МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

КОМБИНИРОВАНИЕ АГЕНТНОГО МОДЕЛИРОВАНИЯ С ЧИСЛЕННЫМ РЕШЕНИЕМ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ

Курсовая работа

Пашкевич Ангелины Дмитриевны

студента 3 курса,

специальность 1-31 03 09

Компьютерная математика и системный анализ

Научный руководитель:

кандидат физ.-мат. наук,

доцент О. А. Лаврова

Минск, 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Имитационные модели могут демонстрировать отдельные процессы жизнедеятельности организмов в среде. Посредством агентного моделирования в географии, экологии и социальных науках есть возможность воссоздать указанные процессы.

В данной курсовой работе исследовательского типа рассказывается о построении агентной модели движения стаи птиц, а также усложнении модели посредством построения препятствий.

Курсовая работа носит исследовательский характер. Целью работы является реализация агентной модели для имитации поведения стаи птиц в непрерывной среде с препятствиями.

Основными задачами являются ознакомление с методом численных решений математических моделей и продолжение изучения систем, которые сложно описать традиционными методами моделирования, но к которым применим агентный подход. А именно:

1. изучить реализации агентной модели для имитации движения стаи птиц средствами среды моделирования AnyLogic и средствами Python [2];
2. изучить возможности пакета FEniCS в Python для решения задачи Дирихле для уравнения Лапласа с помощью метода конечных элементов;
3. средствами Python объединить реализацию агентной модели для стаи птиц с численным решением уравнения Лапласа в среде с целью корректировки локального поведения агентов с помощью численно найденного градиентного поля.

ГЛАВА 1
ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Введение в агентное моделирование

В стандартном протоколе ODD для описания IBM (International Business Machines) и ABM, объясняется, какие аспекты модели должны быть описаны в каждом элементе, и приводится пример для иллюстрации указанного протокола. Концепции проектирования включают в себя вопросы о появлении индивидов, взаимодействии между ними, рассмотрении индивидами предсказаний о будущих условиях. Ссылаясь на такие общие концепции проектирования, каждая агентная модель интегрирована в более широкие рамки науки о сложных адаптивных системах.

В современной компьютерной графике и робототехнике применение агентных становится все более частым явлением. К примеру, приложения с виртуальной реальностью: каждый агент взаимодействует и с другими агентами, и со средой, в которой находится, а сами сценарии взаимодействия становятся все более сложными.

Таким образом, подход к моделированию должен хорошо масштабироваться с учетом сложности среды, количества и возможностей взаимодействия агентов, а также с учетом уровня взаимодействия со средой.

В данной курсовой работе разбирается подход к агентному моделированию движения стаи птиц в реальном времени с возможностью взаимодействия со средой.

1.2 Классическая агентная модель имитации движения стаи птиц.

Прекрасная координация полета стаи птиц с незапамятных времен вызывала интерес, как и у обычных людей, так и у ученых. Стаи птиц очень разнообразны во всех контекстах: во время перелетов, избегая хищничества, в стиле полета стаи.

1.2.1 Поведение птиц в стае

Каждая птица изменяет свою схему полета в направлении других птиц в своей окрестности. Таким образом, для птиц существуют локальные правила поведения [3]:

- разделение — избежание столкновений с соседями,
- выравнивание — ориентация на средний курс соседей,
- сплоченность — ориентация на среднюю позицию соседей.

1.2.2 Описание агентной модели

Каждый агент, представляющий птицу, является элементом класса **`myBird`**, представленного с помощью UML-диаграммы классов:

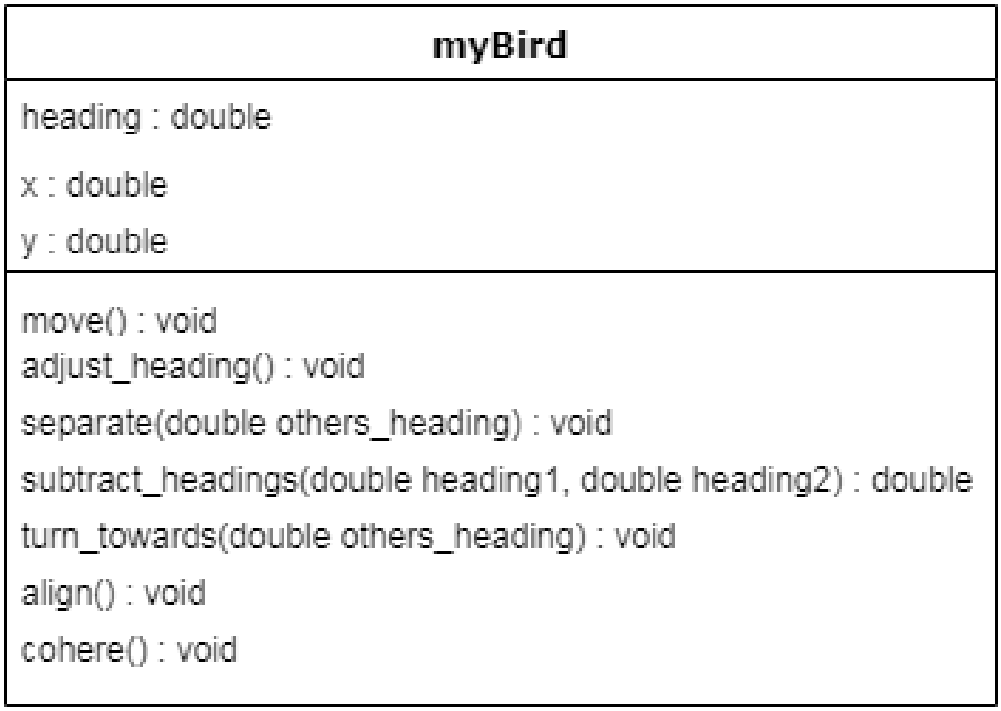

Рис. 1.1 UML-диаграмма класса для агентов

Таким образом, у каждого агента имеются параметры:

- направление — текущее направление в радианах (0...2π),
- координаты — текущее положение в плоскости (x, y) или пространстве (x, y, z).

Также у каждого агента есть методы, удовлетворяющие локальным правилам поведения, для реализации движения в среде.

1. Метод для перемещения птицы в новое положение: **`void move().`** Метод создает для среды тороидальную топологию, рассчитывает новую позицию, изменяет координаты и направление агентов.

