



---

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Брянский государственный технический университет**

---

Утверждаю

Ректор университета

\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_ 2019г.

**КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ**

**ПОСТРОЕНИЕ МНОГОАГЕНТНЫХ МОДЕЛЕЙ  
В ПРОГРАММНОМ КОМПЛЕКСЕ ANYLOGIC**

Методические указания  
к выполнению лабораторной работы №11  
для студентов очной формы обучения  
по направлению подготовки  
09.03.01 «Информатика и вычислительная техника»

**Брянск 2019**

**УДК 004.65**

Компьютерное моделирование. Построение многоагентных моделей в программном комплексе Anylogic [Электронный ресурс]: методические указания к выполнению лабораторной работы № 11 для студентов очной формы обучения по направлению подготовки 09.03.01 «Информатика и вычислительная техника». – Брянск: БГТУ, 2019. – 22 с.

Разработали:

А.А.Трубакова,

ст.преп.;

А.О. Трубаков,

канд. техн. наук, доц.

Рекомендовано кафедрой «Информатика и программное обеспечение»  
БГТУ (протокол № 9 от 07.06.2019г.)

**Методические указания публикуются в авторской редакции**

## 1. ЦЕЛЬ РАБОТЫ

Целью работы является ознакомление с основными принципами построения многоагентных моделей в среде *AnyLogic* на примере моделирования броуновского движения.

Продолжительность работы – 2 часа.

## 2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучение основных возможностей и принципов построения многоагентных моделей в среде имитационного моделирования *AnyLogic*.
2. Построение модели броуновского движения шаров в среде *AnyLogic*.
3. Самостоятельная часть.

## 3. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Моделирование агентов и многоагентных систем в *AnyLogic* не представляет сложности ни в концептуальном, ни в техническом аспекте. Основной концепцией *AnyLogic* является концепция, что модель состоит из активных объектов, имеющих каждый свои правила поведения и взаимодействующих через явно определенные интерфейсы (рис.1). Поэтому агентный подход к построению моделей является в *AnyLogic* совершенно естественным.

Для разработки агентных моделей *AnyLogic* предоставляет мощные графические средства: стейтчарты, события, таймеры, синхронное и асинхронное планирование событий, библиотеки ранее определенных агентов, а также возможность включить в модель фрагменты кода на языке *Java*.

В *AnyLogic* агент реализуется с помощью базового объекта *AnyLogic* – активного объекта. Активный объект имеет параметры, которые можно менять извне, переменные, которые можно считать памятью агента, а также поведение.

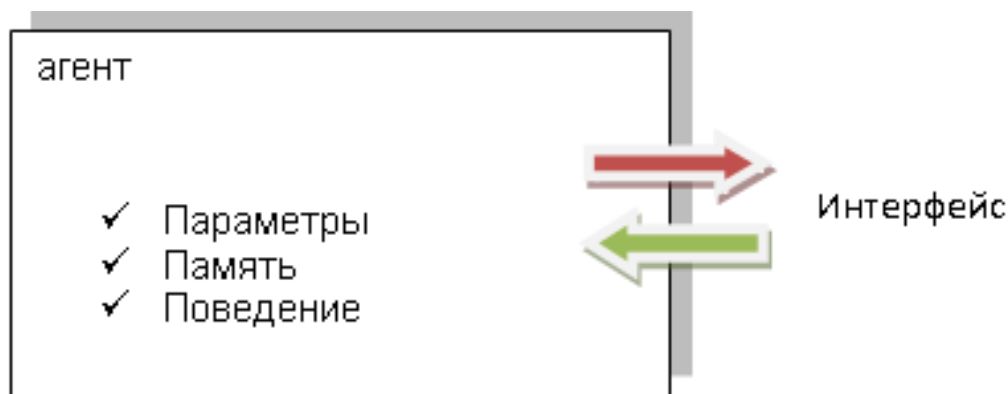


Рис. 1. Общая концепция агента в *AnyLogic*

Поведение агентов может выражать, например, правила действия агента или законы перемещения агента в пространстве, изменения его социального статуса, переходы в разные возрастные или социальные группы и т.д.

Взаимодействия агентов в *AnyLogic* может быть организовано через:

- явно определенные интерфейсные объекты (порты и интерфейсные переменные);
- посылку сообщения стейтчарту;
- вызов функции агента;
- изменение параметра.

## 4. ПОСТРОЕНИЕ МОДЕЛИ БРОУНОВСКОГО ДВИЖЕНИЯ ШАРОВ

### 4.1. Постановка проблемы

Рассмотрим динамическую систему, имитирующую столкновения частиц при хаотичном броуновском движении. Эта модель позволяет изучить агентный подход к решению задач, связанных со взаимодействием отдельных объектов.

