользователем параметров и нажатии кнопки *Simulate* вызовется функция *on\_button\_clicked*, которая в свою очередь вызовет функцию *function*, работа которой рассмотрена далее.

Сперва создается список всех координат *pos\_coord*, размером среды. При помощи данного списка и метода *random.sample*, генерируется список *coord*, содержащий набор координат агентов и длинной, равной сумме количеств всех агентов в среде. Далее генерируется список *agents* – это список агентов трех типов и запускается цикл *while*. В нем активируется функция *plot\_distribution*, описанная выше. Цикл работает до тех пор, пока состояние среды не стабилизируется, то есть не будет происходить перемещений, что означает, что каждый агент счастлив и находится на своем местоположении.

Таким образом, реализована дискретная модель Шеллинга с тремя типами агентов, и позволяющая увидеть эффект сегрегации при изменении определенных параметров.

## **2.2. Непрерывная стохастическая модель Шеллинга**

Минус декретной модели в том, что координаты имеют целочисленные значения, но в реальном мире люди не двигаются по идеальной прямой траектории, так лучше и для следующей модели. Вместо целочисленных координат будем рассматривать вещественные, это дает большое разнообразие перемещений агента по среде. Также значительно увеличивается возможное число агентов, а именно в непрерывной модели рассматриваются в качестве координат пары вещественных чисел с округлением до 4 знаков. Сама среда задана от 0 до 1. Таким образом максимальное число агентов, которое помещается в данной модели равно 100000000. В данной модели будут использоваться такие параметры:

1. толерантность – это способность агента терпеть своих соседей с отличным типом от своего;
2. дружелюбность – это показатель общительности и расположенности агента к агентам другого типа;
3. ассимиляция – это способность агента приспосабливаться к агентам иного типа, например, это сравнимо с ситуацией, когда к русскому человеку рядом подселяется американец, намного проще выучить английский язык, чем переезжать. Но это работает не для всех. Агент с низкой способностью к ассимиляции будет менять свое местоположение.

Приведенные выше параметры приближают нас к реальной ситуации, когда человеку не нравится его окружение. Он стремится переехать на новое место, либо ассимилировать. Непрерывная модель нужна нам как одна из задач для достижения главной цели. Поскольку, построив непрерывную модель, уже можно рассматривать процессы не только связанные с переездом на новое место жительства, но и процессы, происходящие в социальных сетях.

Сперва нужно импортировать необходимые для работы модели аналогичные библиотеки из дискретной модели.

Весь основной процесс работы модели заключен в классе *Agent*. Рассмотрим подробнее его работу. Исходный код находится в Приложении Б.

На вход подаются тип агента, порог толерантности (*tolerance\_p*), идентификатор включения использования параметра дружелюбность (friendly), порог дружелюбности (*friendly\_p*) и порог ассимиляции (*assimilation\_p*). Также внутри функции-конструктора задаются координаты и определяются толерантность, дружелюбность, как случайные вещественные числа из промежутка *(0, 1)* и ассимиляция, как случайное цело число от 0 до 100. Функции *get\_distance* и *happy* работают также, как и в случае дискретной модели: они определяют ближайших соседей (сколько укажите) и счастлив ли агент, на том месте, где располагается, соответственно. Основные изменения заключается в функции *update*. Она является основополагающей для сегрегационного эффекта с дополнительными параметрами. Запускается цикл, пока агент не станет счастлив, затем смотрим включен ли идентификатор параметра дружелюбности (friendly). Отличия этих разветвлений будут лишь в использовании параметра дружелюбности. Однако именно эти отличия покажут интересные зависимости. Далее в каждом из этих ветвлений происходят сравнения с пороговыми значениями, которые были даны на вход классу. Изменит ли свое положение агент или ассимилирует, зависит только от этих условий. Ассимиляция для агента – это всего лишь изменение типа агента на тип его ближайших соседей.

Для визуализации полученных данных используется библиотека *matplotlib*. Функция *plot\_distribution*, с помощью которой происходит визуализация данных, аналогична функции визуализации из дискретной модели Шеллинга. Пример работы данной функции представлен на рисунке 3 (график слева – генерация агентов, график справа – агенты разбились на небольшие кластеры).