2. Метод для изменения направления в соответствии с локальными правилами поведения: **`void adjust_heading().`**Определяет ближайшего агента; если данный агент и ближайший находятся слишком близко, то изменяет направление в противоположную сторону от ближайшего агента, в противном случае выравнивает и согласовывает направление с окружающими агентами.
3. Метод для отведения агента от места сплоченности соседей (близких агентов): **`void separate(double others_heading).`** Определяет разницу в направлении с близким агентом, отворачивает данного агента от направления сплоченных агентов.
4. Метод для определения разницы между направлениями двух агнетов: **`double subtract_heading(double heading1,double heading2).`** Определяет разницумежду направлениями и нормализует между значениями $-\pi$ и π радиан.
5. Метод для разворота агента в сторону направления и положения ближайших агентов: **`void align ().`** Определяет средний курс ближайших агентов и регулирует направление данного агента в вычисленном курсе.
6. Метод для разворота направление агента в сторону положения ближайших соседей: **`void cohere ().`** Определяет среднюю позицию ближайших агентов и регулирует курс данного агента в направлении вычисленного положения.
7. Метод для сонаправления курса данного агента с курсом ближайших агентов: **`void turn_towards (double others_heading).`** Определяет разницу в направлениях и разворачивает агента в сторону направления курса ближайших соседей.

1.3 Задача дирихле для уравнения Лапласа

1.3.1 Постановка задачи

Для заданного точного решения *u(x, y)* построить задачу Дирихле для уравнения Лапласа в области Ω.

$$\begin{cases} \Delta u = 0, \\ u|_{\partial\Omega} = f_1, \\ u|_{\partial\Omega^{(1)} \cup \partial\Omega^{(2)}} = f_2. \end{cases}$$

В данной курсовой работе область Ω соответствует граница среды для агентной модели, $\Omega^{(1)}$ — область первого препятствия в среде, $\Omega^{(2)}$ — область второго препятствия в среде.

Для создания градиентного поля в области задаются граничные условия на краях препятствий.

Теперь необходимо решить построенную задачу средствами библиотеки **FEniCS** и сравнить визуально численное решение, полученное методом конечных элементов, с точным решением *u(x,y)*.

1.3.2 Реализация решения задачи Дирихле для уравнения Лапласа [1]

Построение сетки

Средствами **FEniCS** в Python в области Ω строится сетка, в узлах которой будет искаться решение поставленной задачи Дирихле для уравнения Лапласа.

```
from dolfin import *
import mshr
domain = mshr.Rectangle(Point(0,0), Point(1,1))-
mshr.Circle(Point(300,150), 10000)-mshr.Circle(Point(600,200), 2500)
mesh = mshr.generate_mesh(domain, 40)
plot(mesh)
```

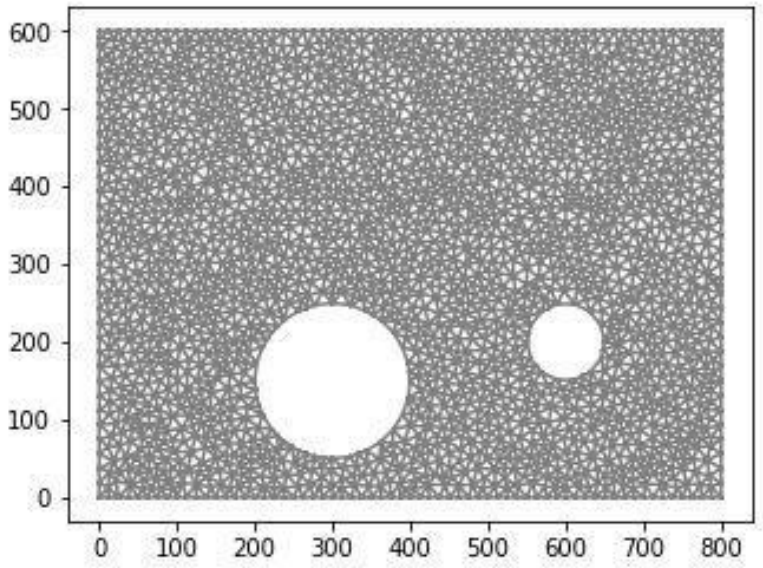

Рис. 1.2 Сетка узлов

Определение граничных условий и задание функции из условия Дирихле

$$\begin{cases} \Delta u = 0, \\ u|_{\partial\Omega} = 0, \\ u|_{\partial\Omega^{(1)} \cup \partial\Omega^{(2)}} = 1. \end{cases}$$

Где, $\partial\Omega^{(1)} = \{(x, y) \in \mathbb{R}^2 : (x - 300)^2 + (y - 150)^2 = 10000\}$,
$\partial\Omega^{(2)} = \{(x, y) \in \mathbb{R}^2 : (x - 600)^2 + (y - 200)^2 = 2500\}$,
$\Omega = [0; 800] \times [0; 600] \subset \mathbb{R}^2$.

Определение границы областей препятствий:

```
tol = 1E-14
center = np.array([[300, 150], [600, 200]])
radius = [10000, 2500]

def boundary(x):
    return abs(x[0]) < tol or abs(x[1]) < tol \
    or abs(x[0] - 1) < tol or abs(x[1] - 1) < tol


def boundary12(x):
  x = np.array(x)
  return np.power(x - center[0], 2).sum() <= radius[0] * radius[0] + tol
\
    or np.power(x - center[1], 2).sum() <= radius[1] * radius[1] + tol
```

Задание функции Дирихле:

```
g_D1 = Expression('0', degree=1)
g_D2 = Constant(1.0)
```

Граничные условия:

```
bc1 = DirichletBC(V1, g_D1, boundary)
bc2 = DirichletBC(V1, g_D2, boundary12)
```

Определение функций правой части:

```
f = Expression('0', degree=1)

a1 = dot(grad(u1), grad(v1))*dx # линейные элементы
L1 = f*v1*dx
```

Решение системы линейных алгебраических уравнений

```
u1 = Function(V1) # переопределение функции u1
bcs = [bc1, bc2]
solve(a1 == L1, u1, bcs)
```

Визуализация решения

Средствами **FEniCS** в Python в области Ω строится численное решение задачи Дирихле для уравнения Лапласа и визуализируется на графике построенное градиентное поле.

```
plot(u1)
```