Первым шагом при создании агентной модели является создание агентов. Агент является основным строительным блоком агентной модели. Агентная модель состоит из множества агентов и их окружения. Для каждого агента задается набор правил, согласно которым он взаимодействует с другими агентами; это взаимодействие и определяет общее поведение системы. В нашей модели агентами будут сталкивающиеся шары.

Одинаковые по размеру и массе шары движутся в ограниченном двумерном пространстве (камере) без потери энергии с упругим соударением.

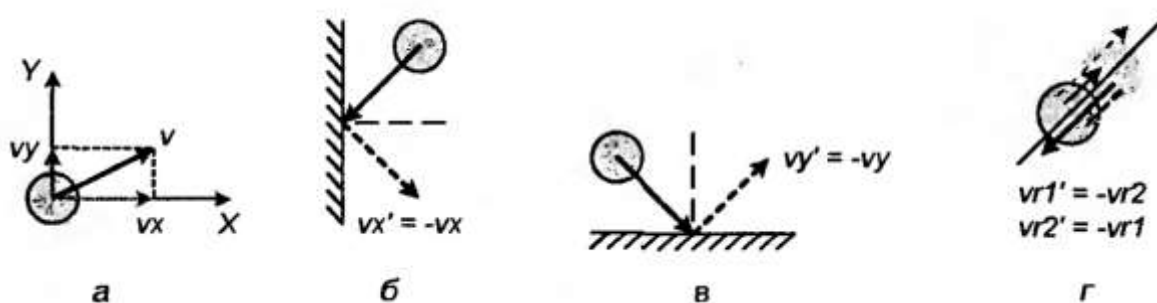


Рис. 2. Изменение скорости при соударениях

Рис. 2 показывает примеры столкновений шара с препятствием. Если препятствие – горизонтальная либо вертикальная стенка, то новое значение скорости шара получить легко: соответствующая составляющая скорости шара изменяет свое направление. Если препятствием является другой шар, то при центральном упругом столкновении таких шаров они просто обмениваются скоростями. Мы предполагаем в нашей модели, что силы трения очень малы по сравнению с упругими силами. Поэтому при нецентральном столкновении действием сил трения можно пренебречь. Соединим центры масс сталкивающихся шаров прямой и разложим скорость каждого шара на нормальную составляющую, направленную вдоль линии центров двух шаров, и касательную составляющую, перпендикулярную к ней. Поскольку силы трения отсутствуют, изменяться будут только нормальные составляющие скорости шаров ( $vr$ ) так же, как при центральном столкновении, т. е. при упругом столкновении шары с одной массой обмениваются нормальными составляющими скоростей.

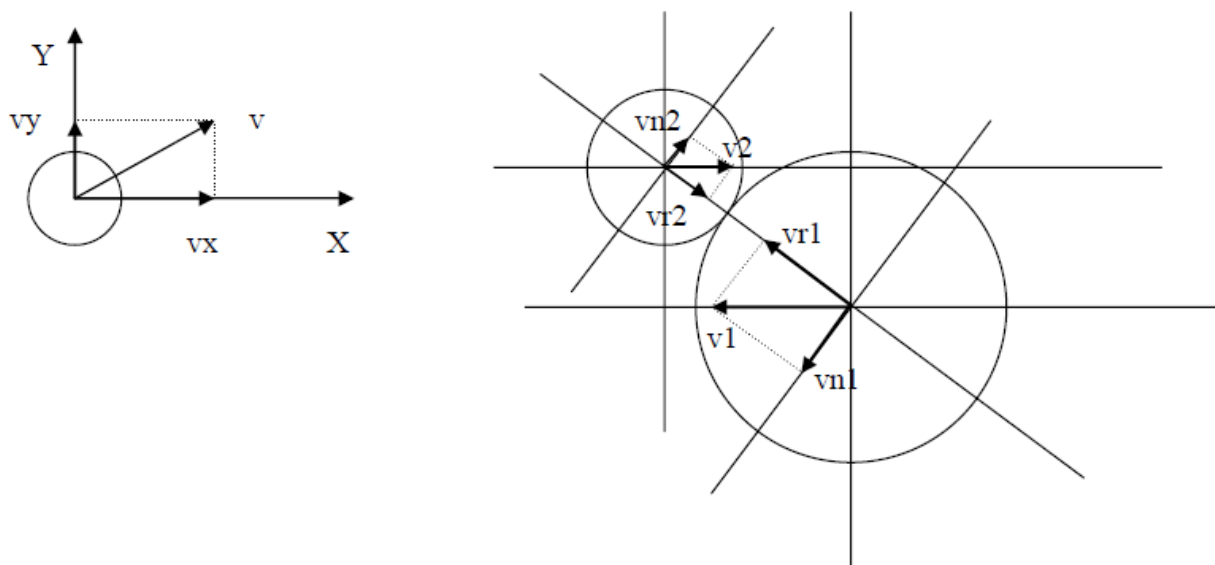


Рис. 3. Изменение скорости при соударении шаров друг с другом

#### 4.2. Построение модели

Создаем модель. Добавим в модель агента *Ball* со своим собственным поведением. Активный объект – шар,двигающийся в ограниченном двумерном пространстве и сталкивающийся с другими шарами. Корневой активный агент *Main* будет включать массив таких шаров, т.е. представлять собой среду для жизни шаров.