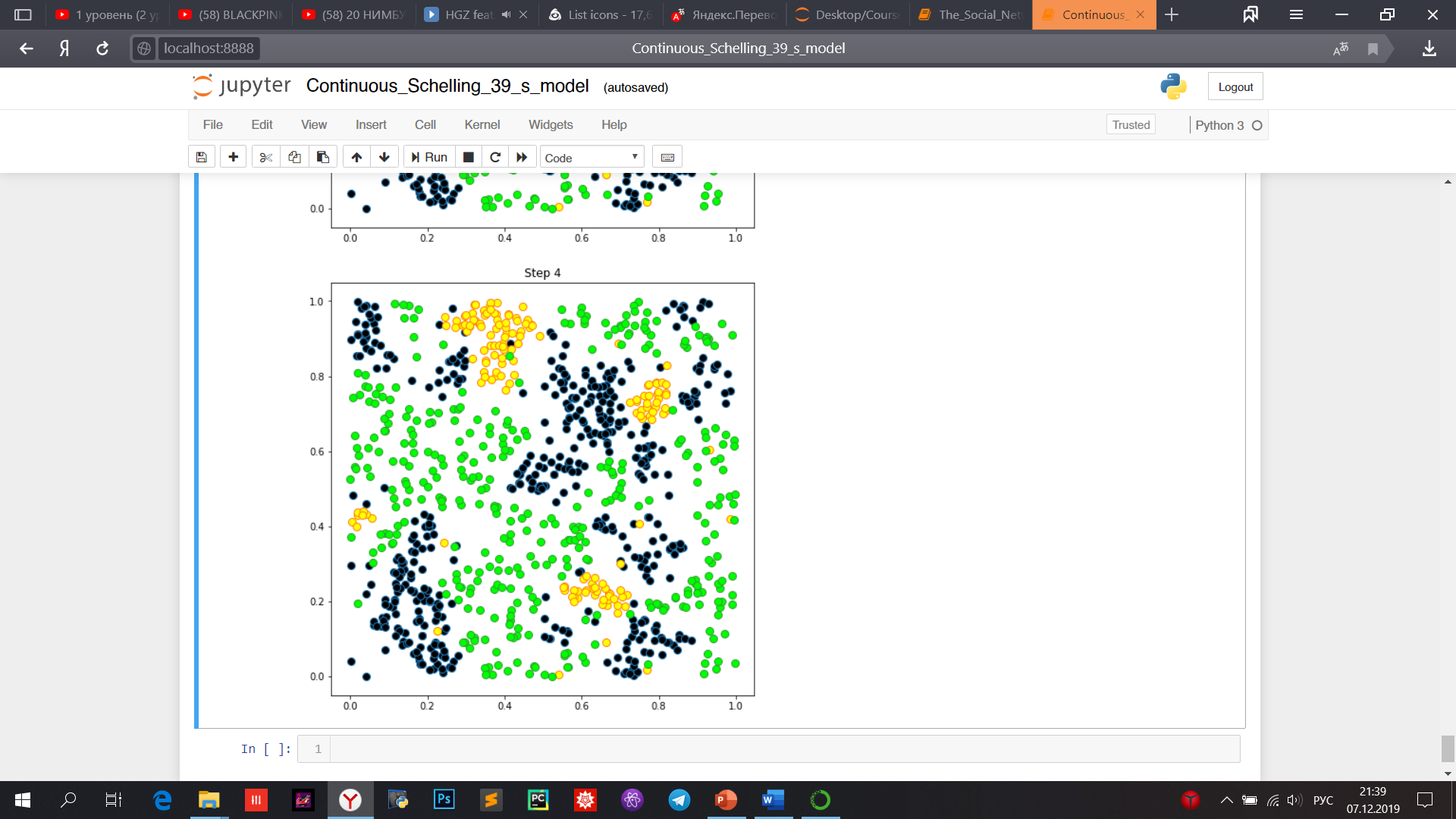


Рисунок 3 – Демонстрация работы функции *plot\_distribution*

Для более удобного взаимодействия с моделью пользователю представлен пользовательский интерфейс, который был сделан с помощью библиотеки *ipywidgets*. Из библиотеки требуется импортировать две функции: *interact* и *interactive*. Код представлен в Приложении В.

Создание интерфейса аналогично созданию интерфейса в дискретной модели Шеллинга, рассмотренной в предыдущей части, за исключением создания флажка для определения степени дружелюбности.

Готовый пользовательский интерфейс представлен на Рисунке 4.