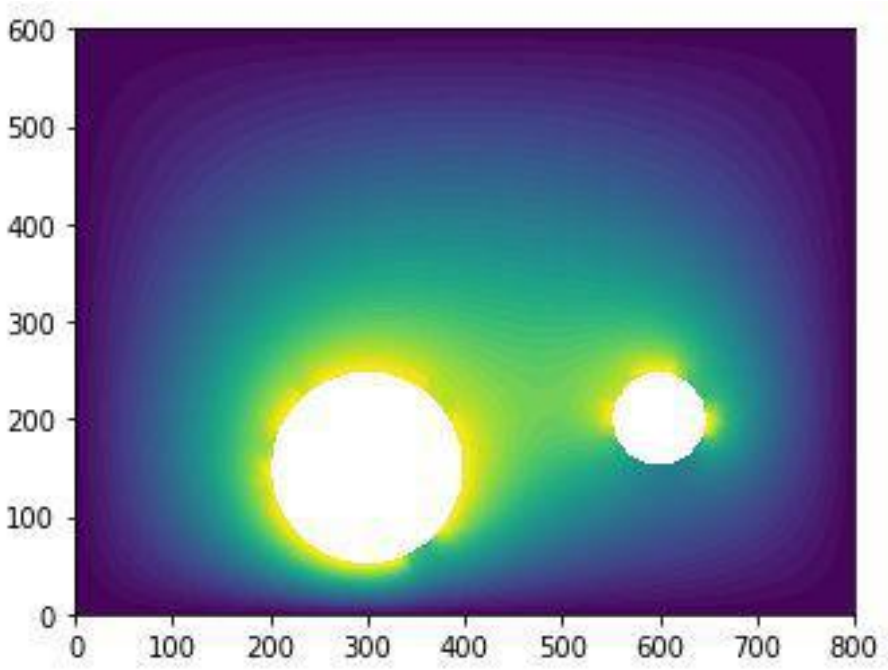

Рис. 1.3 Градиентное поле

ГЛАВА 2
АГЕНТНАЯ МОДЕЛЬ ДВИЖЕНИЯ СТАИ

2.1 Реализация агентной модели движения стаи птиц средствами AnyLogic

2.1.1 Создание среды и агентов

Для реализации агентной модели движения стаи птиц средствами AnyLogic было использовано непрерывное пространство, расположение агентов задавалось при запуске. В процессе работы задавалось популяция

с 10 агентами и размерностью пространства 500×500. Тип сети выбирался согласно расстоянию, радиус соединения 15 (Рис. 2.1).

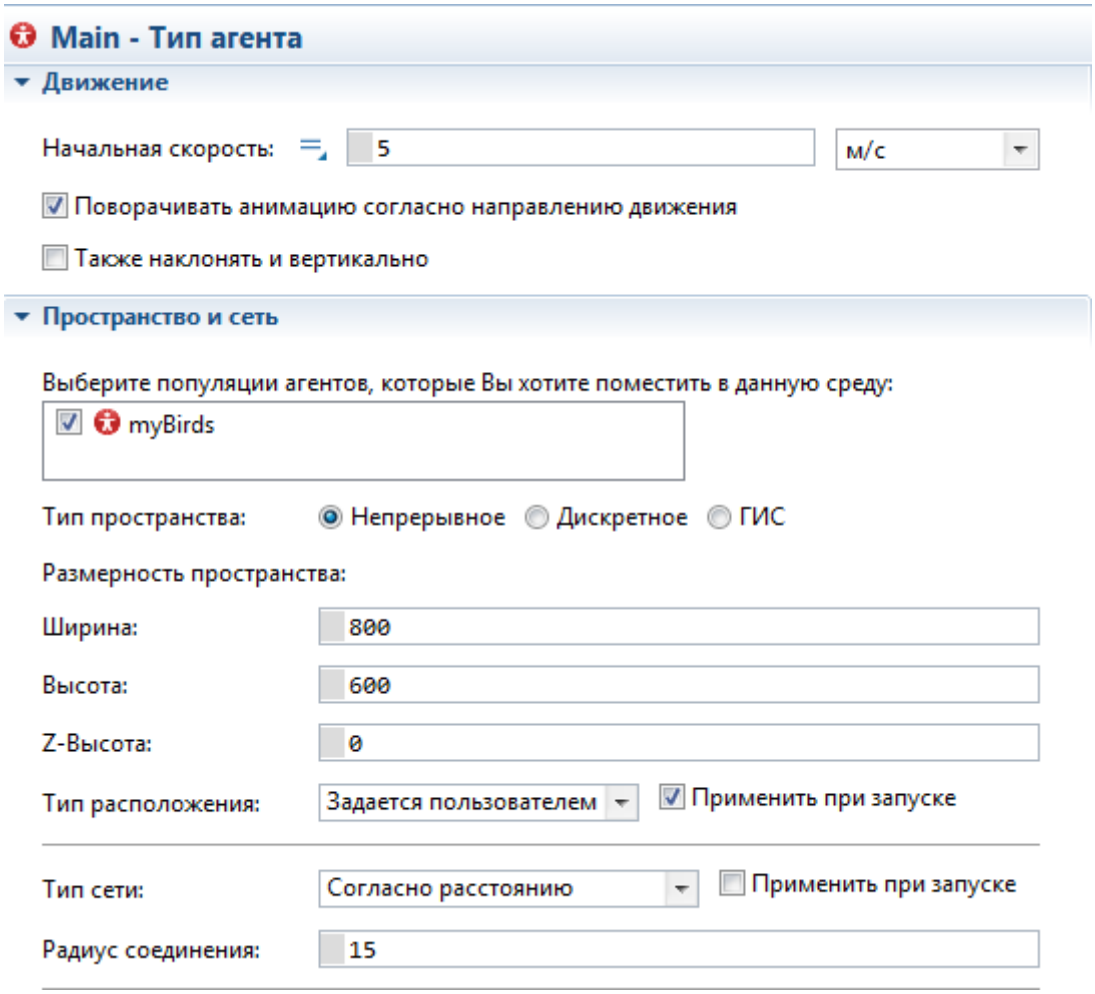

Рис. 2.1 Задание типа пространства, размерности

Далее моделировался процесс движения агентов без препятствий в среде, согласно правилам локального поведения и описанной агентной модели в Главе 1 пункте 1.2.

Создавался тип агентов **`myBirds.`** У агентов задавались действия при запуске и на каждом шаге для задания начального направления и координат агента и их изменения на каждом шаге соответственно.