Модель будет состоять из двух классов активных объектов: сталкивающихся шаров и плоского пространства, содержащего эти шары.

Добавим пять параметров агенту *Ball*:

- $X_{max}$  и  $Y_{max}$  – это максимальные координаты двумерной прямоугольной области *area*, в которой двигаются шары;
- $v$  – начальная скорость шара;
- $r$  – радиус шара;
- $g$  – ускорение свободного падения.

Параметры  $r$  и  $g$  шара определены как глобальные (в терминах языка *Java* статические), т.е. относящиеся ко всему классу шаров, а не к каждому шару (экземпляру класса). Эти параметры мы будем изменять в экспериментах с моделью.

Модель шара имеет три вещественных переменных (переменные компонентов агента): положение его центра  $traceX$  и  $traceY$  в координатах  $X$  и  $Y$  и составляющая скорости –  $vx$  по координате  $x$ . Добавим три накопителя:  $x$ ,  $y$ ,  $vy$ . В свойствах накопителей укажем Режим задания уравнения – произвольный. Три переменных описаны интегралами. Укажем координаты для  $x$ ,  $y$ ,  $vy$  соответственно:

$$\begin{aligned} d(x)/dt &= vx; \\ d(y)/dt &= vy; \\ d(vy)/dt &= g. \end{aligned} \tag{1}$$

Отообразим зависимость между переменными с помощью *Связей* рис. 4.

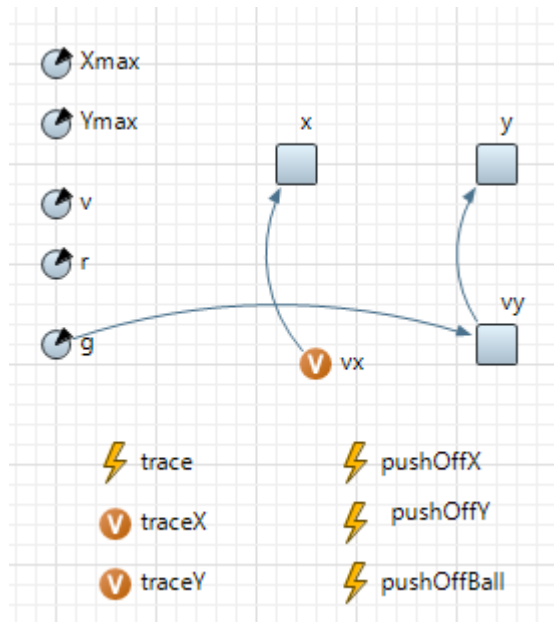


Рис. 4. Связи для определения поведения агента Ball

Поведение шара – это свободное движение, определяемое скоростями  $vx$  и  $vy$ .

Добавим событие *trace* для добавления текущей позиции шара и в свойствах этого события в пункте Действие запишем следующее:

```
for (int i = traceX.length - 1; i > 0; i--)
{
    traceX[i] = traceX[i-1];
    traceY[i] = traceY[i-1];
}
```

$traceX[0] = x;$

$traceY[0] = y;$

Для события *trace* добавим режим срабатывания (циклически).

**trace - Событие**

Тип события: По таймауту

Режим: Циклический

☒ Использовать модельное время ☐ Использовать календарные даты

Время первого срабатывания (абс.): 0 дни

Время срабатывания: 14.05.2019 8:00:00

Период: 100 дни

☒ Записывать лог в базу данных

**Действие**

```
for (int i = traceX.length - 1; i > 0; i--)
{
    traceX[i] = traceX[i-1];
    traceY[i] = traceY[i-1];
}
traceX[0] = x;
traceY[0] = y;
```

Рис. 5. Режим срабатывания события *trace*

Для того, чтобы сгенерировать начальное значение популяции шаров и их хаотичное расположение в пределах некоторой прямоугольной области используем переменные *traceX* и *traceY*, при этом определив их тип в свойствах, как показано на рисунке ниже.

**traceX - Переменная**

Имя: traceX ☒ Отображать имя ☐ Исключить

Видимость: ☒ да

Тип: Другой... double[]

Начальное значение: new double[20];

Рис. 6. Типы данных для определения положения центра шаров

Столкновение с каждым видом препятствия отслеживается специальным объектом – событием, которое выполняет определенные действия. В нашей модели определено три события, по одному на каждый тип столкновений. Для того, чтобы шарик мог оттолкнуться от горизонтального и



вертикального ограничений, которые мы позже зададим в виде прямоугольной области, а также, от другого шара, добавим три события: *pushOffX*, *pushOffY*, *pushOffBall*.