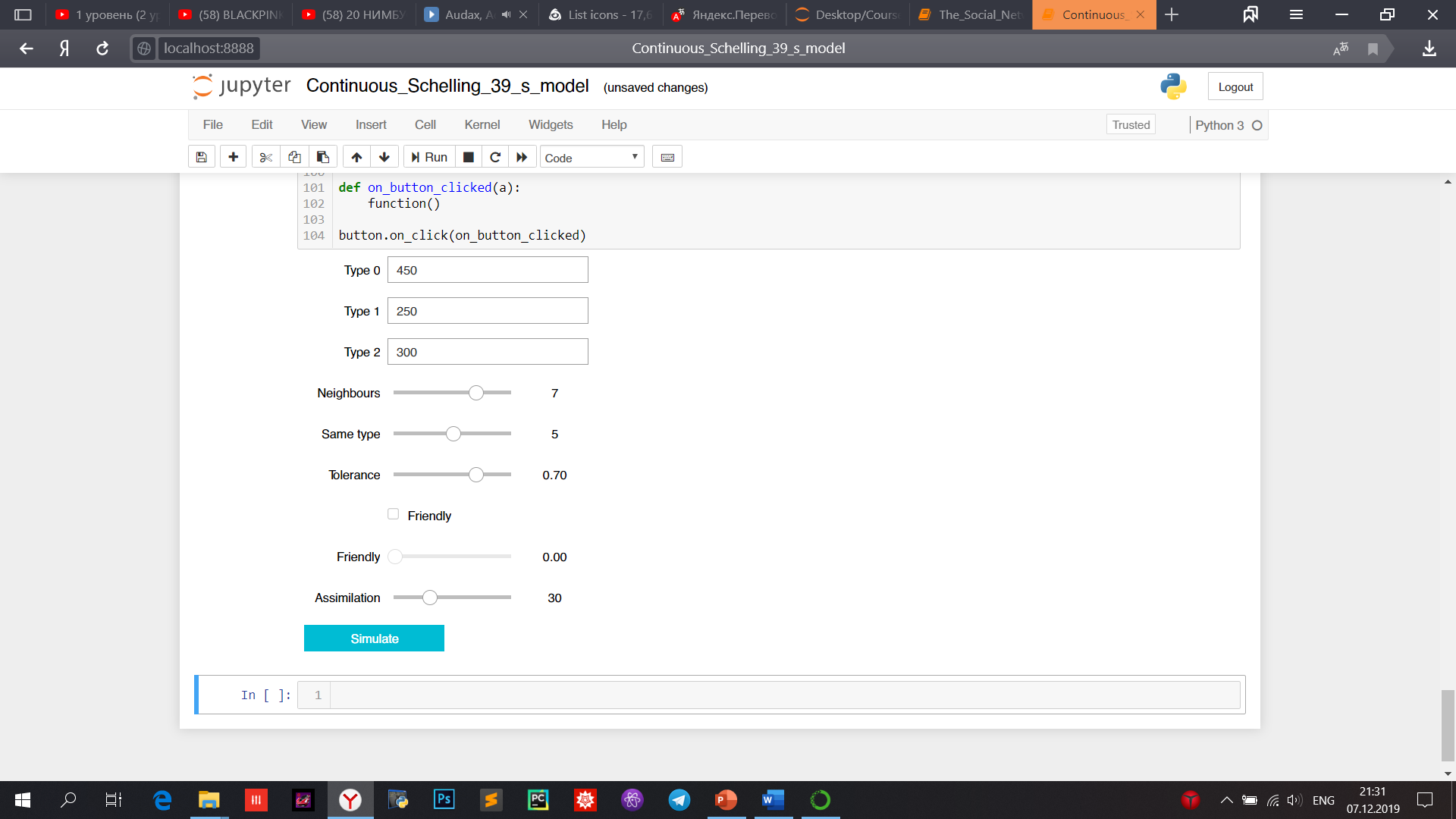


Рисунок 4 – Пользовательский интерфейс для непрерывной модели Шеллинга

После определения доступных параметров и нажатия кнопки *Simulate*, происходит активация функции *on\_button\_clicked*, которая в свою очередь активирует функцию *function*. Как и в дискретной модели, основной функцией, запускающей модель, является функция.

Основной принцип работы в ней сохранился, изменения затронули лишь определение класса агента, поскольку у нас появились дополнительные параметры и теперь не нужен список координат, так как не будет проблем с пустыми областями и одинаковыми координатами агентов. Данное событие маловероятно, так как в среде есть 100000000 позиций, а симуляция рассматривается не больше, чем на 100000 агентов.

Таким образом, непрерывная модель дает гибкость в перемещении агентов и возможность рассмотрения более реалистичных и абстрактных ситуаций. Благодаря построенной непрерывной модели, легко перейти к главной цели, построению модели для социальных сетей.

## **2.3. Модель Шеллинга для социальных сетей**

Как было сказано ранее, здесь будут использоваться результаты, полученные в предыдущем пункте, но увеличится уровень абстракции модели. Социальная сеть довольно сложная для описания вещь, поскольку для понимания ее устройства нужно понимать поведение людей. Человеческая личность – это сложный набор характеристик. Нельзя точно описать поведение индивидов в социуме, так как оно очень хаотично, не предсказуемо и зависит более чем от тысячи факторов, будь то хорошие новости по телевизору, случайно пролитое кофе с утра, плохая погода или ссора с другом. Все это влияет на то, какой индивид будет сегодня, что и как будете делать и, соответственно, влияет на поведение в социальных сетях. Люди могут вместо обыденных сайтов или групп искать информацию в абсолютно другой манере, на других сайтах и в других социальных сетях. Например, доступ к интернету зависит от местоположения, а желание общаться от настроения человека. В данной модели будут использоваться параметры, приближающие нас к реальной ситуации, но которых недостаточно для полной объективной симуляции системы, однако модель построена таким образом, что добавление новых параметров не вызовет труда при дальнейших модификациях. Роль соседей будут играть уже не физические лица, а интернет знакомые, блогеры, группы в социальных сетях, форумы и сайты. А роль счастья будет играть информированность агента, то есть он информируется только тогда, когда в округе есть информированные агенты с безопасной информацией и одинаковыми предпочтениями (про это будет рассказано далее). Исходный код класса *Agent* представлен в Приложении Г.

Для начала работы необходимо импортировать аналогичные библиотеки из дискретной модели.

Рассмотрим более детально предназначение каждой из них. Библиотека *ipywidgets* необходима для создания пользовательского интерфейса. Библиотека *numpy* добавляет поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операции с этими массивами. Модуль *random* представляет функции для генерации случайных чисел, букв. Модуль *math* необходим для работы с числами. Из этой библиотеки нам потребуется лишь одна функция – функция *sqrt* для извлечения квадратного корня из числа. Далее идет библиотека *matplotlib*, необходимая для визуализации различного рода данных.

Далее располагается класс модели, в котором задаются следующие параметры:

1. тип агента;
2. толерантность;
3. дружелюбность;
4. безопасность;
5. популярность;
6. коммуникабельность;
7. предпочтения;
8. доступ к интернету;
9. желание общаться;
10. специальные точки.

В данной модели используется два типа агентов – информированные и неинформированные. Информированность может означать что угодно, будь то владение секретной информацией от коллег или же новость, встречающаяся у вас в новостной ленте. Этим понятием можно манипулировать, в зависимости от того, где применять модель и какой информационный процесс исследовать.

Толерантность и дружелюбность не отличаются от их задания в обычной непрерывной модели Шеллинга, реализованной в п. 2.2.

Безопасность – это случайно выбранный элемент списка , который определяет, насколько источник информации защищен. Повторение значения True дает возможность увеличить число защищенных объектов.

Популярность отражает состояние агента, когда он обладает привилегиями в отношении числа знакомых. В социуме есть разделение людей – обычные граждане и, например, блогеры, такие люди обладают большим охватам аудитории в социальных сетях и как правило имеют большое число знакомств. Так и в данной модели популярные агенты отличаются тем, что могут просматривать большее число соседей в округе.

Коммуникабельность является одним из важнейших понятий для обмена информации в обществе. Чем более агент коммуникабелен, тем быстрее и в больших количествах он получит информацию. Коммуникабельный агент способен просматривать большее число соседей, чем обычный, но не больше, чем популярный. В связи с тем, что популярные люди в большинстве случаев является коммуникабельными, было принято решение задавать данный параметр только между непопулярными агентами.

