МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

Пашкевич Ангелина Дмитриевна

АГЕНТНОЕ МОДЕЛИРОВАНИЕ В ДИНАМИЧЕСКИ ИЗМЕНЯЮЩЕЙСЯ СРЕДЕ

Дипломная работа

Научный руководитель:

кандидат физ.-мат. наук,

доцент О. А. Лаврова

Допущена к защите

«___» _____ 2020 г.

Зав. кафедрой дифференциальных уравнений и системного анализа

доктор физ.-мат. наук, профессор В. И. Громак

Минск, 2020

ОГЛАВЛЕНИЕ

РЕФЕРАТ

В дипломной работе 45 страниц, 9 рисунков, 10 источников.

Ключевые слова: АГЕНТНОЕ МОДЕЛИРОВАНИЕ, ЛОКАЛЬНЫЕ ПРАВИЛА ПОВЕДЕНИЯ, ДИНАМИЧЕСКИ ИЗМЕНЯЮЩАЯСЯ СРЕДА, ГРАДИЕНТНОЕ ПОЛЕ.

В дипломной работе строятся и изучаются агентные модели движения стаи птиц с глобальным и локальным контролем динамически изменяющейся среды.

Целью дипломной работы является реализация агентной модели для имитации поведения стаи птиц в динамически изменяющейся среде.

Для достижения поставленной цели использовался язык Python и его математические библиотеки.

В дипломной работе получены следующие результаты:

1. реализована модель движения стаи птиц на основе комбинации агентного подхода и численного решения краевой задачи, где агент обладает локальной и глобальной информацией об изменяющейся среде;
2. реализована модель на основе агентного подхода, где агент обладает только локальной информацией из своей области видимости.

Новизна результатов состоит в добавлении в среду агентной модели препятствий, положение которых изменяется со временем, и агента-лидера.

Дипломная работа носит исследовательский характер. Ее результаты могут быть использованы для дальнейшего изучения механизмов адаптации в мультиагентных системах.

Достоверность полученных результатов обусловлена применением математических методов для решения и анализа задач.

Дипломная работа выполнена автором самостоятельно.

У дыпломнай працы 45 старонак, 9 малюнкаў, 10 крыніц.

Ключавыя словы: АГЕНТНОЕ МАДЭЛЯВАННЕ, ЛАКАЛЬНЫЯ ПРАВІЛЫ ПАВОДЗІН, ДЫНАМІЧНА ЗМЕНЛІВАЕ АСЯРОДДЗЕ, ГРАДЫЕНТНАЕ ПОЛЕ.

У дыпломнай працы будуюцца і вывучаюцца агентныя мадэлі руху чароды птушак з глабальным і лакальным кантролем дынамічна зменлівага асяроддзя.

Мэтай дыпломнай працы з'яўляецца рэалізацыя агентнай мадэлі для імітацыі паводзін чароды птушак у дынамічна зменлівам асяроддзі.

Для дасягнення пастаўленай мэты выкарыстоўвалася мова праграмавання Python і яго матэматычныя бібліятэкі.

У дыпломнай працы атрыманы наступныя вынікі:

1. рэалізавана мадэль руху чароды птушак на аснове камбінацыі агентнага падыходу і колькаснага рашэння краёвай задачы, дзе агент валодае лакальнай і глабальнай інфармацыяй аб зменлівам асяроддзі;
2. рэалізаваная мадэль на аснове агентнага падыходу, дзе агент валодае толькі лакальнай інфармацыяй з сваёй вобласці бачнасці.

Навізна вынікаў складаецца ў даданні ў сераду агентнай мадэлі перашкод, становішча якіх змяняецца з часам, і агента-лідэра.

Дыпломная праца носіць даследчы характар. Яе вынікі могуць быць выкарыстаны для далейшага вывучэння механізмаў адаптацыі ў мульціагентных сістэмах.

Дакладнасць атрыманых вынікаў абумоўлена ўжываннем матэматычных метадаў для вырашэння і аналізу задач.

Дыпломная праца выканана аўтарам самастойна.

The thesis project contains 45 pages, 9 figures, 10 sources.

Key words: AGENT MODELING, LOCAL RULES OF BEHAVIOR, DYNAMICALLY CHANGING ENVIRONMENT, GRADIENT FIELD.

In the thesis project agent models of the movement of a flock of birds are built and studied. This model is with global and local control of a dynamically changing environment.

The purpose of the thesis project is to implement an agent model to simulate the behavior of a flock of birds in a dynamically changing environment.

To achieve this goal Python and its mathematical libraries were used.

In the thesis project the following results were obtained:

1. A model of the movement of a flock of birds based on a combination of the agent approach and the numerical solution of a boundary value problem, where the agent has local and global information about the changing environment, is implemented;
2. A model based on the agent approach has been implemented, where the agent has only local information from its scope.

The novelty of the results consists in adding obstacles to the environment and the agent-leader. The position of obstacles changes with time.

Thesis project is of a research nature. Its results can be used to further study adaptation mechanisms in multiagent systems.

The validity and certainty of the obtained results are dictated by the use of mathematical methods for solving and analyzing problems.

The thesis project is performed by the author independently.

ВВЕДЕНИЕ

Агентное моделирование систем основано на описании локальных правил поведения участников системы (агентов) и последующей эмуляции этих поведений ни микроуровне для получения закономерностей поведения исходной системы на макроуровне. Кроме локальных правил поведения агентов в модели может быть определено глобальное управление с помощью векторного поля для моделирования поведения системы в динамически изменяющейся среде [10].

Посредством имитационных моделей предоставляется возможность описания и визуализации процессов жизнедеятельности простых организмов и более сложных животных в среде. Также имеется возможность построения моделей социальных процессов, в том числе, связанных с действиями агента в обществе.

В данной дипломной работе исследовательского типа рассматривается модель для имитации поведения стаи птиц в области, где динамическая среда создается за счет движения препятствий. Предполагается, что геометрия препятствий имеет круговую форму, чтобы не возникали вычислительные сложности в процессе моделирования. Траектория движения препятствий является произвольной, задание движения будет описано далее в работе.

Целью работы является разработка и реализация агентной модели для

имитации поведения стаи птиц в среде с подвижными препятствиями.

Основными задачами являются продолжение изучения систем, которые сложно описать традиционными методами моделирования, но к которым применим агентный подход, расширение модели движения стаи птиц. А именно:

1. Разработка и реализация комбинированного подхода к моделированию стаи птиц на основе агентной модели и численного решения уравнения Лапласа в среде для построения градиентного поля, с помощью которого осуществляется глобальное управление поведением агентов-птиц для облета препятствий. Данная задача является продолжением результатов курсовой работы [4] для случая динамического изменения положения препятствия в пространстве. Комбинированная модель реализуется средствами **Python** с применением пакета FEniCS для численного решения краевой задачи с помощью метода конечных элементов на каждом временном шаге моделирования.
2. Разработка и реализация агентной модели для имитации поведения стаи птиц с локальным контролем среды каждым из агентов-птиц на наличие препятствия для корректировки локального поведения. Агентная модель реализуется средствами **Python**.
3. Проектирование расширения агентной модели из пункта 2 за счет добавления в модель агента-лидера, ведущего стаю птиц на облет препятствия.

4. Сравнительный анализ двух подходов к моделированию поведения стаи птиц в среде с подвижными препятствиями на основе глобального управления поведением агентов и на основе локального контроля среды моделирования каждым из агентов.

ГЛАВА 1
ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Введение в агентное моделирование

Агентное моделирование — это подход к моделированию, при котором демонстрируется взаимодействие между агентами, друг с другом, и, возможно, средой, описанными в программе. Агенты могут представлять, например, людей (работников организации, участников чрезвычайной ситуации), животных и их группы, простые организмы, клетки, машины и другую технику. Также они могут имитировать объекты, которые выполняют задачи и накапливают знания, эволюционируют, однако у агентов может и не быть как таковой физической основы [7].