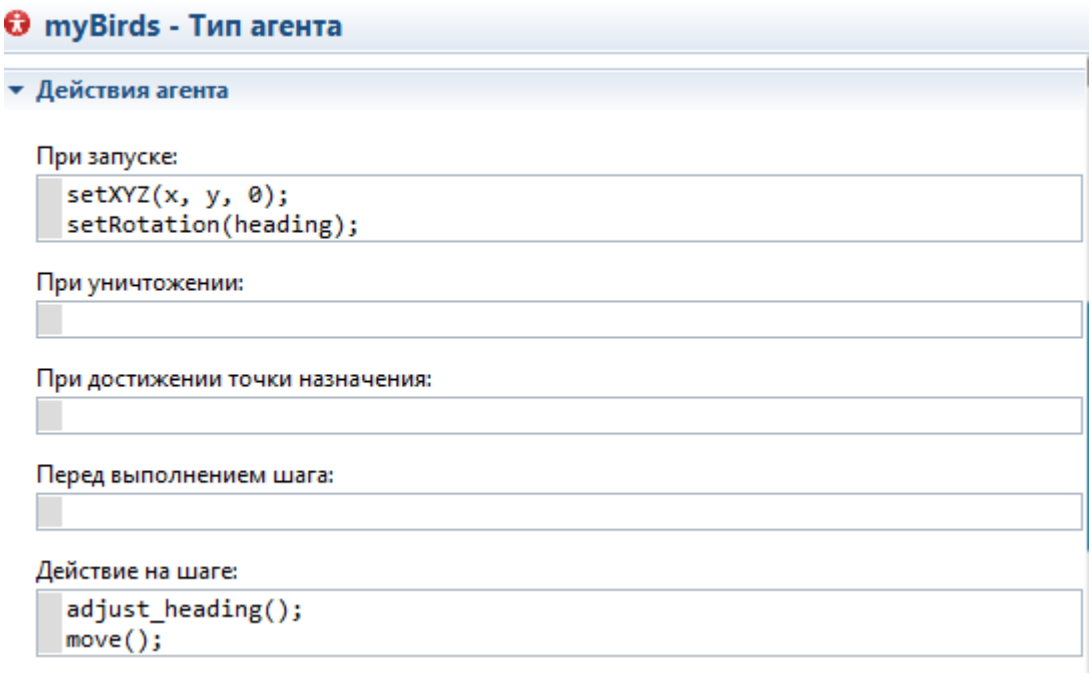

Рис. 2.2 Задание типа агента

Далее задавались необходимые параметры **max_turn_away** и **max_turn** для возможности ограничения угла поворота агента при движении.

Были заданы переменные, соответствующие основным параметрам агентов:

- **heading** — направление движения агента,
- **x** — координата по оси *Ox* в пространстве,
- **y** — координата по оси *Oy* в пространстве.

После были заданы основные функции для реализации движения агентов:

1. Функция для перемещения птицы в новое положение: **void move().**

```
x += getSpeed() * cos(heading);
y += getSpeed() * sin(heading);
if (x > main.spaceWidth()) x -= main.spaceWidth();
else if (x < 0) x += main.spaceWidth();
if (y > main.spaceHeight()) y -= main.spaceHeight();
else if (y < 0) y += main.spaceHeight();
setXYZ(x,                          y,                          0);
setRotation(heading);
```

2. Функция для изменения направления в соответствии с локальными правилами поведения: **void adjust_heading().**

```
if(getConnectionsNumber() == 0) return;
double closest_head = heading;
```

```
double dist = 100, new_dist;
for(Agent a : getConnections()){
      new_dist = distanceTo(a);
      if(new_dist < dist){
            dist = new_dist;
            closest_head = ((myBirds) a).heading;

      }

}
if(dist < crowded){ //short range repulsion
      separate(closest_head);
}else{ //long range attraction
      align();
      cohere();
}
```

3. Функция для отведения агента от места сплоченности ближайших ашентов: **`void separate(double others_heading)`**. Аргументы: `double others_heading` — направление ближайшего агента к данному.

```
double diff = subtract_headings(heading, others_heading);
if(diff == 0) return;
if(diff > 0) heading -= max_turn_away;
else heading += max_turn_away;
if(heading < 0) heading += 2*PI;
if(heading > 2*PI) heading -= 2*PI;
```

4. Функция возвращает разницу **diff** между направлениями двух агнетов: **`double subtract_heading(double heading1,double heading2).`**
   Аргументы: **`double heading1,double heading2`** — направления 1-го и 2-го агентов.

```
double diff = heading2-heading1;
if(diff <= -PI) diff += 2*PI;
if(diff > PI) diff -= 2*PI;
return diff;
```

5. Метод для разворота агента в сторону направления и положения ближайших агентов: **`void align ().`**

```
double avg_head = heading;
for(Agent a : getConnections())
      avg_head += ((myBirds)a).heading;
avg_head /= getConnectionsNumber()+1;
turn_towards(avg_head);
```

6. Функция для разворота направление агента в сторону положения ближайших соседей: **void cohere ().**

```
double avg_pos_x = x, avg_pos_y = y;
for(Agent a : getConnections()){
      avg_pos_x += a.getX();
      avg_pos_y += a.getY();

}
avg_pos_x /= getConnectionsNumber()+1;
avg_pos_y /= getConnectionsNumber()+1;
double pos_head = atan2(y-avg_pos_y,x-avg_pos_x);
turn_towards(pos_head);
```

7. Функция для сонаправления курса данного агента с курсом ближайших агентов: **void turn_towards (double others_heading).**

```
double diff = subtract_headings(heading, others_heading);
if(diff == 0) return;
if(diff > 0) heading += min(diff,max_turn);
else heading += max(diff,-max_turn);
if(heading < 0) heading += 2*PI;
if(heading >= 2*PI) heading -= 2*PI;
```