Предпочтения – это то, что определяет каждого человека. У людей есть хобби или просто любимое дело и с ростом интернет технологий, появилась возможность совершенствоваться в своих навыках при помощи социальных сетей. В данной модели выделено 8 крупных направлений интернет-блогинга, а именно политика, экономика, спорт, искусство, наука, образование, приготовление еды и религия. Каждому агенту задается случайным образом по два предпочтения. Теперь во время определения счастья агента (информированности), у него и соседа, должно совпадать хотя бы одно предпочтение.

Доступ к интернету и желание общаться – это случайно выбранные элементы списка . Доступ к интернету дает возможность зайти в социальную сеть и получить информацию, но это не всегда так. Модель учитывает человеческий фактор, то есть, когда в одной комнате рядом сидят 5 человек в интернете и рассказывают оттуда что-то интересное, подсознательно хочется на это посмотреть. Поэтому агент может получить информацию от людей, находящихся рядом. Желание общаться соответственно определяет будет ли агент получать информацию или нет. Это также зависит от толерантности. Например, если у агента очень высокая толерантность, но он не желает общаться, то он все равно проинформируется.

Последний параметр в данной модели – это специальные точки. По факту это список из координат, при попадании в которые с агентами происходят разные эффекты. Первый тип точек сделан для возникновения случайного эффекта информированности, что равнозначно, когда человек заходит в социальную сеть и видит актуальные новости и посты, вне зависимости от его предпочтений. Также есть второй тип точек, при попадании в которые толерантность и дружелюбность агента может увеличиться на 0.10, что равносильно прогрессу человеческого разума в течение определенного периода времени.

Как и в двух предыдущих модель для визуализация используется библиотека *matplotlib*. За процесс визуализации отвечает функция *plot\_distribution*, подробный разбор которой был приведен в описании к дискретной модели Шеллинга. Пример работы данной функции представлен на рисунке 5 (график слева – генерация агентов, график справа – агенты разбились на небольшие кластеры).

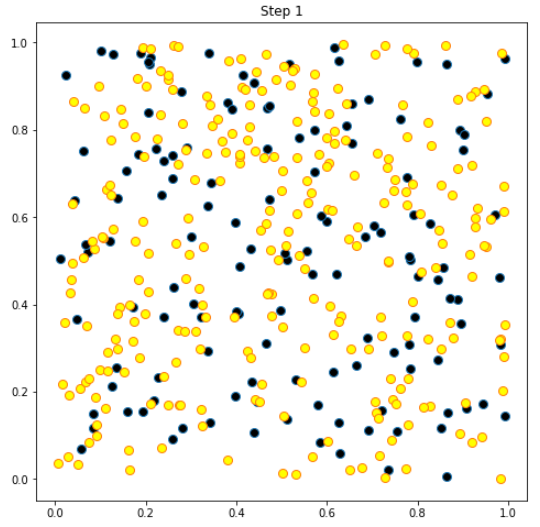
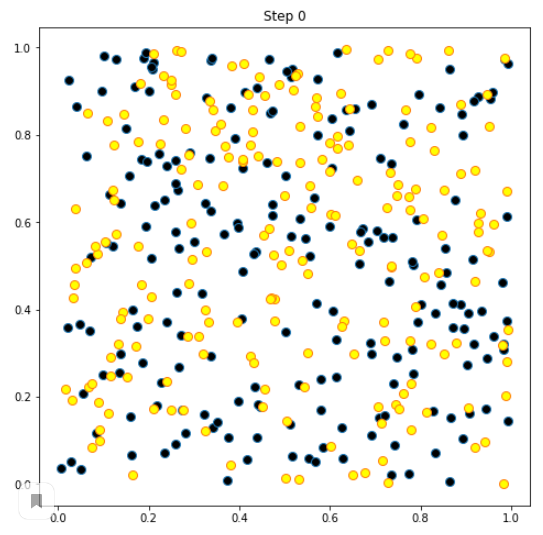


Рисунок 5 – Демонстрация работы функции *plot\_distribution* в модели социальных сетей