В агентном моделировании задаются правила поведения для агентов и также среды, в которой они присутствуют, при необходимости. Далее на основе этих правил строится общее поведение системы. Таким образом, появляется возможность более глубокого понимания и даже контроля работы системы, происходящих в ней событий. Контроль обеспечивается посредством изменения в правилах поведения агентов и в среде.

Агентная модель учитывает не только поведение самого агента, но и его взаимодействие с другими, возможности влияния внешней среды на поведение и решения агента. Это делает ситуации, смоделированные с использованием агентов, более приближенными к реальным [1,2].

В данной дипломной работе разбирается подход к агентному моделированию движения стаи птиц в реальном времени с возможностью

взаимодействия в динамически изменяющейся среде. В среде заданы препятствия, положение которых зависит от времени.

1.2 Классическая агентная модель имитации движения стаи птиц

Во все времена полет стаи птиц завораживал человека и их координация, траектория вызывали интерес у исследователей.

Летая стаями, они почти никогда не сталкиваются друг с другом. Все птицы в ней действуют по конкретным правилам: каждый член стаи следит за своими сородичами и координирует свое направление движения согласно их положению. Это демонстрирует наглядный образец коллективного поведения среди животных.

1.2.1 Поведение птиц в стае

Построим модель движения стаи птиц. Каждая птица — агент, который изменяет свою схему движения в направлении других агентов в зоне своей видимости — некоторой окрестности. Опишем локальные правила поведения для классической модели [6]:

- разделение — избегание столкновений с другими агентами-соседями,
- выравнивание — ориентация на среднее направление агентов-соседей,
- сплоченность — ориентация на среднюю позицию агентов-соседей.

1.2.2 Описание классической агентной модели движения стаи птиц

В классической модели движения стаи птиц агент, представляющий непосредственно птицу, — элемент класса **myBird**, представленного с помощью UML-диаграммы классов:

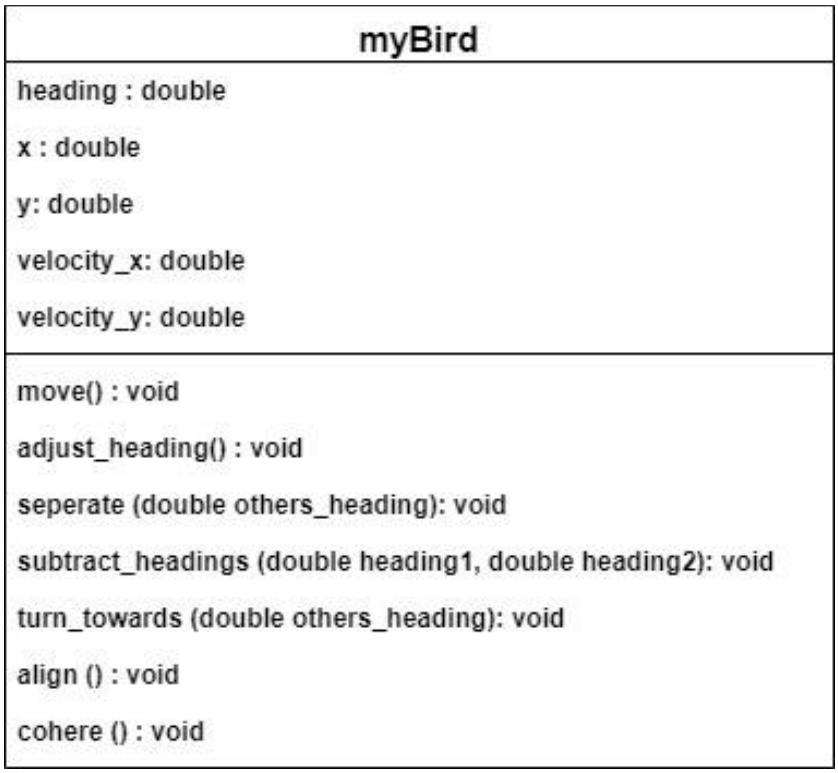

Рис. 1.1 UML-диаграмма класса для агентов-птиц

Параметры агента в классе для классической модели движения стаи птиц:

- направление — текущее направление в радианах (0...2π),
- координаты — текущее положение в плоскости (x, y),
- скорость — координаты вектора скорости в плоскости (velocity_x, velocity_y).

В рамках работы [4] в указанной модели у агентов были реализованы следующие методы, соответствующие 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 разницу между направлениями и нормализует между значениями –π 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 вычисляемом 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 направления 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 [4].

ГЛАВА 2
АГЕНТНАЯ МОДЕЛЬ ДВИЖЕНИЯ СТАИ ПТИЦ С ГЛОБАЛЬНЫМ КОНТРОЛЕМ СРЕДЫ

2.1 Агентная модель в динамически изменяющейся среде

Модель движения стаи птиц в среде с препятствиями — усложнение классической модели движения стаи птиц. В новой модели в среде создаются препятствия — области, столкновение с которыми должны избегать агенты-птицы.

Градиентное поле, которое можно построить, решив задачу Дирихле уравнения Лапласа, которая будет описана в п.2.1.1, поможет обойти созданные в среде препятствия. Таким образом, если у агент-птица находится в точке, где значения градиента в поле увеличивается, то агент близок к препятствию и, возможно, движется в его направлении. В таком случае агенту нужно избежать столкновения, то есть попадания внутрь области препятствия. Для этого локальные правила поведения, описанные в Главе 1 п.1.2, были дополнены новыми правилами предотвращения столкновения с препятствием, которые позволяют агенту выбрать лучшее направления движения без последующего столкновения с препятствием.

Вышесказанное означает, что сначала агент-птица в соответствии с правилами локального поведения (из классической модели) регулирует направление своего движения, а после, если агент-птица расположен вблизи препятствия, его направление пересчитывается по новому правилу предотвращения столкновения (описано в работе [4]):

1. находится вектор градиентного поля в точке, в которой находится агент, $grad(x, y)$ и нормируется: $\vartheta = (\vartheta_x, \vartheta_y) := \frac{grad(x,y)}{||grad(x,y)||}$;
2. считаются два касательных направления к препятствию в точке, где расположен данный агент:

   $tangent_1 := (\vartheta_y, -\vartheta_x)$,

   $tangent_2 := (-\vartheta_y, \vartheta_x)$,
3. пусть вектор направления, выбранного по правилам локального поведения, — $heading$; тогда из двух направлений касательных выбирается то, которое составляет наименьший угол с подсчитаным направлением:

   if $< tangent_1, \frac{heading}{||heading||} > \geq < tangent_2, \frac{heading}{||heading||} >$, then

   $tangent := tangent_1$,

   else $tangent := tangent_2$;
4. В целях избегания захождения в область препятствия при высокой скорости определим новый вектор направления агента $heading := 0.9tangent - 0.1\frac{grad(x,y)}{||grad(x,y)||}$[4].

В новой модели движения стаи птиц с динамически изменяющейся средой (с градиентным полем) агент, представляющий непосредственно птицу, — элемент класса **myBird**, представленного с помощью UML-диаграммы классов:

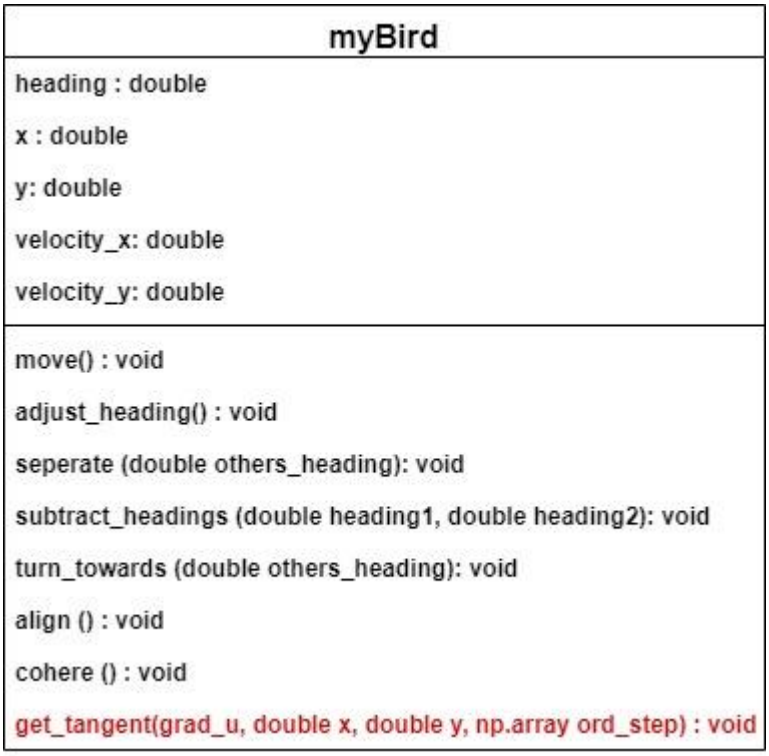

Рис. 2.1 UML-диаграмма класса для агентов-птиц в динамически изменяющейся среде (с градиентным полем)