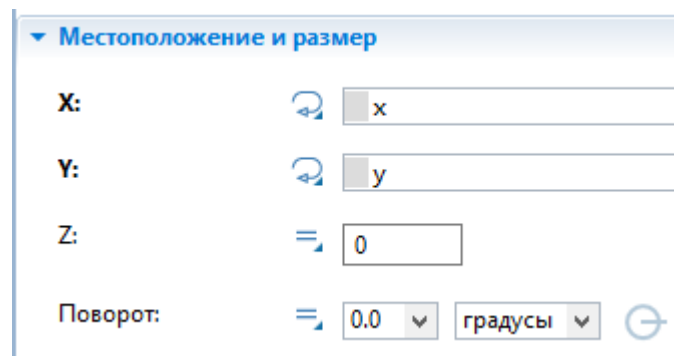
Столкновение с границей происходит, когда центр шара приблизится к ней на расстояние радиуса  $r$  и шар движется в направлении этой границы. При наступлении события столкновения с границей, изменяется знак соответствующей составляющей скорости шара, и шар отскакивает от нее.

В свойствах события *pushOffX* в пункте *Тип события* укажем при выполнении условия. В окне *Условие*, укажем:  $x \leq r \ \&\& \ v_x < 0 \ || \ x \geq X_{max} - r \ \&\& \ v_x > 0$ . А в пункте *Действие*:  $v_x = -v_x$ ; *pushOffX.restart()*;


Свойства события *pushOffY* заполняются аналогично по примеру выше.


Свойства события *pushOffBall* заполним позже, после того, как добавим функцию, возвращающую указатель на шар, с которым столкнулся данный шарик.


Добавим элемент *group* агенту *Ball* – группу шаров. Привяжем месторасположение координат  $x$  и  $y$ , зададим их как динамические величины.



▼ Местоположение и размер

X: 

Y: 

Z: 



Поворот:    

Рис. 7. Свойства элемента *group*

Переходим на корневой агент *Main*, добавляем элемент *area* – прямоугольник, в котором будут сталкиваться шарики. Добавим, также, популяцию агентов *balls*, и начальное количество агентов примем равным 10. Добавить популяцию агентов возможно перетаскиванием агента *Ball* на рабочую область корневого агента *Main*. Изменить *одиночного агента* на *популяцию* возможно при редактировании его свойств. Заполним свойства

*balls*: где  $X_{max} = area.getWidth()$  и  $Y_{max} = area.getHeight()$ , остальные свойства интуитивно понятны.

Условие столкновения данного шара с другим шаром проверяется функцией *checkHit()*, которая возвращает указатель на тот шар, с которым столкнулся данный шар.

Если столкновения не было, функция возвратит пустой указатель *null*. Если столкновение произошло, то скорости  $v_x$  и  $v_y$  данного шара и того шара, с которым произошло столкновение, пересчитываются в соответствии с рисунком 3.

Теперь можно заполнить свойства события *pushOffBall*:

```
Ball b = main.checkHit( this );
double rx = b.x - x;
double ry = b.y - y;
double vrx = vx - b.vx;
double vry = vy - b.vy;
double k = (vrx*rx+vry*ry) / (rx*rx+ry*ry);
double vnx = rx*k;
double vny = ry*k;
double vtx = vx - vnx;
double vty = vy - vny;
vx -= vnx;
vy -= vny;
b.vx += vnx;
b.vy += vny;
pushOffBall.restart();
```

Последним пунктом необходимо указать действие агента при запуске, т.е. операции, которые необходимо выполнить при порождении каждого экземпляра шара.

```
do
{
```

```

        x = uniform(r, Xmax-r );
        y = uniform( r, Ymax-r );
    }
    while (main.checkHit( this ) != null);
    double alpha = uniform(2*PI);
    vx = v*cos(alpha);
    vy = v*sin(alpha);

```

Это задание случайного направления движения любого порождаемого шара (новый шар будет двигаться со скоростью  $v$  в случайно выбранном направлении), а также случайные значения координат его центра (распределенные равномерно в заданном пространстве так, чтобы не попасть на уже имеющиеся в этом поле шары).

### 4.3. Изменение числа шаров

Для удаления случайного шара (элемента популяции *Balls*) из модели, во время ее работы, следует использовать оператор:

```
if( balls.size() > 0 ) remove_balls( balls.random() )
```

который сначала проверяет, есть ли в модели хотя бы один шар, с помощью метода *size()*, а затем, для удаления случайного шара используется метод *remove\_balls()*, которому в качестве параметра передается ссылка на один из реплицированных объектов, выбранных случайным образом *balls.random()*.