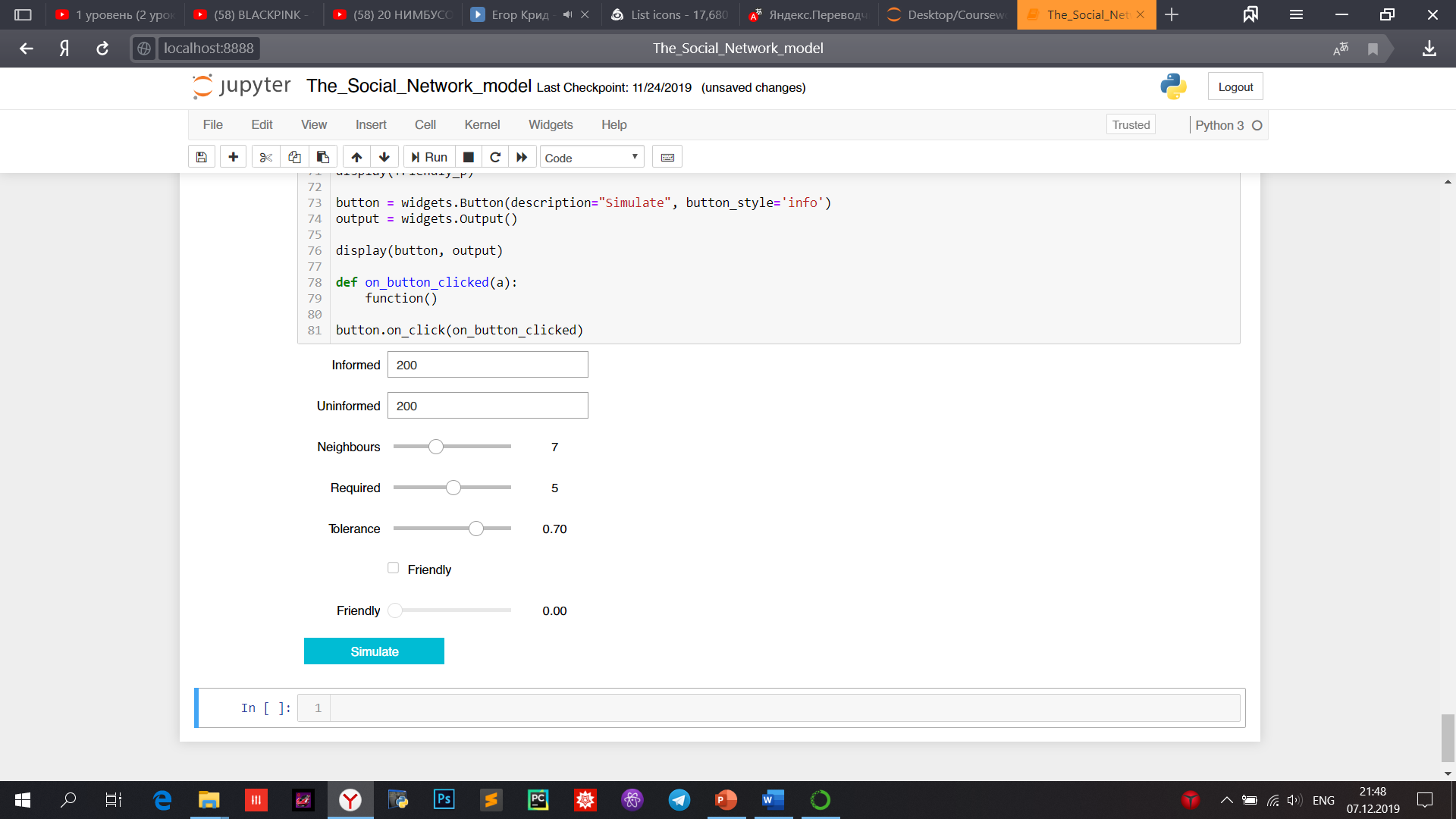
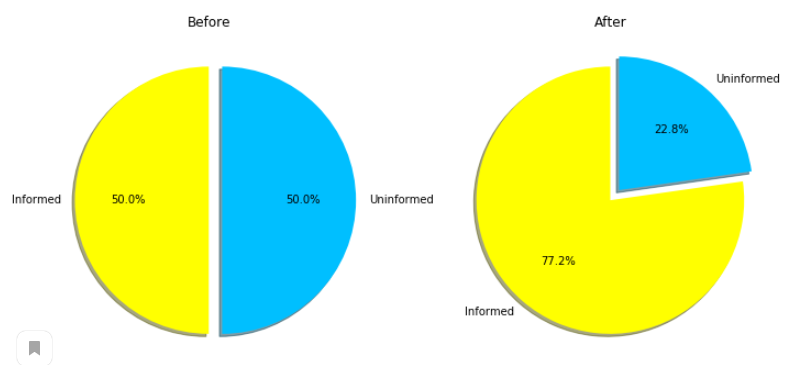
 Пользовательский интерфейс создан с помощью многофункциональной библиотеки *ipywidgets*, работа которой также была разобрана в предыдущих частях. Получившийся интерфейс представлен на рисунке 6.

Рисунок 6 – Пользовательский интерфейс для социальной модели

Как и у предыдущей модели по нажатию кнопки Simulate происходит активация функции *on\_button\_clicked*, которая в свою очередь активирует функцию function, работа которой в целом не отличается работы в непрерывной модели Шеллинга.

 Также, в качестве более детального анализа, с помощью библиотеки *matplotlib* были созданы две круговые диаграммы (см Рисунок 7), а также одна гистограмма (см Рисунок 8), демонстрирующие результат исследования.

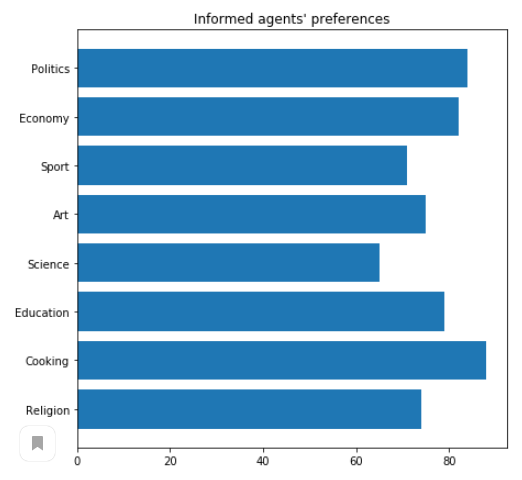
Рисунок 7 – Круговые диаграммы, демонстрирующие результат информированности агентов

Рисунок 8 – Гистограмма, демонстрирующая результаты предпочтений информированных агентов

Таким образом, реализована модифицированная модель Шеллинга для социальных сетей с двумя типами агентов, позволяющая проводить исследования по распространению информации в интернете.

# **ЗАКЛЮЧЕНИЕ**

По итогу работы были рассмотрены теоретические материалы по сегрегационным процессам в социуме и агентно-ориентированному моделированию социальных аспектов общества. Были реализованы три модели. Дискретная модель дает понимание простейшей сегрегации без дополнительных параметров, только по типу агентов.