В агентной модели движения стаи птиц в среде с препятствиями, где, как и в классической модели, заданы агенты-птицы, также задаются агенты-препятствия и правила поведения для них. В рамках дипломной работы динамическая среда создается за счет движения препятствий, то есть их положение зависит от времени. Полагаем, что геометрия препятствий имеет круговую форму, чтобы не возникали вычислительные сложности в процессе моделирования. Траектория движения препятствий является произвольной.

2.1.1 Постановка задачи

В рамках работы [4] препятствия были статическими, задавались градиентным полем, построенным как точное решение *u(x, y)* задачи Дирихле уравнения Лапласа в области Ω.

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

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

В случае с препятствиями, положение которых изменяется со временем, при реализации на каждом шаге будет пересчитываться градиентное поле, то есть для каждого момента t будет решаться задача Дирихле уравнения Лапласа в области Ω:

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

Где области Ω соответствует граница среды для агентной модели, $\Omega^{(1)}(t)$— область первого препятствия в среде, изменяющая свое положение в зависимости от времени t, $\Omega^{(2)}(t)$— область второго препятствия в среде, изменяющая свое положение в зависимости от времени t.

В области Ω для создания градиентного на краях препятствий $\partial\Omega^{(1)}(t), \partial\Omega^{(2)}(t)$ задаются граничные условия.

Построенная задача в каждый момент времени t решается средствами библиотеки FEniCS в **Python**.

2.1.2 Реализация решения задачи Дирихле для уравнения Лапласа. Построение сетки

Решение задачи Дирихле для уравнения Лапласа ищется в узлах сетки, построенной с помощью **FEniCS** в **Python** [3].

Определим препятствия:

```
g_Barriders = [CircleBarrier(400, 390, 200), CircleBarrier(750, 690, 105)]
```

Нарисуем сетку и препятствия:

```
from dolfin import *
import mshr
def get_domain(barriders):
    screen = mshr.Rectangle(dolfin.Point(0, 0), dolfin.Point(1000, 1000))
    for b in barriders:
        screen -= b.get_mesh()
    return screen
def get_mesh(barriders):
    return domain_to_mesh(get_domain(barriders), 30)
def domain_to_mesh(dom, measure = 40):
    return mshr.generate_mesh(dom, measure)

dolfin.plot(get_mesh(g_Barriders))
```

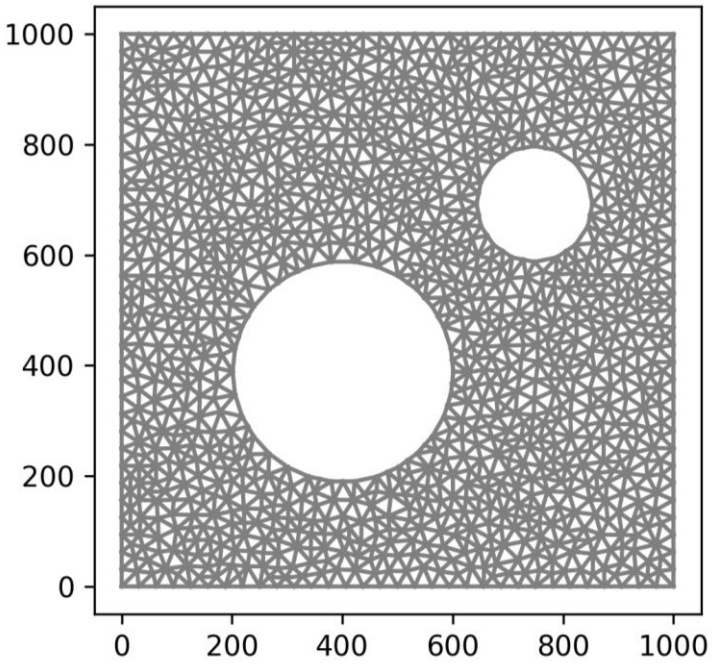

Рис. 2.2 Сетка узлов

Определение граничных условий и задание функции из условия Дирихле:

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

$\partial\Omega^{(i)}(t)$ — границы препятствий, положение которых зависит от времени t.

$\Omega = (0; 1000) \times (0; 1000).$

Определение границы областей препятствий:

```
eps = 1e-14
```

Граница области Ω:

```
def screen_boundary(point):
```

```
    return abs(point[0]) < eps or abs(point[1]) < eps or abs(point[0] -
1000) < eps or abs(point[1] - 1000) < eps
 def boundary_test(self, point, eps):
        return (point[0] - self.x) * (point[0] - self.x) + (point[1] -
self.y) * (point[1] - self.y) <= self.radius * self.radius + eps
```

Граница областей препятствий $\Omega^{(1)}(t), \Omega^{(2)}(t)$:

```
def barriders_boundary(point):
    for b in g_Barriders:
        if b.boundary_test(point, eps):
            return True
    return False
```

Задание функции Дирихле:

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

Граничные условия:

```
bc1 = dolfin.DirichletBC(V1, g_D1, screen_boundary)
bc2 = dolfin.DirichletBC(V1, g_D2, barriders_boundary)
```

Определение функций правой части:

```
poisson_f = dolfin.Expression('0', degree=1)

a1 = dolfin.dot(dolfin.grad(u1), dolfin.grad(v1))*dolfin.dx

L1 = poisson_f*v1*dolfin.dx
```

Решение системы линейных алгебраических уравнений

Переопределение функции u1:

```
u1 = dolfin.TrialFunction(V1)
```

Решение:

```
dolfin.solve(a1 == L1, u1, [dirichlet1, dirichlet2])
```

Визуализация решения

В результате с помощью **FEniCS** в **Python** получается численное решение задачи Дирихле для уравнения Лапласа в области Ω и демонстрируется на графике полученное градиентное поле.

```
plot(u1)
```

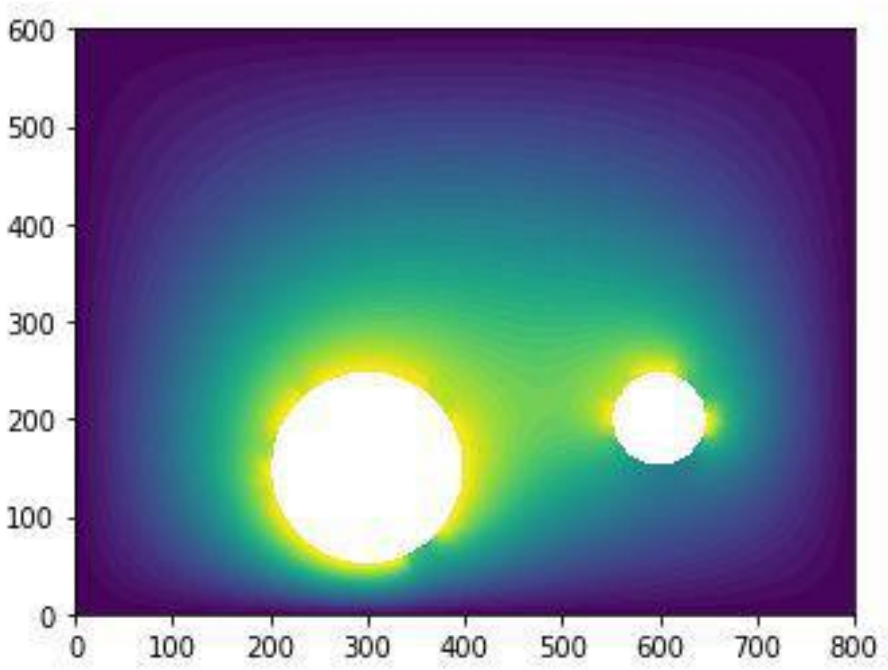

Рис. 2.3 Градиентное поле

2.2 Описание и реализация агентной модели

Классическая модель движения стаи птиц, описанная в Главе 1 пункте 1.2, дополнена динамически изменяемой средой, а именно препятствиями, положение которых изменяется со временем. Ниже будут описаны основные методы, используемые в новой модели.