Для обращения к конкретному объекту, например 3 следует использовать метод *get()*, в котором указать номер объекта (номера объектов в коллекции начинаются с 0). То есть, для того, чтобы удалить шар под №3 нужно выполнить оператор:

```
remove_balls(balls.get(3)).
```

Для создания в модели нового шара (добавления в коллекцию реплицированных объектов *Balls* дополнительного элемента класса *Ball*), во время ее работы, используется метод *add\_balls()*, без параметров.

Можно создавать экземпляры объектов и сразу же задавать значения параметров этих объектов путем вызова следующего метода:

*add\_activeObjectName(parameter1, parameter2, ...)* , где *activeObjectName* – имя реплицированного объекта (в нашем случае *balls*), а параметры перечислены в том порядке, в котором они показаны на рис. 8.

Рис. 8. Параметры реплицированного объекта *Balls*

Эти операторы записаны в поле *Действие* панелей свойств кнопок *Add ball* (добавить шар) и *Remove ball* (удалить шар). При каждом нажатии на соответствующую кнопку во время работы модели будет создан или удален случайный шар.

Обратите внимание, что эти методы создаются в классе *Main*, так что они могут быть вызваны напрямую из любого места класса *Main* (например, из его *Действия при запуске* или из *Действия события* и т.д.).

При необходимости создания или удаления объекта из другого объекта, нужно вначале получить ссылку на объект *Main* с помощью метода *get\_Main()*.

Например, если один шар порождает другой, то следует написать такой код в объекте «родителя»:

```
get_Main().add_balls()
```

Другой часто используемый случай: элемент реплицированного объекта (шар) должен уничтожить сам себя:

```
get_Main().remove_balls( this ), здесь параметр this – это ссылка на самого себя.
```

#### 4.4. Изменение цвета шаров

В класс активного объекта *Ball* зададим еще одну переменную *color* типа *Color*, с помощью нее будем явно задавать значения цветовых компонент цвета (Красный, Зеленый и Синий), (значения задаются в диапазоне от 0 до 255).

Чтобы программно задать произвольный цвет для любого графического объекта, следует вызвать конструктор, например, для создания розовато-лилового цвета: *new Color(255, 127, 255)*.

Для окрашивания шаров в различные случайные цвета, будем использовать функцию *uniform\_discr(255)*, принимающую случайное целое значение, распределенное равномерно между 0 и 255:

```
new Color(uniform_discr(255), uniform_discr(255), uniform_discr(255)).
```

Каждый раз при обращении к переменной *color* конструктор *new Color* будет порождать значение случайного цвета.

## 5. ПОЯСНЕНИЯ К ЗАДАНИЯМ

- 1) Обратите внимание на то, что в исходной модели, в функции *checkHit()*, определяющей столкнутся шары или нет, предполагается, что все шары имеют одинаковый радиус *r*. Следует видоизменить эту функцию таким образом, чтобы при вычислении возможности столкновения шаров учитывалось, что они могут иметь разный размер (реализовать данную

задачу в том случае, если учитывается изменение размера шаров в индивидуальном задании по варианту).

- 2) В вариантах, в которых требуется моделировать поведение шаров двух типов, например, синих и красных, не следует создавать 2 класса активных объектов (для каждого типа шаров). Вместо этого целесообразно объявить *параметр*, определяющий тип шара и в зависимости от этого параметра задавать внешний вид и поведение шаров.
- 3) Узнать номер реплицированного элемента из популяции (номер шара, под которым он стоит) можно с помощью метода *getIndex()*.

## 6. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Доработайте полученную ранее модель, добавив в нее:

- реализацию функции *checkHit()* в *Main*;
- анимацию модели с возможностью динамического управления ускорением свободного падения и возможностью добавлять и удалять шары во время выполнения модели;
- реализацию индивидуального задания по варианту.