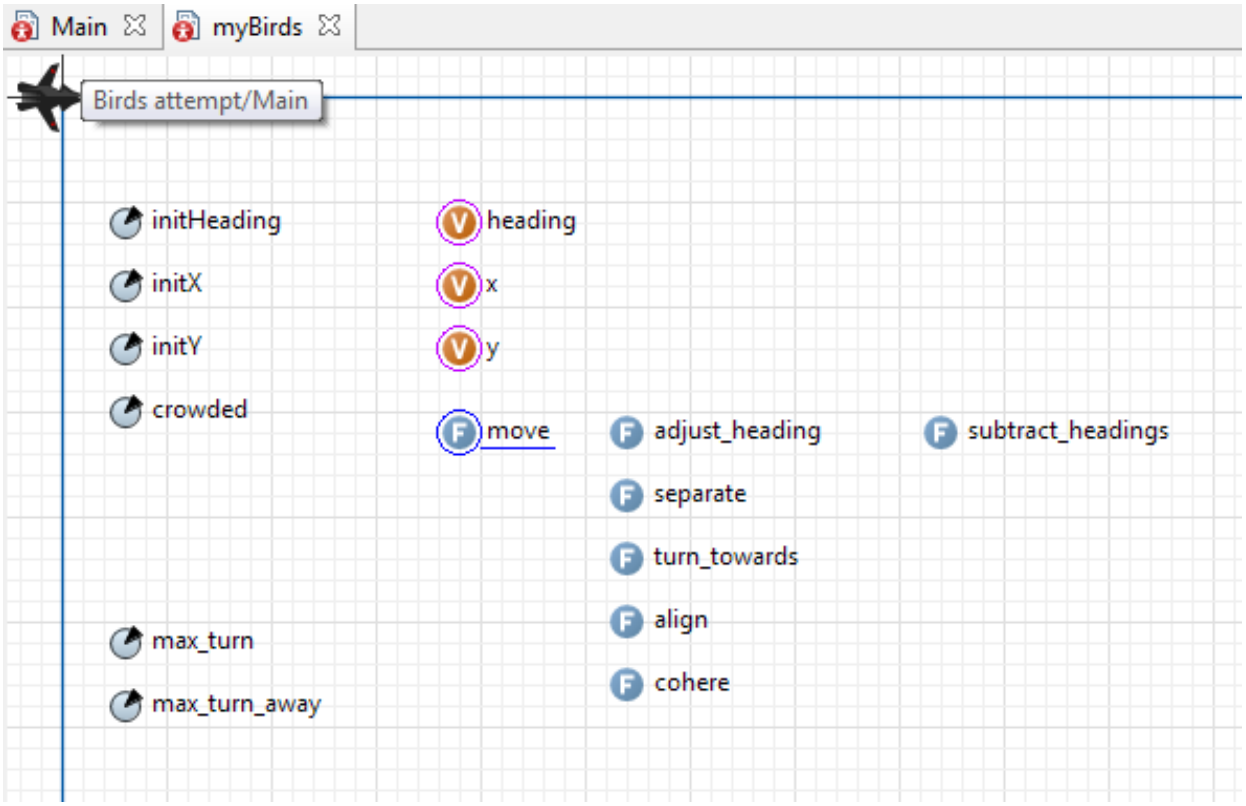

Рис. 2.3 Задание параметров, переменных и функций

2.1.2 Результаты моделирования

Анимация движения агентов

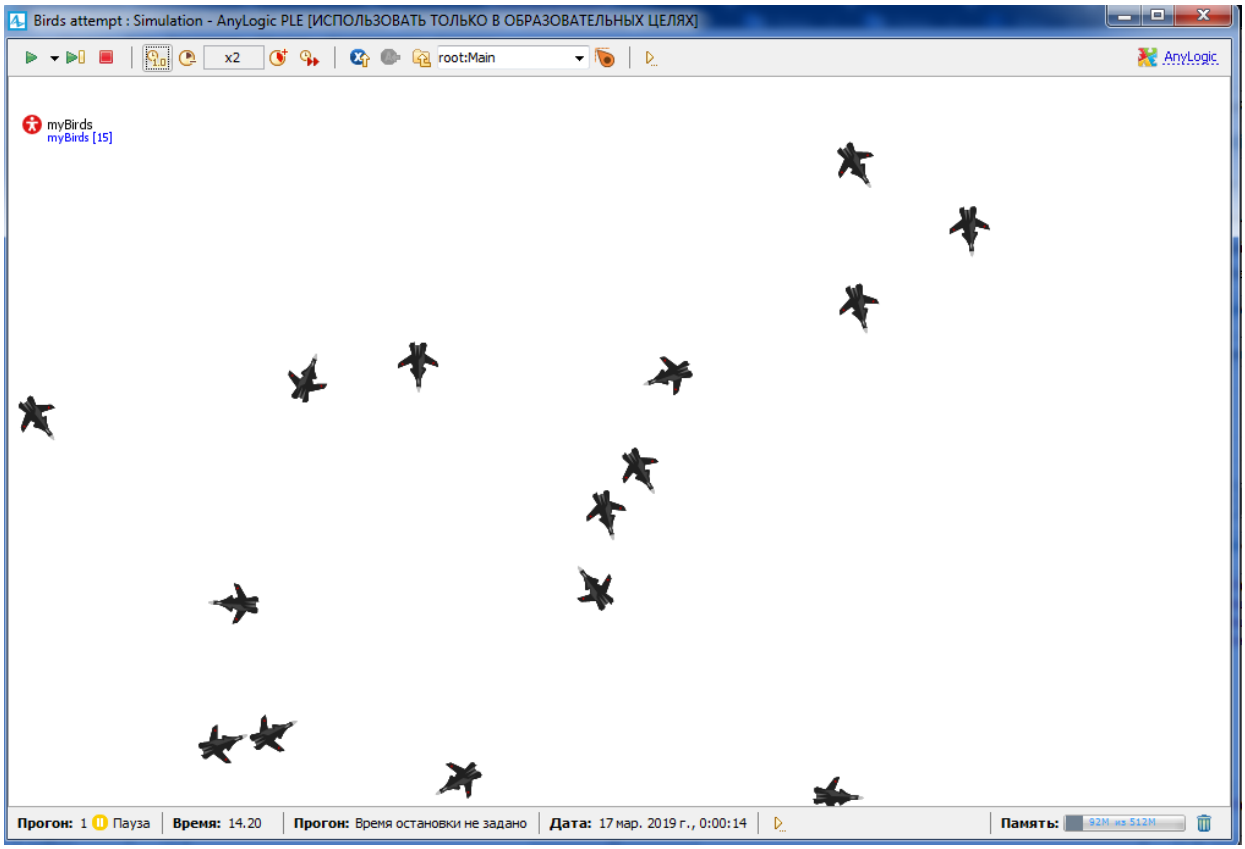

Рис. 2.4 Изображение анимации движения агентов

С помощью двумерной визуализации модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией (то есть агенты, перейдя границу, возвращаются в заданную область).