Были реализованы основные функции для реализации движения агентов-птиц:

1. Функция для перемещения птицы в новое положение: **`move(self, grad_of_pot, close_test).`**

   Аргументы: `self` — ссылка на объект (агента);

   `grad_of_plot` — функция, описывающая градиентное поле в заданной области для задачи Дирихле уравнения Лапласа;

   `close_test` — функция, возвращающая результат (TRUE, координаты препятствия), если агент близок к препятствию или (False, -1, -1) иначе.

```
def move(self, grad_of_pot, close_test):
    step = np.array([self.velocity_x, self.velocity_y])

            if close_test(self.x, self.y) or close_test(self.x +
self.velocity_x, self.y + self.velocity_y):
                    tangent = get_tangent(grad_of_pot, self.x, self.y,
step / np.linalg.norm(step, 2))
                    if (np.arcsin(tangent[1])) >= 0:
          self.set_heading(np.arccos(tangent[0]))
            else:
          self.set_heading(2 * math.pi - np.arccos(tangent[0]))

      self.x += self.velocity_x
      self.y += self.velocity_y

      if (self.x < 0):
            self.x += 1000
      elif (self.x > 1000):
            self.x -= 1000
```

```
        if (self.y < 0):
                self.y += 1000
        elif (self.y > 1000):
                self.y -= 1000
```

2. Функция для вычисления касательного направления в точке градиентного поля: **get_tangent(grad_of_plot, x, y, ort_step)**

   Аргументы: grad_of_plot — функция, описывающая градиентное поле в заданной области для задачи Дирихле уравнения Лапласса;

   x , y — координаты (положения агента-птицы); ort_step — единичный вектор (направления движения агента-птицы).

```
def get_tangent(gradu, x, y, ort_step):
    flow1 = np.array([1, -1])
    flow2 = np.array([-1, 1])
    tangent = gradu(x, y)
    tangent = tangent / np.linalg.norm(tangent, 2)
    tangent = np.array([tangent[1], tangent[0]])
  if ort_step.dot(flow1 * tangent) > ort_step.dot(flow2 * tangent):
            tangent = flow1 * tangent
    else:
            tangent = flow2 * tangent
   tangent = 0.9 * tangent - 0.1 * gradu(x, y) / np.linalg.norm(gradu(x, y), 2)
        return tangent
```

3. Функция для изменения направления, удовлетворяющая правилам локального поведения: **adjust_heading(self, particles).** Аргументы: `self` — ссылка на объект (агента); `particles` — массив агентов-птиц.

```
def adjust_heading(self, particles):
    if self.get_connections_number(particles) != 0:
        closest_head = self.get_heading()
        dist = 1000
        for particle in self.get_connections(particles):
            new_dist = (((self.x - particle.x) ** 2) + ((self.y - particle.y) ** 2)) ** 0.5
            if new_dist < dist:
                dist = new_dist
                closest_head = particle.get_heading()

        if dist < 30:
            head = separate(self.get_heading(), closest_head)
        else:
            self.set_heading(self.align(particles))
            head = self.cohere(particles)
    else:
        head = self.get_heading()

    self.set_heading(head)
```

4. Функция для удаления агента от сплоченности ближайших агентов для избегания столкновений: **separate(heading, others_heading)**.

Аргументы: `heading` — направление данного агента; `others_heading` — направление ближайшего агента к данному.

```
diff = subtract_headings(heading, others_heading)
    if diff > 0:
        heading -= 0.02
    else:
        heading += 0.02
    if heading < 0:
        heading += 2 * math.pi
    if heading > 2 * math.pi:
        heading -= 2 * math.pi
    return heading
```

5. Функция вычисляет разницу **diff** между направлениями агента-птицы и его соседа:

   `subtract_heading(self_heading,boid_heading2).`

   Аргументы: **`self_heading1, boid_heading2`** — направления данного агента и соседнего соответственно.

```
diff = boid_heading - self_heading
    if diff <= -math.pi:
        diff += 2 * math.pi
    if diff > math.pi:
        diff -= 2 * math.pi
    return diff
```

6. Функция для поворота агента в сторону среднего направления ближайших к нему агентов: **`align (self, particles).`**

   Аргументы: **`self, particles`** — данный агент и массив агентов-птиц соответственно.

```
def align(self, particles):
    avg_head = self.get_heading()
    for particle in self.get_connections(particles):
        if particle is not self:
            avg_head += particle.get_heading()
    avg_head /= (self.get_connections_number(particles) + 1)

    return turn_towards(self.get_heading(), avg_head, 0.05)
```

7. Функция для разворота направление агента в сторону среднего положения ближайших соседей: **cohere (self, particles).** Аргументы: **self, particles** — данный агент и массив агентов-птиц соответственно.

```
def cohere(self, particles):
    avg_pos_x = self.x
    avg_pos_y = self.y

    for particle in self.get_connections(particles):
        if particle is not self:
            avg_pos_x += particle.x
            avg_pos_y += particle.y

    n = self.get_connections_number(particles) + 1
    avg_pos_x /= n
    avg_pos_y /= n

    pos_head = math.atan2(self.y - avg_pos_y, self.x - avg_pos_x)

    if pos_head < 0:
```

```
            pos_head += 2 * math.pi

        return turn_towards(self.get_heading(), pos_head, 0.05)
```

8. Функция для направления 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):
    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
```

9. Функция **`simulate_particles(particles, grad_of_plot, close_test)`** полностью реализует алгоритма движения агентов-птиц в среде средствами **Python**, с использованием новых функций, описанных выше.

   Аргументы: `particles` — массив агентов-птиц;

`grad_of_plot` — функция, описывающая градиентное поле в заданной области для задачи Дирихле уравнения Лапласа;

`close_test` — функция, возвращающая результат (TRUE, bx, by), если агент близок к препятствию или (False, -1, -1) иначе.

```
def simulate_particles(particles, grad_of_pot, close_test):
    for p in particles:
        p.adjust_heading(particles)
```

После расчета направлений по правилам локального поведения, учитываем правила предотвращения столкновения с препятствием для каждого агента-птицы и перемещаем их в новую позицию:

```
    for p in particles:
        p.move(grad_of_pot, close_test)
```

Были созданы основные функции для реализации движения препятствий:

1. Функция для сдвига препятствия в новое положение: **`shift(self, dx, dy)`.**

   Аргументы: `self` — ссылка на объект (препятствие); `dx, dy` — длина сдвига по оси Ox и Oy соответственно.

```
def shift(self, dx, dy):
        self.x += dx
        self.y += dy
```

2. Функция для вычисления центра масс препятствия: **`get_center_of_mass(self)`.**

   Аргументы: `self` — ссылка на объект (препятствие).