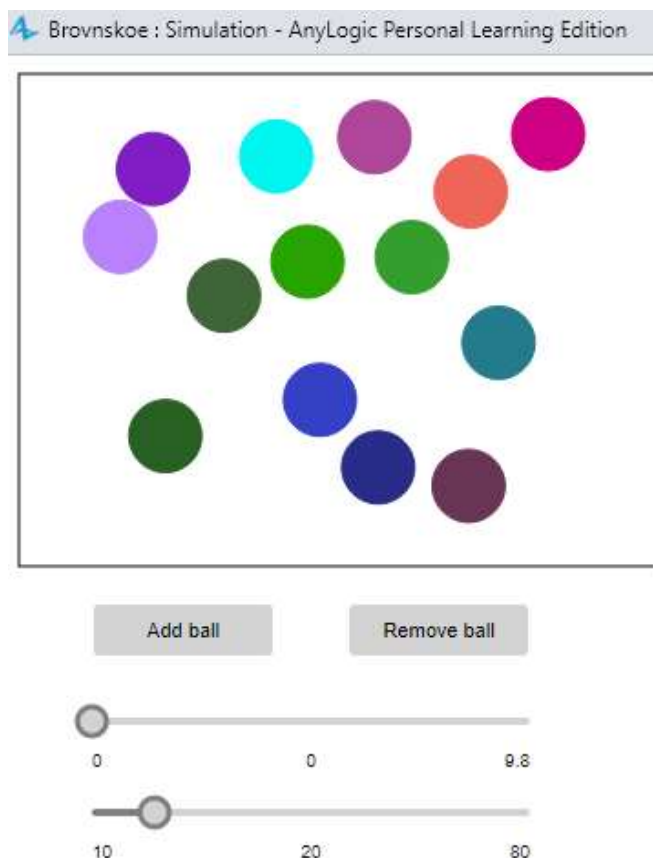


Рис. 9. Анимационное окно модели

Таблица 1

**Индивидуальные задания к выполнению лабораторной работы**

Номер варианта по списку	Перечень заданий
1.	<ul style="list-style-type: none"> <li>Рабочая зона должна быть поделена на 2 части, соответственно верхнюю и нижнюю.</li> <li>В верхней зоне может одновременно находиться не более <math>N</math> шаров. Значение <math>N</math> должно регулироваться бегунком в диапазоне от 0 до 10.</li> <li>Шары могут свободно покидать верхнюю зону, но войти в нее могут, только если в этой зоне число шаров меньше <math>N</math>.</li> <li>В анимации модели отразить количество шаров в верхней зоне.</li> </ul>
2.	<ul style="list-style-type: none"> <li>При соударениях шаров друг с другом должен изменяться их радиус <math>r</math> по следующему алгоритму: радиус того шара, у которого скорость после соударения будет больше, должен</li> </ul>

	<p>увеличиться в 2 раза, а радиус шара с меньшей скоростью должен уменьшиться вдвое.</p> <ul style="list-style-type: none"> <li>• При увеличении радиуса шара до 40 единиц, шар должен исчезать.</li> <li>• При уменьшении радиуса шара до 2,5 единиц – шар должен превратиться в 2 шара нормальной величины (<math>r = 15</math> единиц).</li> <li>• Шары должны быть пронумерованы, номера шаров не должны изменяться и повторяться. В анимации номер должен отображаться рядом с шаром.</li> </ul>
3.	<ul style="list-style-type: none"> <li>• Рабочая зона должна быть поделена на 2 части, соответственно левую и правую.</li> <li>• Шары должны иметь свои уникальные номера, видимые на презентации.</li> <li>• Шары с четным номером могут пересекать разделительную линию только в направлении слева направо.</li> <li>• Шары с нечетным номером могут пересекать разделительную линию только в направлении справа налево.</li> <li>• Кнопка «добавить шар» должна создавать шары с уникальным номером.</li> <li>• В анимации модели отразить количество шаров в левой и правой частях по отдельности.</li> </ul>
4.	<ul style="list-style-type: none"> <li>• Рабочая зона должна быть поделена на 2 части, соответственно левую и правую.</li> <li>• При прохождении разделительной линии слева направо шары исчезают.</li> <li>• При прохождении разделительной линии справа налево появляется 3 новых шара.</li> <li>• В анимации модели отразить общее число шаров и количество шаров, прошедших разделительную линию в том или ином направлении по отдельности.</li> </ul>
5.	<ul style="list-style-type: none"> <li>• После каждого соударения со стенками должен изменяться цвет шара по следующему правилу:</li> </ul>



	<p>а) При соударении с вертикальными стенками каждая компонента <math>RGB</math>-цвета увеличивается на <math>N</math> единиц. По достижении величины 255 единиц, компонента больше не увеличивается.</p> <p>б) При соударении с горизонтальными стенками каждая компонента <math>RGB</math>-цвета уменьшается на <math>N</math> единиц. По достижении величины 0 единиц, компонента больше не уменьшается.</p> <ul style="list-style-type: none"> <li>• Шар, получивший черный цвет исчезает.</li> <li>• Шар, получивший белый цвет порождает новый шар красного цвета, а сам меняет цвет на синий.</li> <li>• В анимации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент <math>R</math>, <math>G</math>, и <math>B</math>.</li> </ul>
6.	<ul style="list-style-type: none"> <li>• Шары должны изменять свой цвет по следующему правилу: <ul style="list-style-type: none"> <li>а) Величина каждой компоненты <math>RGB</math> уменьшается на <math>N</math> единиц в секунду.</li> <li>б) При достижении нуля компонента <math>RGB</math> более не уменьшается.</li> </ul> </li> <li>• Шары черного цвета исчезают. При исчезновении шара появляются 2 шара случайного цвета и со случайным значением скорости. Один шар появляется в правом верхнем углу, другой – в левом верхнем.</li> <li>• В анимации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент <math>R</math>, <math>G</math>, и <math>B</math>.</li> </ul>
7.	<ul style="list-style-type: none"> <li>• Каждый шар должен иметь свой уникальный, неизменный номер, видимый в анимации.</li> <li>• При столкновении шаров одинаковой четности (четный – четный или нечетный – нечетный) должен появиться еще один шар в левом нижнем углу рабочего пространства.</li> <li>• При столкновении шаров разной четности (четный – нечетный) должен исчезнуть тот шар, сумма <math>RGB</math>-компонент цвета, у которого больше.</li> </ul>