Далее была реализована непрерывная модель, которая более приближена к реальной ситуации. В ней содержатся дополнительные параметры, позволяющие изучить процесс сегрегации относительно толерантности, дружелюбности и способности к ассимиляции.

Третья модель – это модифицированная непрерывная модель Шеллинга. Для ее работы был произведен подбор параметров, влияющих на информированность людей в интернете, в частности в социальных сетях. Данная модель позволяет рассмотреть сегрегацию по предпочтениям пользователя. Для всех моделей реализован пользовательский интерфейс, позволяющий быстро и удобно манипулировать параметрами.

В конце работы продемонстрирован пример применения одной из моделей, позволяющий увидеть, как происходит информированность в социальных сетях. По итогу, создается гистограмма по предпочтениям пользователей и количеству понадобившихся итераций для информированности агентов.

Таким образом, выполнены все задачи и цель курсовой работы. Также приведен пример работы одной из моделей и продемонстрированы возможности языка Python в агентно-ориентированном моделировании.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Дауни А.Б. Изучение сложных систем с помощью Python / А.Б. Дауни; пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2019, – 110 с.
2. Макаров В.Л. Агент-ориентированные модели: мировой опыт и технические возможности реализации на суперкомпьютерах: науч. ст. / В.Л. Макаров, А.Р. Бахтизин, Е.Д. Сушко, В.А. Васенини, В.А. Борисов, В.А. Роганов, 2015, - с. 252-262
3. Avetisov V. Phase transitions in social networks inspired by the Schelling model / A. Gorsky, S. Maslov, S. Nechaev, O. Valba., 2018
4. Henry A. Emergence of segregation in evolving social networks / P. Praat, C. Zhang., PNAS, 2008.
5. Schelling T. Models of Segregation. American Economic Review, [**Advances in Applied Sociology**](https://www.scirp.org/journal/Home.aspx?JournalID=1002)**,**[Vol.3 No.8](https://www.scirp.org/journal/Home.aspx?IssueID=4169), 2013, - 59 с.
6. Sargent T. J., John Stachurski. Schelling’s Segregation Model, 2019, - 12 с.

# **ПРИЛОЖЕНИЕ А**

**Исходный код дискретной модели Шеллинга**

np.random.seed(10)

class Agent:

def \_\_init\_\_(self, a\_type, x, y):

self.tol = round(np.random.uniform(0,1),2)

self.x = x

self.y = y

self.a\_type = a\_type

self.location = [self.x, self.y]

#Вычисление евклидового расстояния

def get\_distance(self, other):

a = (self.location[0] - other.location[0])\*\*2

b = (self.location[1] - other.location[1])\*\*2

return sqrt(a + b)

def happy(self, agents):

distances = []

for agent in agents:

if self != agent:

distance = self.get\_distance(agent)

distances.append((distance, agent))

distances.sort(key=lambda x: x[0])

neighbors = [agent for d, agent in distances[:num\_neighbors.value]]

num\_same\_type = sum(self.a\_type == agent.a\_type for agent in neighbors)

return num\_same\_type >= require\_same\_type.value

def update(self, agents, pos\_coord):

#Составляем список пустых ячеек

self.pos\_coord\_loc = copy.deepcopy(pos\_coord)

for agent in agents:

self.pos\_coord\_loc.remove(agent.location)

i = 0

while not self.happy(agents) and i != len(self.pos\_coord\_loc)-1:

self.location = self.pos\_coord\_loc[i]

i += 1

def plot\_distribution(agents, cycle\_num):

x\_values\_0, y\_values\_0 = [], []

x\_values\_1, y\_values\_1 = [], []

x\_values\_2, y\_values\_2 = [], []

for agent in agents:

x, y = agent.location

if agent.a\_type == 0:

x\_values\_0.append(x)

y\_values\_0.append(y)

elif agent.a\_type ==1:

x\_values\_1.append(x)

y\_values\_1.append(y)

else:

x\_values\_2.append(x)

y\_values\_2.append(y)

fig, ax = plt.subplots(figsize=(8, 8))

plot\_args = {'markersize': 8, 'alpha': 1}

ax.set\_facecolor('white')

ax.plot(x\_values\_0, y\_values\_0, 'o', markerfacecolor='black', \*\*plot\_args)

ax.plot(x\_values\_1, y\_values\_1, 'o', markerfacecolor='yellow', \*\*plot\_args)

ax.plot(x\_values\_2, y\_values\_2, 'o', markerfacecolor='lime', \*\*plot\_args)

ax.set\_title(f'Step {cycle\_num-1}')

plt.show()

**ПРИЛОЖЕНИЕ А**

def function():

count = 1

pos\_coord = [[x, y] for x in range(size.value) for y in range(1, size.value+1)]

coord = random.sample(pos\_coord, num\_of\_type\_0.value + num\_of\_type\_1.value + num\_of\_type\_2.value)

agents = [Agent(0, coord[i][0], coord[i][1]) for i in range(num\_of\_type\_0.value)]

agents.extend(Agent(1, coord[i + num\_of\_type\_0.value][0], coord[i + num\_of\_type\_0.value][1]) for i in range(num\_of\_type\_1.value))

agents.extend(Agent(2, coord[i + num\_of\_type\_0.value + num\_of\_type\_1.value][0], coord[i + num\_of\_type\_0.value + num\_of\_type\_1.value][1]) for i in range(num\_of\_type\_2.value))

while True:

plot\_distribution(agents, count)

count += 1

no\_one\_moved = True

for agent in agents:

old\_location = agent.location

agent.update(agents, pos\_coord)

if agent.location != old\_location:

no\_one\_moved = False

if no\_one\_moved:

break

# **ПРИЛОЖЕНИЕ Б**

**Исходный код класса Agent непрерывной модели Шеллинга**

class Agent:

def \_\_init\_\_(self, a\_type, tolerance\_p, friendly, friendly\_p, assimilation\_p):

self.a\_type = a\_type

self.change\_location()

self.tol = round(uniform(0, 1), 4)

self.fr = round(uniform(0, 1), 2)

self.friendly = friendly

self.assimilation = random.randint(0, 100)

def change\_location(self):

self.location = round(uniform(0, 1),4), round(uniform(0, 1),4)

def get\_distance(self, other):

a = (self.location[0] - other.location[0])\*\*2

b = (self.location[1] - other.location[1])\*\*2

return sqrt(a + b)

def happy(self, agents):

distances = []

for agent in agents:

if self != agent:

distance = self.get\_distance(agent)

distances.append((distance, agent))

distances.sort(key=lambda x: x[0])

neighbors = [agent for d, agent in distances[:local\_neighbors.value]]

num\_same\_type = sum(self.a\_type == agent.a\_type for agent in neighbors)

return num\_same\_type >= require\_same\_type.value

def update(self, agents):

while not self.happy(agents):

if not self.friendly:

if self.tol < tolerance\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

elif self.tol < tolerance\_p.value and self.assimilation > assimilation\_p.value:

if self.a\_type == 0 or self.a\_type == 1:

self.a\_type = 2

return

elif self.a\_type == 1 or self.a\_type == 2:

self.a\_type = 0

return

else:

self.a\_type = 1

return

elif self.tol > tolerance\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

else:

return

else:

if self.tol < tolerance\_p.value and self.fr > friendly\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

elif self.tol < tolerance\_p.value and self.fr > friendly\_p.value and self.assimilation > assimilation\_p.value:

if self.a\_type == 0 or self.a\_type == 1:

self.a\_type = 2

return

**ПРИЛОЖЕНИЕ Б**

elif self.a\_type == 1 or self.a\_type == 2:

self.a\_type = 0

return

else:

self.a\_type = 1

return

elif self.tol > tolerance\_p.value and self.fr > friendly\_p.value and self.assimilation > assimilation\_p.value:

if self.a\_type == 0 or self.a\_type == 1:

self.a\_type = 2

return

elif self.a\_type == 1 or self.a\_type == 2:

self.a\_type = 0

return

else:

self.a\_type = 1

return

elif self.tol > tolerance\_p.value and self.fr > friendly\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

elif self.tol < tolerance\_p.value and self.fr < friendly\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

elif self.tol < tolerance\_p.value and self.fr < friendly\_p.value and self.assimilation > assimilation\_p.value:

self.change\_location()

elif self.tol > tolerance\_p.value and self.fr < friendly\_p.value and self.assimilation > assimilation\_p.value:

if self.a\_type == 0 or self.a\_type == 1:

self.a\_type = 2

return

elif self.a\_type == 1 or self.a\_type == 2:

self.a\_type = 0

return

else:

self.a\_type = 1

return

elif self.tol > tolerance\_p.value and self.fr < friendly\_p.value and self.assimilation < assimilation\_p.value:

self.change\_location()

else:

return

# **ПРИЛОЖЕНИЕ В**

**Исходный код интерфейса непрерывной модели Шеллинга**

number\_of\_type0 = widgets.IntText(

value = 450,

description="Type 0")

display(number\_of\_type0)

#количество агентов типа 1

number\_of\_type1 = widgets.IntText(

value = 250,

description="Type 1")

display(number\_of\_type1)

#количество агентов типа 2

number\_of\_type2 = widgets.IntText(

value = 300,

description="Type 2")

display(number\_of\_type2)

#количество соседей

local\_neighbors = widgets.IntSlider(

value=7,

min=0,

max=10,

step=1,

description='Neighbours',

disabled=False,

continuous\_update=False,

orientation='horizontal',

readout=True,

readout\_format='d'

)

display(local\_neighbors)

#reqires same time

require\_same\_type = widgets.IntSlider(

value=5,

min=0,

max=10,

step=1,

description='Same type',

disabled=False,

continuous\_update=False,

orientation='horizontal',

readout=True,

readout\_format='d'

)

display(require\_same\_type)

tolerance\_p = widgets.FloatSlider(

value=0.7,

min=0,

max=1.0,

step=0.01,

description='Tolerance',

disabled=False,

continuous\_update=False,

orientation='horizontal',

)

display(tolerance\_p)

**ПРИЛОЖЕНИЕ В**

interact(f, x=widgets.Checkbox(description='Friendly',disabled=False))

#процент дружелюбности

friendly\_p = widgets.FloatSlider(

value=0,

min=0,

max=1.0,

step=0.01,

description='Friendly',

disabled=True,

continuous\_update=False,

orientation='horizontal',

)

display(friendly\_p)

#процент ассимиляции

assimilation\_p = widgets.IntSlider(

value=30,

min=0,

max=100,

step=1,

description='Assimilation',

disabled=False,

continuous\_update=False,

orientation='horizontal',

readout=True,

readout\_format='d'

)

display(assimilation\_p)

button = widgets.Button(description="Simulate", button\_style='info')

output = widgets.Output()

display(button, output)

def on\_button\_clicked(a):

function()

button.on\_click(on\_button\_clicked)