```
def get_center_of_mass(self):
```

```
        return (self.x, self.y)
```

3. Функции, реализующие движение препятствий:

   Аргументы: `barriers` — массив препятствий; `f` — функция, задающая векторное поле поворота

   Зададим функцию сдвига (один шаг):

```
def browian_shifting(barriders, max_measure_of_screen = 0.05):
    delta = max_measure_of_screen * 1000
    for b in barriers:
            b.shift(random.uniform(-delta,+delta),random.uniform(-
      delta, +delta))
```

   Зададим поле скоростей для смещения препятствий:

```
def moving_in_vector_field(barriers, f, coef = 1):
    for b in barriers:
        vx, vy = f(b.get_center_of_mass())
        vx *= coef
        vy *= coef
        b.shift(vx, vy)
```

2.3 Результаты моделирования

Анимация движения агентов в динамически изменяющейся среде, заданной градиентным полем

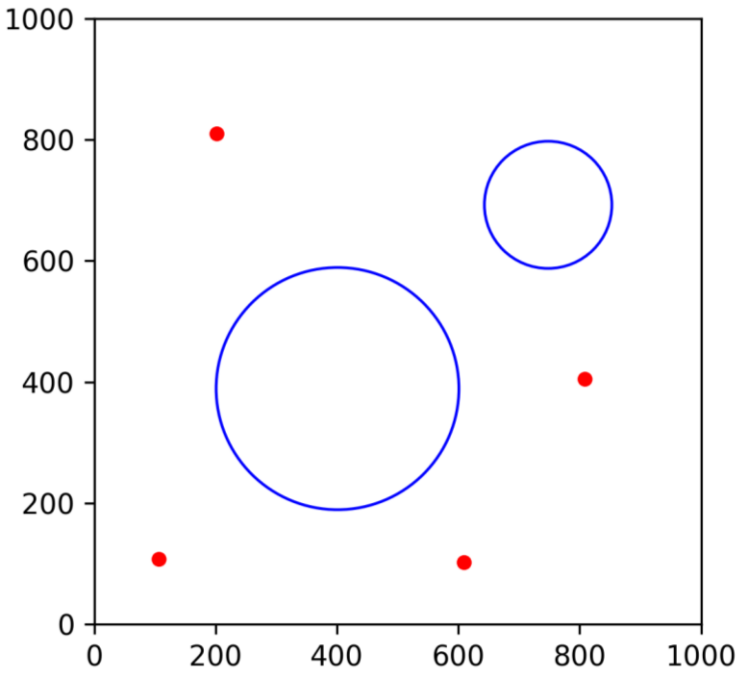

Рис. 2.4 Изображение анимации движения агентов в динамически изменяющейся среде (с градиентным полем)

Посредством визуализации (двумерной) модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией с учетом обхода препятствий, изменяющих свое положение со временем.

ГЛАВА 3
АГЕНТНАЯ МОДЕЛЬ ДВИЖЕНИЯ СТАИ ПТИЦ С ЛОКАЛЬНЫМ КОНТРОЛЕМ СРЕДЫ

3.1 Описание агентной модели

В Главе 3 описывается модель стаи птиц, в которой контроль среды будет производиться на локальном уровне. Геометрия областей препятствий будет также иметь круговую форму, чтобы не возникали вычислительные сложности в процессе моделирования. Траектория движения препятствий является произвольной. Однако, препятствия не будут задаваться посредством градиентного поля. Таким образом, агенты-птицы будут самостоятельно распознавать препятствия на локальном уровне и избегать их.

В новой модели у агентов-птиц сохраняются правила поведения из классической модели и, как и в Главе 2, добавляется правило предотвращения столкновения с препятствием, которое позволяют агенту-птице выбрать лучшее направления движения без последующего столкновения с препятствием. Таким образом,

После выбора направления движения в соответствии с правилами локального поведения, агент выполняет следующие действия:

1. смотрит, нет ли на расстоянии видимости препятствий;

2. если препятствий в зоне видимости агента-птицы нет, то вычисленное по старым локальным правилам направление движения не изменяется;

3. если в зоне видимости агента-птицы обнаружено препятствие, то вычисляется вектор между координатой центра препятствия (bx, by) и координатой, в которой находится агент-птица (x, y), и нормируется:

   $\vartheta = (\vartheta_x, \vartheta_y) := \frac{(dx,dy)}{||(dx,dy)||}$,

   где $dx = bx - x, dy = by - y$;

4. считается нормальное направление к полученному вектору в п.3 и нормируется:

   $norm := \frac{(-\vartheta_y,\vartheta_x)}{||(-\vartheta_y,\vartheta_x)||}$;

5. пусть единичный вектор направления, выбранного по правилам локального поведения, — $heading$; для избегания попадания в область препятствия при высокой скорости определим новый вектор направления агента $heading := 0.03heading + (1 - 0.03)norm$;

6. новый вектор направления нормируется: $heading = (direct_x, direct_y) := \frac{heading}{||heading||}$.

В новой модели движения стаи птиц с локальным контролем динамически изменяющейся среды агент, представляющий

непосредственно птицу, — элемент класса **myBird**, представленного с помощью UML-диаграммы классов:

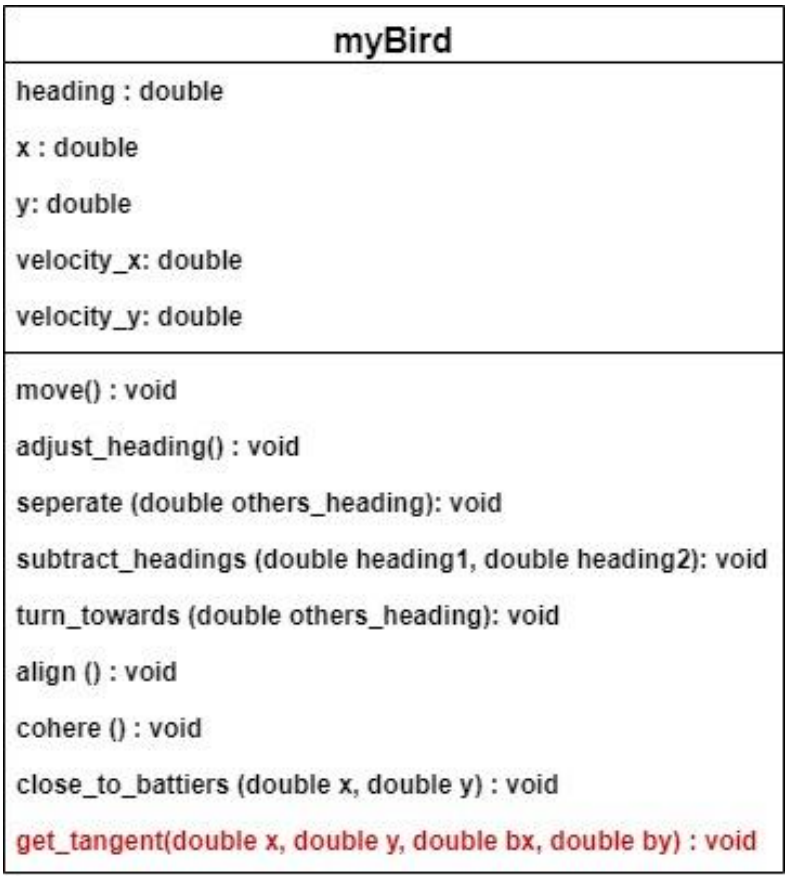

Рис. 3.1 UML-диаграмма класса для агентов-птиц с возможностью локального контроля среды

3.2 Реализация агентной модели

В новой модели параметры препятствий (области круговой формы) и агентов-птицы задаются аналогично, как и в Главе 2. Опишем изменения, внесенные в реализацию агентной модели движения стаи птиц.

Были созданы новые функции для реализации движения агентов-птиц:

1. Функция для определения близко ли расположен агент к препятствию: **close_test(x, y).** Если в зоне видимости есть препятствие, функция возвращает (TRUE, bx, xy), где bx и by - координаты центра масс препятствия, если препятствий вблизи агента-птицы нет, то возвращает (FALSE, -1, -1). Аргументы: **x, y** — координаты агента в области (x, y).

```
def too_close_to_barriders(x, y):
    for b in g_Barriders:
        dist = b.dist_to(x, y)
        if dist <= 80:
            return True, b.x, b.y
    return False, -1, -1
```

2. Функции для подсчета направления обхода препятствия **get_tangent(x, y, bx, by).** Аргументы: **x, y** — координаты агенты в области (x, y); b**x, by** — координаты препятствия.