	<ul style="list-style-type: none"> <li>В анимации модели отразить общее количество шаров. Рядом с каждым шаром сделать надпись, указывающую на значения компонент <math>R</math>, <math>G</math>, и <math>B</math>.</li> </ul>
8.	<ul style="list-style-type: none"> <li>При соударениях шаров друг с другом должен изменяться их радиус <math>r</math> по следующему алгоритму: радиус того шара, который в момент соударения находится выше, должен увеличиться на 50%, а радиус нижнего шара должен уменьшиться на 50%.</li> <li>При увеличении радиуса шара до 40 единиц, шар должен исчезать.</li> <li>При уменьшении радиуса шара до 4 единиц – шар должен превратиться в 2 шара, радиусом 15, при этом новый шар должен возникнуть в правом верхнем углу.</li> <li>Рядом с каждым шаром должна быть надпись, указывающая на значения компонент <math>R</math>, <math>G</math>, и <math>B</math>.</li> </ul>
9.	<p>Требуется доработать модель таким образом, чтобы при ударе о левую стенку шар раздваивался (т.е. создавалась его копия), при этом скорость нового шара должна быть такая же, как у старого по абсолютной величине и направлению. Начальная координата нового шара должна быть в <i>центре</i> рабочего пространства. При ударе о правую стенку шары должны исчезать. Модель должна показывать текущее количество шаров и абсолютную скорость каждого шара.</p>
10.	<ul style="list-style-type: none"> <li>В модели 2 вида шаров – синие и желтые. При столкновении шаров разного цвета шар с большей скоростью исчезает.</li> <li>Шар с меньшей скоростью порождает появление 2-х шаров. Скорости вновь созданных шаров = 0. Координаты – случайные. Цвет – как у прародителя.</li> <li>В анимации модели показывать, сколько столкновений перенес каждый шар с шаром другого цвета.</li> </ul>
11.	<ul style="list-style-type: none"> <li>В модели 2 вида шаров – голубые и пурпурные. При столкновении шаров разного цвета, исчезает шар с меньшей скоростью.</li> </ul>

	<ul style="list-style-type: none"> <li>• Время жизни шаров 15 сек. По истечении срока жизни шары исчезают. Срок жизни шаров обнуляется при столкновении с шаром другого цвета.</li> <li>• При столкновении шаров одного цвета создается новый шар.</li> <li>• Этот шар возникает того же цвета, со случайными координатами и направлением движения. Скорость нового шара равна сумме скоростей столкнувшихся шаров.</li> <li>• В анимации модели показывать время жизни каждого шара.</li> </ul>
12.	<ul style="list-style-type: none"> <li>• При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.</li> <li>• При столкновениях шаров одинакового цвета исчезает тот из них, скорость которого больше.</li> <li>• При столкновениях шаров разных цветов появляется шар случайного основного цвета в центре рабочей области.</li> <li>• В анимации модели отразить количество красных, зеленых и синих шаров по отдельности.</li> </ul>
13.	<ul style="list-style-type: none"> <li>• Рабочая зона должна быть поделена на 2 части, соответственно верхнюю и нижнюю.</li> <li>• При соударении шаров в верхней зоне исчезает тот шар, сумма <i>RGB</i>-компонент цвета у которого меньше.</li> <li>• При соударении шаров в нижней зоне появляется новый шар в случайном месте, при этом радиус нового шара равен полусумме радиусов столкнувшихся шаров.</li> <li>• В анимации модели отразить общее количество шаров. Рядом с каждым шаром указать его радиус.</li> </ul>
14.	<ul style="list-style-type: none"> <li>• При соударениях шаров друг с другом должен изменяться их радиус <math>r</math> по следующему алгоритму: радиус того шара, у которого скорость после соударения будет меньше должен увеличиться в 2 раза, а радиус шара с большей скоростью должен уменьшиться вдвое.</li> <li>• При увеличении радиуса шара до 60 единиц, шар должен исчезать (лопаться).</li> </ul>

	<ul style="list-style-type: none"> <li>• При уменьшении радиуса шара до 1,5 единиц – шар должен превратиться в 2 шара нормальной величины.</li> </ul>
15.	<ul style="list-style-type: none"> <li>• При старте модели создаются шары только 3-х основных цветов: красный, зеленый и синий.</li> <li>• При столкновениях шаров одинакового цвета появляется шар того же цвета в левом нижнем углу.</li> <li>• При столкновениях шаров разных цветов исчезает тот из них, скорость которого меньше.</li> <li>• В анимации модели отразить количество красных, зеленых и синих шаров по отдельности.</li> </ul>

Формой отчета по данной лабораторной работе является построенная в системе *AnyLogic* модель.