2.2 Реализация агентной модели движения стаи птиц средствами Python

2.2.1 Описание реализованной модели

Классическая модель движения стаи птиц, описанная в Главе 1 пункте 1.2, была реализована средствами Python в курсовой работе Старовойтовой Виктории[2].

Были реализованы основные функции для реализации движения агентов:

1. Функции для перемещения птицы в новое положение: **`move()`, `move_agent()`.**

```
def move():
```

```
for boid in boids:
 boid.heading = boid.adjust_heading(boids)
for boid in boids:
 boid.move_agent()

def move_agent(self):
 self.position.x += self.velocity * math.cos(self.heading)
 self.position.y += self.velocity * math.sin(self.heading)
 if self.position.x > WIDTH:
  self.position.x -= WIDTH
 else:
  if self.position.x < 0:
   self.position.x += WIDTH
 if self.position.y > HEIGHT:
  self.position.y -= HEIGHT
 else:
  if self.position.y < 0:
   self.position.y += HEIGHT
```

2. Функция для изменения направления в соответствии с локальными правилами поведения: **adjust_heading().**

```
def adjust_heading(self, boids):

 if self.getConnectionsNumber(boids) != 0:
  closest_head = self.heading
  dist = 1000

  for boid in self.getConnections(boids):
   new_dist = (((self.position.x - boid.position.x) ** 2) +
               ((self.position.y - boid.position.y) ** 2)) ** 0.5

   if new_dist < dist:
    dist = new_dist
    closest_head = boid.heading
  if dist < crowded:
   print "separate"

   head = separate(self.heading, closest_head)
  else:
   print "align"
   max_turn = 0.05
   self.heading = self.align(boids)
   max_turn = 0.02
   head = self.cohere(boids)
 else:
  head = self.heading
 return head
```

3. Функция для отведения агента от места сплоченности ближайших агентов: **separate(heading, others_heading)**. Аргументы: heading — направление данного агента; others_heading — направление ближайшего агента к данному.

```
def separate(heading, others_heading):
 diff = subtract_headings(heading, others_heading)
 if diff > 0:
  heading -= max_turn_away
 else:
  heading += max_turn_away
 if heading < 0:
  heading += 2 * math.pi
 if heading > 2 * math.pi:
  heading -= 2 * math.pi
 return heading
```

4. Функция возвращает разницу **diff** между направлениями двух агентов: **subtract_heading(self_heading,boid_heading2).** Аргументы: **self_heading1, boid_heading2** — направления данного агента и соседнего соответсвенно.

```
def subtract_headings(self_heading, boid_heading):
 diff = boid_heading - self_heading
 if diff <= -math.pi:
  diff += 2 * math.pi
 if diff > math.pi:
  diff -= 2 * math.pi
 return diff
```

5. Метод для разворота агента в сторону направления и положения ближайших агентов: **align (self, boids).** Аргументы: **self, boids** — данный агент и агент-сосед соответственно.

```
 def align(self, boids):
  avg_head = self.heading
  for boid in self.getConnections(boids):

   if boid is not self:
    avg_head += boid.heading

  avg_head /= (self.getConnectionsNumber(boids)+1)

  head = turn_towards(self.heading, avg_head,0.05)
  return head
```

6. Функция для разворота направление агента в сторону положения ближайших соседей: **cohere (self, boids).**

Аргументы: **self, boids** — данный агент и агент-сосед соответственно.

```
def cohere(self, boids):
 avg_pos = TwoD(self.position.x,self.position.y)
 print avg_pos
 for boid in self.getConnections(boids):
  print boid.position
  if boid is not self:
   avg_pos += boid.position
 avg_pos /= (self.getConnectionsNumber(boids)+1)

 print avg_pos
 pos_head = math.atan2(self.position.y - avg_pos.y, self.position.x
- avg_pos.x)

 head = turn_towards(self.heading, pos_head,0.05)
 return head
```

7. Функция для сонаправления курса данного агента с курсом ближайших агентов: **turn_towards (heading, boid_heading, max_turn).**

   Аргументы: **heading, boid_heading** — направление данного агента и агента-соседа соответственно; **max_turn** — максимально возможное значение угла поворота.

```
def turn_towards(heading, boid_heading, max_turn):
 print max_turn
 print heading
 print boid_heading
 diff = subtract_headings(boid_heading,heading)
 if diff !=0:
  if diff > 0:
   heading += min(diff, max_turn)
  else:
   heading += max(diff, -max_turn)
  if heading < 0:
   heading += 2 * math.pi
  if heading >= 2 * math.pi:
   heading -= 2 * math.pi
 return heading
```

2.2.2 Результаты моделирования

Анимация движения агентов[2]