```
def get_tangent(x, y, bx, by):
    dox = bx - self.x
            doy = by - self.y

            nx = -doy
            ny = dox
            nnorm = math.sqrt(nx * nx + ny * ny)
            nx /= nnorm
            ny /= nnorm
            dirx = self.velocity_x
            diry = self.velocity_y
```

```
            dnorm = math.sqrt(dirx * dirx + diry * diry)
            dirx /= dnorm
            diry /= dnorm

            dirx = dirx * 0.03 + nx * (1 - 0.03)
            diry = diry * 0.03 + ny * (1 - 0.03)

            dnorm = math.sqrt(dirx * dirx + diry * diry)
            dirx /= dnorm
            diry /= dnorm
             dir          =          np.array([dirx,          diry])
             return dir
```

3. Функция **`move(self, close_test)`** была изменена под новые условия задания среды и ее локального контроля. Аргументы: `self` — ссылка на объект (агента); `close_test` — функция, возвращающая результат (TRUE, bx, by), если агент близок к препятствию или (False, -1, -1) иначе.

```
  def move(self, close_test):
        cl, bx, by = close_test(self.x, self.y)

        if cl:
            dir = get_tangent(self.x, self.y, bx, by)

            speed = self.get_speed()

            self.velocity_x = dir[0] * speed
            self.velocity_y = dir[1] * speed
```

```
        self.x += self.velocity_x
        self.y += self.velocity_y

        if (self.x < 0):
            self.x += 1000
        elif (self.x > 1000):
            self.x -= 1000

        if (self.y < 0):
            self.y += 1000
        elif (self.y > 1000):
            self.y -= 1000
```

4. Функция **`simulate_particles(particles, close_test)`** полностью реализует алгоритма из п.3.1 средствами **Python**, с использованием новых функций, описанных выше. Аргументы: `particles` — массив агентов-птиц; `close_test` — функция, возвращающая результат (TRUE, bx, by), если агент близок к препятствию или (False, -1, -1) иначе.

```
def simulate_particles(particles, close_test):
    for p in particles:
        p.adjust_heading(particles)
```

После расчета направлений по правилам локального поведения, учитываем правила предотвращения столкновения с препятствием для каждого агента-птицы и перемещаем их в новую позицию:

```
    for p in particles:
        p.move(close_test)
```

3.3 Результаты моделирования

Анимация движения агентов в динамически изменяющейся среде с ее локальным контролем агентами

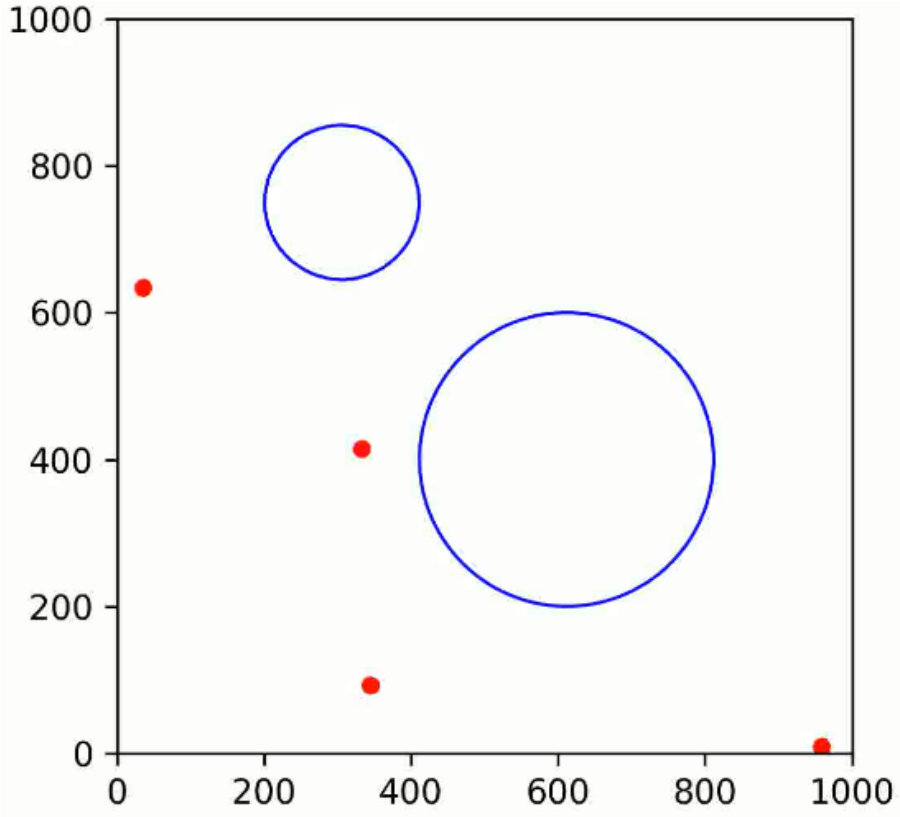

Рис. 3.2 Изображение анимации движения агентов в динамически изменяющейся среде (с локальным контролем)

Посредством визуализации (двумерной) модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией с учетом обхода препятствий посредством локального контроля среды агентами-птицами. В данной модели препятствия изменяют свое положение со временем.

ГЛАВА 4
АГЕНТНАЯ МОДЕЛЬ ДВИЖЕНИЯ СТАИ ПТИЦ С ЛОКАЛЬНЫМ КОНТРОЛЕМ СРЕДЫ И АГЕНТОМ-ЛИДЕРОМ

4.1 Описание агентной модели

В Главе 4 описывается модель стаи птиц, в которой контроль среды производиться на локальном уровне и появляется агент-лидер. Агент-лидер — аналог вожака стаи. Геометрия областей препятствий будет также иметь круговую форму, чтобы не возникали вычислительные сложности в процессе моделирования. Препятствия статичны, то есть их положение не изменяется со временем.

В новой модели у агентов-птиц сохраняются правила поведения из классической модели и модели из Главы 3, добавляется правило ориентации на вожака, которое позволяют агенту-птице следовать за агентом-лидером, если он находится в зоне видимости. Введение в модель агента-лидера, теоретически должна способствовать сплоченности стаи и более синхронному поведению агентов в ней. Так же такая модель более приближена к реальным условиям в стаях птиц, где есть вожак, ведущий стаю. Также в модели появляется понятие стаи, посредством подчинения агентов-птиц своим лидерам. Для этого в реализации алгоритма ориентации на вожака каждому агенту будет указываться его лидер, на которого птица будет равняться в зоне видимости.

Таким образом, после выбора направления движения в соответствии с правилами локального поведения (без правила предотвращения столкновения с препятствием), агент выполняет следующие действия, соответствующие правилу ориентации на вожака:

1. смотрит, нет ли на расстоянии видимости его агента-лидера;
2. если агента-лидера в зоне видимости агента-птицы нет, то вычисленное ранее направление движения не изменяется;
3. если в зоне видимости агента-птицы обнаружен его агент-лидер, то искусственно строится такая система координат, в которой агент-лидер — начало системы координат (его координаты (0, 0) в новой системе), а его вектор скорости имеет направление (0, 1).
4. если в новой системе координат агент-птица идет позади вожака (координата $y > 0$ и:
    a. координата $x > 0$, угол направления изменяется на значение $+\frac{\pi}{2}$,
    b. координата $x < 0$, угол направления изменяется на значение $-\frac{\pi}{2}$,
    c. скорость агента-птицы немного уменьшается;
5. если в новой системе координат агент-птица идет наравне с вожаком или обгоняет его (координата $y \geq 0$), то угол направления изменяется так, что становится равен углу направления вожака;
6. если скорость агента-лидера больше скорости агента-птицы, то скорость птицы увеличивается, иначе уменьшается.

Далее агент-птица выполняет действия по правилам предотвращения столкновения с препятствием, описанным в Главе 3 п.3.1.

В новой модели движения стаи птиц с локальным контролем динамически изменяющейся среды и агентом-лидером агент, представляющий непосредственно птицу, — элемент класса **myBird**, представленного с помощью UML-диаграммы классов:

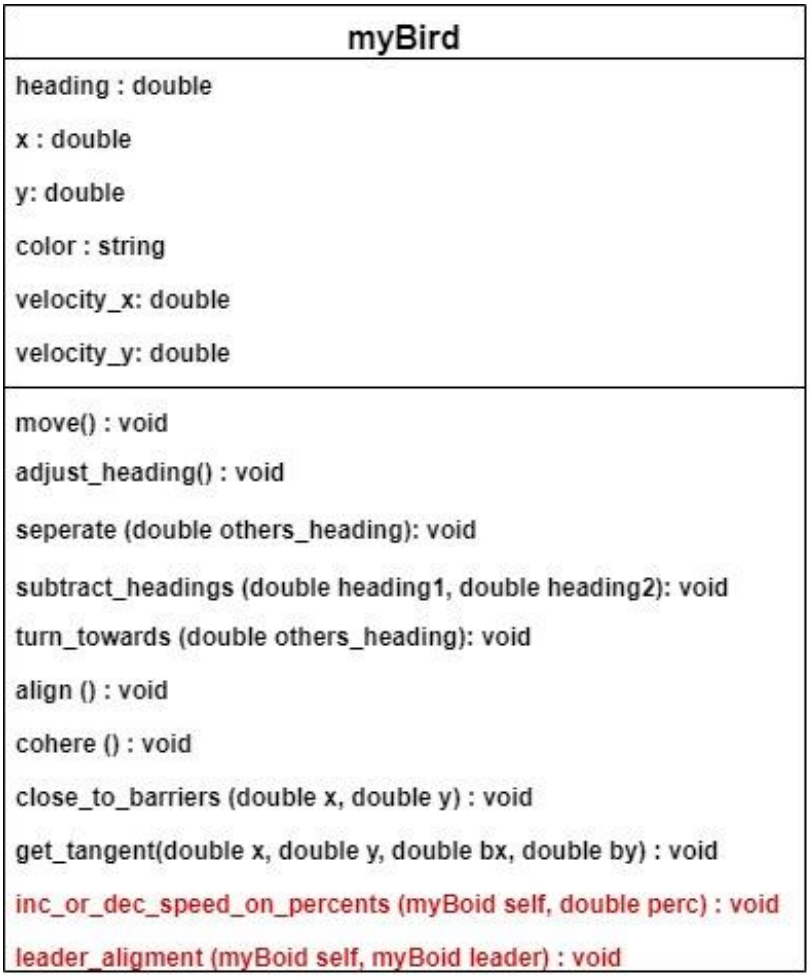

Рис. 4.1 UML-диаграмма класса для агентов-птиц с возможностью локального контроля среды и с агентом-лидером

4.2 Реализация агентной модели

В новой модели правила локального поведения, предотвращения столкновения с препятствием и их реализации такие же, как и в Главе 3. Опишем изменения, внесенные в реализацию агентной модели движения стаи птиц, связанные с введением правила ориентации на вожака.

Были созданы новые функции для реализации движения агентов-птиц:

1. Функция ориентации направления агента-птицы на своего агента-вожака: **`leader_alignment(self, leader).`** Функция реализует алгоритм, описанный в п.4.1 пп.1-6. `Аргументы: self` — ссылка на объект (агента); `leader` — агент-лидер данного агента-птицы.