## 7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие классы активных объектов включает проект модели *Броуновское движение шаров*?
2. Каким образом в модели реализован отскок шаров от стен?
3. Как динамически изменять цвет шара в модели?
4. Что такое популяция агентов? Для чего она используется?
5. Какие методы используются для создания/удаления нового агента в модели?

## СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Campbell, S.L. Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4. / Campbell S.L., Chancelier J.P., Nikoukhah R. – 2-е изд. – New York: Springer, 2010. – 329 с.
2. Боев, В.Д. Компьютерное моделирование [Электронный ресурс] / В.Д. Боев, Р.П. Сыпченко. – 2-е изд. – М.: Интернет-Университет

- Информационных Технологий (ИНТУИТ), 2016. – 525 с. – Режим доступа: <http://www.iprbookshop.ru/73655.html>.
3. Введение в математическое моделирование [Электронный ресурс]: учебное пособие / В.Н. Ашихмин [и др.]. – М.: Логос, 2016. – 440 с. – Режим доступа: <http://www.iprbookshop.ru/66414.html>.
  4. Тупик, Н.В. Компьютерное моделирование [Электронный ресурс]: учебное пособие / Н.В. Тупик. – 2-е изд. – Саратов: Вузовское образование, 2019. – 230 с. – Режим доступа: <http://www.iprbookshop.ru/79639.html>.
  5. Осоргин, А.Е. AnyLogic 6. Лабораторный практикум. / А.Е. Осоргин. – Самара: ПГК, 2011. – 100 с.
  6. Карпов, Ю. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 /Ю. Карпов. – СПб.: БХВ-Петербург, 2005. – 400 с.
  7. Лоу, А.М. Имитационное моделирование /А.М.Лоу, В.Д.Кельтон. – 3-е изд. – СПб.: Питер, 2004. –847 с.
  8. Маликов, Р.Ф. Практикум по моделированию сложных систем в среде AnyLogic 6 [Электронный ресурс]: учебное пособие / Р.Ф.Маликов. – Уфа: Издательство БГПУ, 2013. –296 с. – Режим доступа: [https://www.anylogic.ru/upload/Books\\_ru/Практикум\\_по\\_ИМ\\_16-04-14.pdf](https://www.anylogic.ru/upload/Books_ru/Практикум_по_ИМ_16-04-14.pdf)
  9. Каталевский, Д. Ю. Основы имитационного моделирования и системного анализа в управлении [Электронный ресурс]: учебное пособие / Д. Ю. Каталевский. – 2-е изд. – М.: Издательский дом «Дело» РАНХиГС, 2015. – 496 с. – Режим доступа: [https://www.anylogic.ru/upload/pdf/katalevsky\\_osnovy\\_imitatsionnogo\\_modelirovania.pdf](https://www.anylogic.ru/upload/pdf/katalevsky_osnovy_imitatsionnogo_modelirovania.pdf)

Компьютерное моделирование. Построение многоагентных моделей в программном комплексе Anylogic: методические указания к выполнению лабораторной работы № 11 для студентов очной формы обучения по направлению подготовки 09.03.01 «Информатика и вычислительная техника»

ТРУБАКОВА АННА АЛЕКСЕЕВНА  
ТРУБАКОВ АНДРЕЙ ОЛЕГОВИЧ

Научный редактор Д. А. Коростелев  
Компьютерный набор А.А. Трубакова  
Иллюстрации А.А. Трубакова

---

Подписано в печать \_\_.\_\_.\_\_. Усл.печ.л. 1,27 Уч.-изд.л. 1,27

---

Брянский государственный технический университет  
241035, Брянск, бульвар 50 лет Октября, 7 БГТУ  
Кафедра «Информатика и программное обеспечение», тел. 56-09-84

## Сопроводительный лист на издание в авторской редакции

Название работы Компьютерное моделирование. Построение многоагентных моделей в программном комплексе Anylogic: методические указания к выполнению лабораторной работы № 11 для студентов очной формы обучения по направлению подготовки 09.03.01 «Информатика и вычислительная техника»

Актуальность и соответствующий научно-методический уровень подтверждаю \_\_\_\_\_

(подпись научного редактора)

Рукопись сверена и проверена автором \_\_\_\_\_

(подпись автора)

Рекомендуется к изданию \_\_\_\_\_

(подпись заведующего кафедрой)