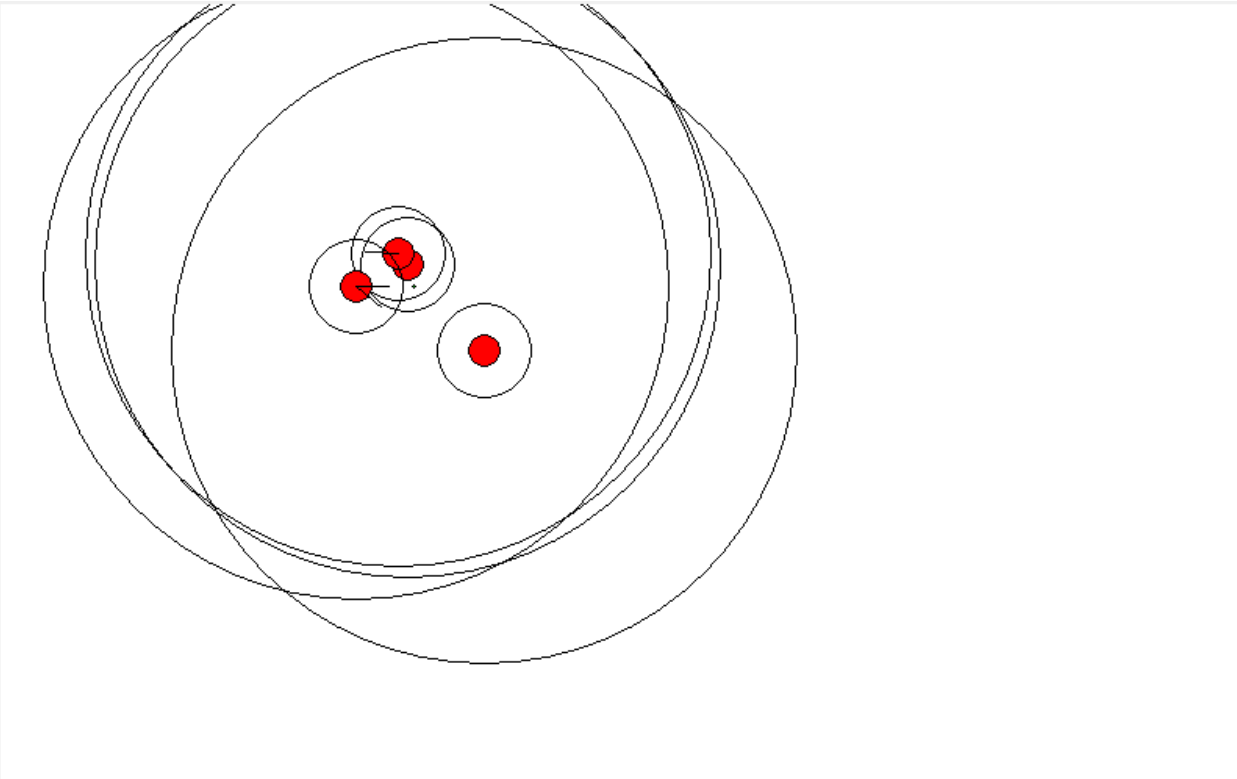

Рис. 2.5 Изображение анимации движения агентов

С помощью двумерной визуализации модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией без учета препятствий.

ГЛАВА 3
АГЕНТНАЯ МОДЕЛЬ ДВИЖЕНИЯ СТАИ ПТИЦ С УЧЕТОМ ПРЕПЯТСТВИЙ

3.1 Описание модели движения стаи птиц с учетом препятствий

Модель движения стаи птиц с учетом препятствий — усложненная классическая модель движения стаи птиц. А именно: в среде создаются препятствия — области, которые агент должен обходить, не соприкасаясь.

Обойти созданные в среде препятствия можно с помощью градиентного поля, построенного посредством решения задачи Дирихле уравнения Лапласа, которое описано в Главе 1 пункте 1.3. Таким образом, агент, направление которого совпадает с увеличением значения градиента в градиентном поле, близок к препятствию и движется в его. Во избежании столкновения — попадания в область препятствия — в дополнение к локальным правилам поведения, описанным в Главе 1 пункте 1.2, были созданы правила предотвращения столкновения с препятствием: выбор наилучшего направления движения.

Таким образом, после выбора направления движения в соответствии с правилами локального поведения, если агент находится вблизи препятствия, то его направление пересчитывается по правилу:

1. находится вектор градиентного поля в точке, в которой находится агент, grad(x, y) и нормируется: $v = (v_x, v_y) := \frac{grad(x,y)}{||grad(x,y)||}$;
2. считаются два касательных направления к препятствию в точке, где расположен данный агент:
   $tangent_1 := (v_y, -v_x)$,
   $tangent_2 := (-v_y, v_x)$;
3. пусть вектор направления, выбранного по правилам локального поведения, $heading$; тогда из двух направлений касательных выбирается то, которое составляет наименьший угол с подсчитаны направлением $heading$:
   if $< tangent_1, \frac{heading}{||heading||} > \geq < tangent_2, \frac{heading}{||heading||} >$, then $tangent := tangent_1$,
   else $tangent := tangent_2$;

4. Во избежании захождения в область препятствия при высокой скорости определим новый вектор направления агента$heading := 0.9tangent - 0.1\frac{grad(x,y)}{||grad(x,y)||}$.

3.2 Реализация агентной модели движения стаи птиц с учетом препятствий в Python

3.2.1 Реализация новой модели

Опишем изменения, внесенные в реализацию агентной модели движения стаи птиц, описанной в Главе 2 пункте 2.2.

1. Функции для определения близко ли расположен агент к препятствию: **is_close_to_boundary(x).** Аргументы: **x** — координаты агенты в области (x, y).

```
def is_close_to_boundary(x):
    dist = np.linalg.norm(x - center[0])
    for c, r in zip(center, radius):
        dist = np.linalg.norm(x - c, 2)
        if dist <= r + 3 * BOID_RADIUS:
            return True
 return False
```

2. Функции для подсчета касательных направлений к области препятствия в точке расположения агента в плоскости **get_tangent(gradu, x, y).** Аргументы: **gradu** — градиентное поле в области; **x, y** — координаты агенты в области (x, y).

```
def get_tangent(gradu, x, y):
    ksi = np.random.rand()
    if ksi <= 0.5:
        flow = np.array([1, -1])
    else:
        flow = np.array([-1, 1])
    tangent = gradu(x, y)
    tangent = tangent / np.linalg.norm(tangent, 2)
    tangent = np.array([tangent[1], tangent[0]])
    return flow * tangent
```

3. Изменение функции **move_agent(self)** для соответствия движения агента правилам локального поведения и правилам предотвращения столкновения с препятствием. Аргументы: **self** — агент.

```
 def move_agent(self):
```

```
        coord = np.array([self.position.x, self.position.y])
        step  =  np.array([self.velocity  *  math.cos(self.heading),
self.velocity * math.sin(self.heading)])
        if is_close_to_boundary(coord) or is_close_to_boundary(coord
+ step):
            tangent     =     get_tangent(gradu,     self.position.x,
self.position.y)
            if (np.arcsin(tangent[1])) >= 0:
                self.heading = np.arccos(tangent[0])
            else:
                self.heading = 2 * math.pi - np.arccos(tangent[0])
        self.position.x += self.velocity * math.cos(self.heading)
        self.position.y += self.velocity * math.sin(self.heading)
        if self.position.x > WIDTH:
            self.position.x -= WIDTH
        else:
            if self.position.x < 0:
                self.position.x += WIDTH
        if self.position.y > HEIGHT:
            self.position.y -= HEIGHT
        else:
            if self.position.y < 0:
                self.position.y += HEIGHT
```