```
def leader_alignment(self, leader):
      dist_to_leader = math.sqrt((self.x - leader.x) * (self.x - leader.x) + (self.y - leader.y) * (self.y - leader.y))
```

Сравним расстояние до агента-лидера с радиусом видимости агента-птицы:

```
      if dist_to_leader > 200:
            return


      lh = leader.get_heading()
```

Приведем к новой системе координат, где агент-лидер является центром новой системы координат, а его вектор скорости имеет направление (0, 1):

```
      do_rot = math.pi / 2 - lh
```

```
        new_self_x = self.velocity_x * math.cos(do_rot) -
self.velocity_y * math.sin(do_rot)
        new_self_y = self.velocity_x * math.sin(do_rot) +
self.velocity_y * math.cos(do_rot)

        if (new_self_y < 0):
            if (new_self_x > 0):
                self.set_heading(self.get_heading() + math.pi *
            0.5)
            elif (new_self_x < 0):
                self.set_heading(self.get_heading() - math.pi *
            0.5)
            self.inc_or_dec_speed_on_percents(-0.06)
        else:
            self.set_heading(leader.get_heading())

        if leader.get_speed() > self.get_speed():
            self.inc_or_dec_speed_on_percents(0.06)
        elif leader.get_speed() < self.get_speed():
            self.inc_or_dec_speed_on_percents(-0.06)
```

2. Функции для изменения скорости агента-птицы при ориентации на вожака **`inc_or_dec_speed_on_percents(self, perc).`** Данная функция регулирует положение агентов-птиц относительно агентов-лидеров, чтобы первые не обгоняли и не отставали от вожака в зоне видимости.

   `Аргументы: self` — ссылка на объект (агента); `perc` — коэффициент регулировки скорости агента.

```
def inc_or_dec_speed_on_percents(self, perc):
```

```
        speed = (1 + perc) * self.get_speed()
        heading = self.get_heading()
        self.velocity_x = speed * math.cos(heading)
        self.velocity_y = speed * math.sin(heading)
```

3. Функция **`simulate_particles(particles, close_test)`** полностью реализует алгоритма из п.4.1 средствами **Python**, с использованием новых функций, описанных выше. Аргументы: `particles` — массив агентов-птиц; `close_test` — функция, возвращающая результат (TRUE, bx, by), если агент близок к препятствию или (False, -1, -1) иначе.

```
def simulate_particles(particles, close_test):
    for p in particles:
        p.adjust_heading(particles)
```

После расчета направлений по правилам локального поведения, учитываем нахождения вблизи каждого агента-птицы своего вожака и его направление:

```
particles[1].leader_alignment(particles[0])
particles[2].leader_alignment(particles[0])
particles[3].leader_alignment(particles[0])

particles[5].leader_alignment(particles[4])
particles[6].leader_alignment(particles[4])
particles[7].leader_alignment(particles[4])
```

После расчета направлений по правилам локального поведения и правилам ориентации на вожака, учитываем наличие вблизи каждого агента-птицы препятствий и перемещаем агентов в новую позицию:

```
for p in particles:
    p.move(close_test)
```

4.3 Результаты моделирования

Анимация движения агентов в среде с ее локальным контролем и агентом-лидером

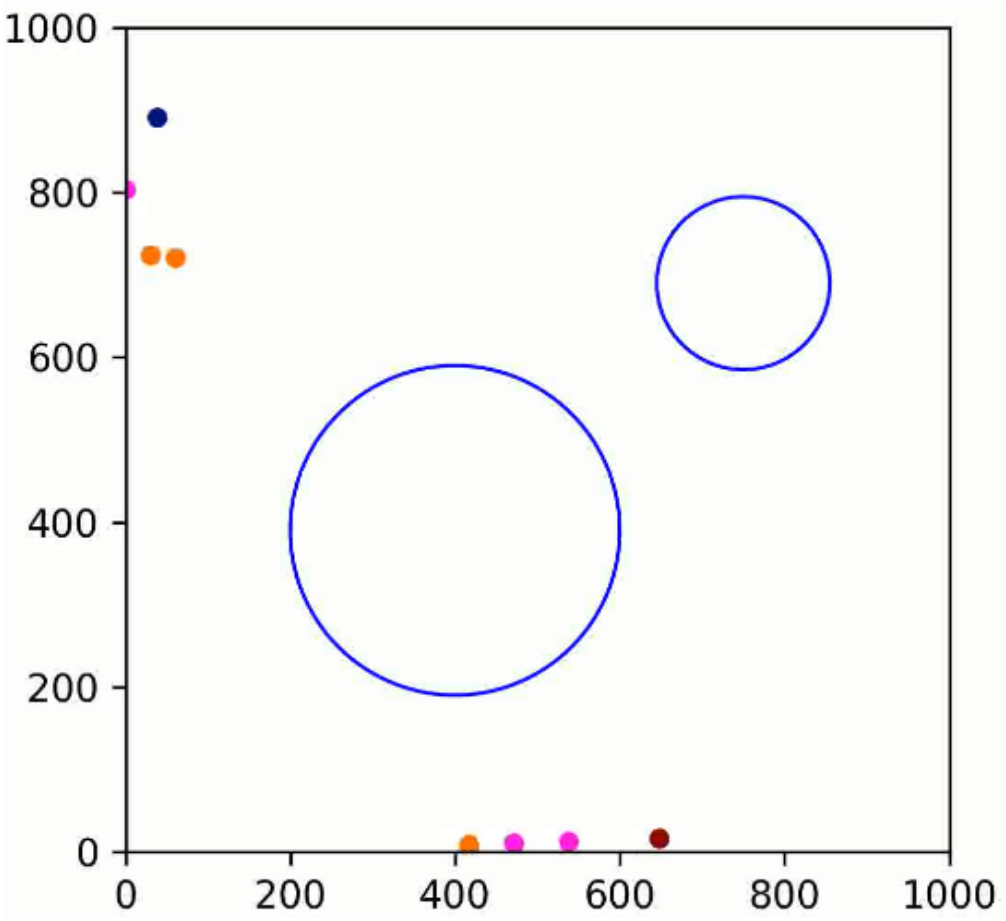

Рис. 4.2 Изображение анимации движения агентов в среде (с локальным контролем и агентом-лидером)

Посредством визуализации (двумерной) модели движения можно наблюдать взаимодействие агентов в среде с тороидальной топологией с учетом обхода препятствий посредством локального контроля среды

агентами-птицами. В данной модели препятствия не изменяют свое положение со временем. Синие агент — агент-лидер для агентов-птиц желтого цвета, красный агент — агент-лидер для агентов-птиц розового цвета.

ГЛАВА 5 СРАВНИТЕЛЬНЫЙ АНАЛИЗ

В рамках дипломного проекта были реализованы три модели движения стаи птиц:

1. с комбинированием агентного подхода и численного решения краевой задачи, где агент обладает:
    a. **локальной информацией** о положении соседних агентах,
    b. **глобальной информацией** о препятствиях в среде;
2. с агентным подходом, где агент обладает только **локальной информацией** о соседних агентах и о наличии препятствий в области видимости;
3. с агентным подходом, где агент обладает только **локальной информацией** о соседних агентах, о наличии препятствий в области видимости и об агенте-лидере среди соседей.

После реализации указанных моделей и их сравнения были сделаны следующие выводы:

1. Расчеты с комбинированным подходом требуют бóльших временных и вычислительных затрат, так как на каждом шаге (итерации) средствами **FEniCS** в **Python** решалась задача Дирихле для уравнения Лапласа с новыми условиями. Под новыми условиями подразумевается изменение положения областей (препятствий), на которых задаются граничные условия.

2. Добавление в систему агента-лидера является дополнительным механизмом для адаптивного поведения агентной системы. Агент-лидер

может быть использован для моделирования задач целевого поведения системы, таких как облет препятствий, преследование объектов или достижение системой заданного положения в пространстве.

.

ЗАКЛЮЧЕНИЕ

В ходе дипломной работы были сделаны следующие выводы. Во-первых, для построения агентной модели движения стаи птиц в среде с препятствиями можно использовать **глобальное управление**. Во-вторых, к такой модели с динамически изменяющейся средой может быть применено комбинирование агентного подхода и численного решения краевой задачи (для задания глобального управления). Посредством метода конечных элементов было построено решение задачи Дирихле для уравнения Лапласа и построено градиентное поле [4]. Таким образом, с помощью полученного градиентного поля были реализованы правила предотвращения столкновения агента с препятствием.

В-третьих, есть возможность использовать только агентный подход. В таком случае агент обходит препятствия и избегает столкновения с другими агентами на основании **локальной информации** об их присутствии в радиусе видимости. В-четвертых, модель можно дополнить агентом-лидером, как дополнительным механизмом для адаптивного поведения агентной системы. Агента-лидера можно использовать для моделирования задач целевого поведения системы, а именно для обхода препятствий, преследование и выравнивания на объекты и других агентов, а также для того, чтобы установить заданное положение системы в пространстве.

В первой главе дипломной работы приводится информация о таких понятиях, как агентное моделирование, агент, и классическая модель движения стаи птиц. В главе описываются параметры агентов и их функции, как элементов класса. Функции согласуются с локальными правилами поведения агентов классической модели движения стаи птиц.

Во второй главе содержится описание агентной модели движения стаи птиц в динамически изменяющейся среде, алгоритм и правила поведения для предотвращения столкновения с препятствиями. Так как в данной модели используется и локальное, и глобальное управление, в начале главы приведена постановка задачи Дирихле для уравнения Лапласа и ее численное решение средствами **FEniCS** в **Python.** Была описана реализация самой модели движения стаи птиц, где положение препятствий изменяется со временем, также приведены изображения анимаций реализованной модели в **Python**.

В третьей главе описаны алгоритм и правила предотвращения столкновения с препятствием для агентной модели движения стаи птиц в среде с локальным контролем. Также представлены изменения в реализации модели в **Python**, приведено изображение анимации реализованной усовершенствованной агентной модели.

В четвертой главе описаны алгоритм и правила ориентации на вожака для агентной модели движения стаи птиц в среде с агентом-лидером. Представлены изменения и дополнения в реализации модели в **Python**,

описанной в Главе 3, приведено изображение анимации реализованной новой агентной модели.

В главе 5 описаны результаты сравнительного анализа моделей, описанных в данной дипломной работе.

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

В дальнейшем целесообразным является совершенствование адаптационных механизмов изучаемой агентной системы за счет имитации процессов запоминания и обучения для каждого агента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Каталевский, Д. Ю., Основы имитационного моделирования и системного анализа в управлении: учебное пособие; 2-е изд., перераб. и доп. М.: Издательский дом «Дело» РАНХиГС, 2015.
2. Кельтон, В., Лоу А., Имитационное моделирование (перевод с английского). 3-е изд. СПб.: Питер, 2004.
3. Лаврова О.А., Лабораторная работа. ДВ Метод конечных элементов, ММФ, КМ и СА, БГУ, октябрь 2017. [Электронный ресурс] – Режим доступа: http://km.mmf.bsu.by/courses/2017/fem/FEMLR3_Poisson.html
4. Пашкевич А., Комбинирование агентного моделирования с численным решением математических моделей, ММФ, КМ и СА, БГУ, 2019. [Электронный ресурс] – Режим доступа: http://km.mmf.bsu.by/courses/2019/msmod/CourseDiplomWorks/PashkevichAngelina2019.pdf
5. Старовойтова В., Объектно-ориентированная реализация агентных моделей, курсовая работа, ММФ, КМ и СА, БГУ, 2018. [Электронный ресурс] – Режим доступа: http://km.mmf.bsu.by/courses/2019/msmod/CourseDiplomWorks/StarovoitovaViktoriya2018.pdf
6. Agent-Based Modelling and Simulation in AnyLogic. Cells and Birds, Otto von Guericke University, Magdeburg, 2015

7. AnyLogic – платформа для имитационного моделирования [Электронный ресурс]. – Режим доступа: https://www.anylogic.ru/.
8. FEniCS - это исследовательский и программный проект, направленный на создание математических методов и программного обеспечения для автоматизированного вычислительного математического моделирования. – Режим доступа: https://fenicsproject.org/.
9. Python – высокоуровневый язык программирования общего назначения. – Режим доступа: https://www.python.org/.
10. S. Goldenstein etc., Scalable Dynamical Systems for Multi-Agent Steering and Simulation, 2001 // Computers & Graphics 25(6):983-998.