# **ПРИЛОЖЕНИЕ Г**

**Исходный код непрерывной модели для социальных сетей**

np.random.seed(10)

transitions = []

information\_types = ["Politics", "Economy", "Sport", "Art", "Science", "Education", "Cooking", "Religion"]

special\_points = [(round(uniform(0, 1),4), round(uniform(0, 1),4)) for i in range(10000)]

special\_points2 = [(round(uniform(0, 1),4), round(uniform(0, 1),4)) for i in range(100000)]

def f(x):

if x:

friendly\_p.disabled = False

friendly\_p.value = 0.75

else:

friendly\_p.disabled = True

friendly\_p.value = 0

class Agent:

def \_\_init\_\_(self, a\_type, tolerance\_p, friendly, friendly\_p):

self.a\_type = a\_type

self.change\_location()

self.wish = random.randint(0, 1)

self.tol = round(uniform(0, 1), 2)

self.fr = round(uniform(0, 1), 2)

self.friendly = friendly

self.popular = ''

self.sociability = random.choice(["S","US"])

self.preferences = random.sample(information\_types, 2)

self.access\_internet = random.choice([True, False, True])

self.safety = random.choice([True, True, True, False])

def change\_location(self):

self.location = round(uniform(0, 1),4), round(uniform(0, 1),4)

def get\_distance(self, other):

a = (self.location[0] - other.location[0])\*\*2

b = (self.location[1] - other.location[1])\*\*2

return sqrt(a + b)

def inform(self, agents):

if self.a\_type == 1:

self.popular = 'p'

else:

self.popular = 'up'

def happy(self, agents):

distances = []

for agent in agents:

if self != agent:

distance = self.get\_distance(agent)

distances.append((distance, agent))

distances.sort(key=lambda x: x[0])

if self.popular == 'up':

if self.sociability == 'S':

neighbors = [agent for d, agent in distances[:local\_neighbors.value + 4]]

num\_same\_type = sum((agent.a\_type == 1 and (len(list(set(self.preferences) & set(agent.preferences))) != []))

for agent in neighbors)

return num\_same\_type >= require\_same\_type.value

else:

**ПРИЛОЖЕНИЕ Г**

neighbors3 = [agent for d, agent in distances[:local\_neighbors.value]]

num\_same\_type3 = sum((agent.a\_type == 1 and (len(list(set(self.preferences) & set(agent.preferences))) != []))

for agent in neighbors3)

return num\_same\_type3 >= require\_same\_type.value

else:

neighbors2 = [agent for d, agent in distances[:local\_neighbors.value + 8]]

num\_same\_type2 = sum((agent.a\_type == 1 and self.safety and (len(list(set(self.preferences) & set(agent.preferences))) != []))

for agent in neighbors2)

return num\_same\_type2 >= require\_same\_type.value

def update(self, agents):

t2 = False

count = 0

for i in special\_points:

if self.location == i:

self.a\_type = 1

count += 1

for i in special\_points2:

if self.location == i:

t2 = True

if t2 and self.friendly:

self.tol += 0.10

self.fr += 0.10

elif t2 and not self.friendly:

self.tol += 0.10

else:

if self.friendly:

if self.happy(agents) and self.a\_type == 0:

if self.fr < friendly\_p.value and self.wish == 0 and self.tol < tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 1 and self.tol < tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 0 and self.tol > tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol > tolerance\_p.value:

self.a\_type = 1

count += 1

elif self.fr < friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol > tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 0 and self.access\_internet and self.tol < tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol < tolerance\_p.value:

self.a\_type = 1

count += 1

elif self.fr > friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol < tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 0 and self.tol > tolerance\_p.value:

self.a\_type = 1

count += 1

elif self.fr > friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol > tolerance\_p.value:

self.a\_type = 1

**ПРИЛОЖЕНИЕ Г**

count += 1

elif self.fr > friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol > tolerance\_p.value:

return

else:

return

elif self.happy(agents) and self.a\_type == 1:

return

elif not self.happy(agents) and self.a\_type == 0:

if self.fr < friendly\_p.value and self.wish == 0 and self.tol < tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol < tolerance\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.fr < friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol < tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 0 and self.tol > tolerance\_p.value:

return

elif self.fr < friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol > tolerance\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.fr < friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol > tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 0 and self.tol < tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol < tolerance\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.fr > friendly\_p.value and self.wish == 1 and self.tol < tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 0 and self.access\_internet and self.tol > tolerance\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.fr > friendly\_p.value and self.wish == 0 and not self.access\_internet and self.tol > tolerance\_p.value:

return

elif self.fr > friendly\_p.value and self.wish == 1 and self.access\_internet and self.tol > tolerance\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.fr > friendly\_p.value and self.wish == 1 and not self.access\_internet and self.tol > tolerance\_p.value:

return

else:

return

else:

return

else:

**ПРИЛОЖЕНИЕ Г**

if self.happy(agents) and self.a\_type == 0:

if self.wish == 0 and self.tol < tolerance\_p.value:

return

elif self.wish == 1 and self.tol < tolerance\_p.value:

return

elif self.wish == 0 and self.tol > tolerance\_p.value:

return

elif self.wish == 1 and self.access\_internet and self.tol > tolerance\_p.value:

self.a\_type = 1

count += 1

elif self.wish == 1 and not self.access\_internet and self.tol > tolerance\_p.value:

return

else:

return

elif self.happy(agents) and self.a\_type == 1:

return

elif not self.happy(agents) and self.a\_type == 0:

if self.wish == 0 and self.tol < tolerence\_p.value:

return

elif self.wish == 1 and self.access\_internet and self.tol < tolerence\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.wish == 1 and not self.access\_internet and self.tol < tolerence\_p.value:

return

elif self.wish == 0 and self.tol > tolerence\_p.value:

return

elif self.wish == 1 and self.access\_internet and self.tol > tolerence\_p.value:

while not self.happy(agents):

self.change\_location()

count += 1

elif self.wish == 1 and not self.access\_internet and self.tol > tolerence\_p.value:

return

else:

return

else:

return

transitions.append(count)