3.2.2 Результаты моделирования

Анимация движения агентов

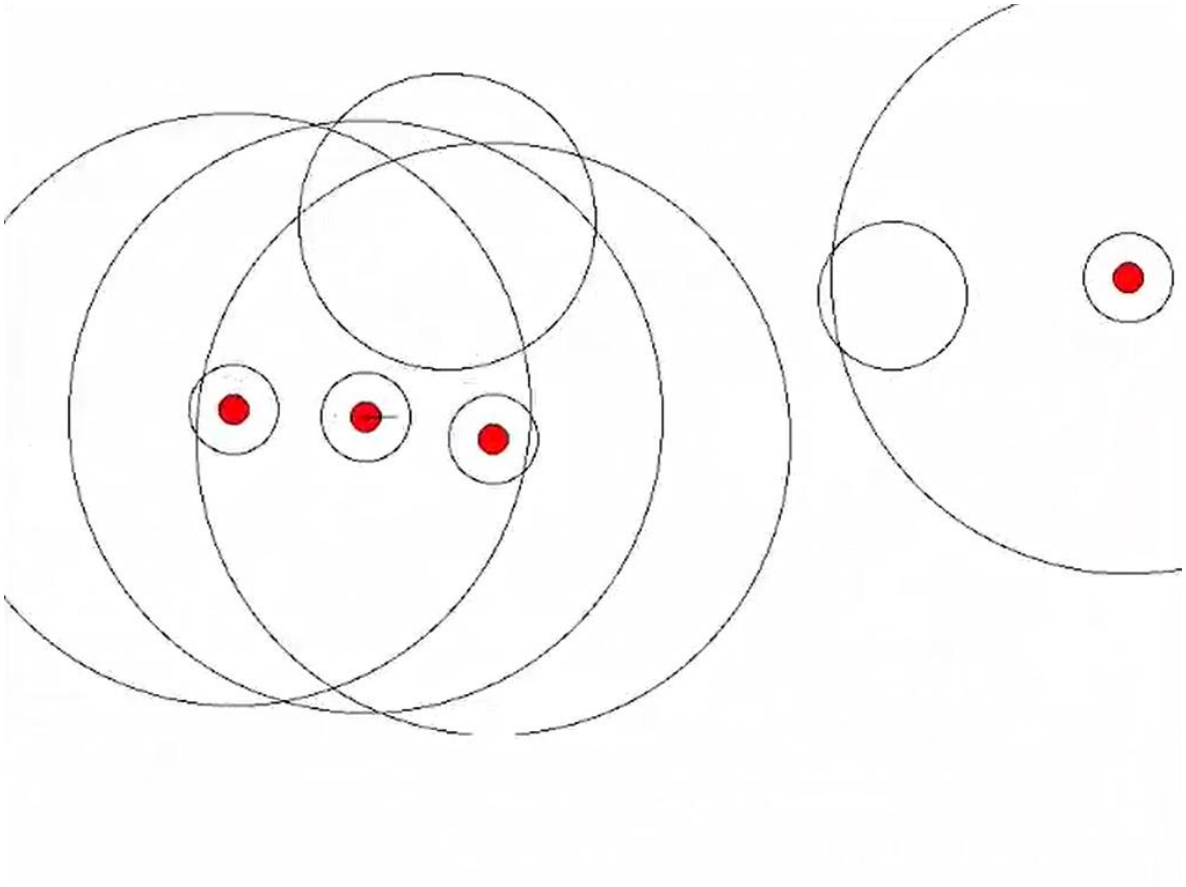

Рис. 3.1 Изображение анимации движения агентов с учетом препятствий

С помощью двумерной визуализации модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией с учетом препятствий. При этом можно заметить, что птицы, уже собравшиеся в стаю, не расходятся, преодолевая препятствие, а остаются в этой же стае вместе.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были сделаны следующие выводы. Во-первых, классическая модель движения стаи птиц может быть построена с помощью агентов и решена математически. Во-вторых, математическую модель движения стаи птиц с учетом препятствий решается построить нельзя. В-третьих, есть возможность использовать имитационное моделирование для построения агентной модели движения стаи птиц с учетом препятствий с помощью глобального управления. А именно был использован подход построения градиентного поля методом конечных элементов (решение задачи Дирихле уравнения Лапласа) в заданной среде агентов для возможности введения правила предотвращения столкновения с препятствием.

В первой главе курсовой работы приводятся сведения о различных понятиях, таких как агентное моделирование, агент, модель движения стаи птиц. В главе описываются агенты как элементы класса, локальные правила их в среде. Также приведена постановка задачи Дирихле для уравнения Лапласа и ее численное решение Данная глава состоит преимущественно из теоретических основ разрабатываемой темы.

Во второй главе содержится описание реализации агентных моделей. Первая — классическая модель движения стаи птиц — реализована средствами Anylogic. Во втором случае предоставлено описание реализации данной модели средствами Python [2]. В данной главе приведены изображения анимаций реализованных агентных моделей в соответствующих программах.

В третьей главе описан алгоритм правила предотвращения столкновения с препятствием для агентной модели движения стаи птиц с учетом препятствий. Также представлены изменения в реализации классической модели в Python, приведено изображение анимации реализованной усовершенствованной агентной модели.

В результате курсового проекта и полученных знаний были достигнуты цели и задачи, которые были поставлены.

В перспективе хотелось бы усовершенствовать и реализовать в агентной модели некоторые расширения модели движения стаи птиц с учетом препятствий: оптимизация процесса обхода препятствий представленных невыпуклыми множествами. Также представляет интерес создание для усовершенствованной модели трехмерной реализации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лаврова О.А., Лабораторная работа. ДВ Метод конечных элементов, ММФ, КМ и СА, БГУ, октябрь 2017.
2. Старовойтова В., Объектно-ориентированная реализация агентных моделей, курсовая работа, ММФ, КМ и СА, БГУ, 2018.
3. Otto von Guericke University, Magdeburg, Agent-Based Modelling and Simulation in AnyLogic. Cells and Birds. 2015
4. AnyLogic – платформа для имитационного моделирования [Электронный ресурс]. – Режим доступа: https://www.anylogic.ru/.
5. Python – высокоуровневый язык программирования общего назначения. – Режим доступа: https://www.python.org/.
6. FEniCS - это исследовательский и программный проект, направленный на создание математических методов и программного обеспечения для автоматизированного вычислительного математического моделирования. – Режим доступа: https://fenicsproject.org .