МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

МОДЕЛИРОВАНИЕ МУРАВЬЯ ЛЭНГТОНА

Курсовая работа

Галушкин Дмитрий Алексеевич

студента 2 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

Агентное моделирование является одним из основных методов исследования сложных систем, и в последнее время этот подход моделирования активно развивается. Причина высокого уровня интереса к агентному моделированию заключается в том, что данный инструмент позволяет имитировать поведение и отношения в сложных социальных условиях, где традиционные аналитические инструменты часто не справляются.

Агентное моделирование систем основано на описании локальных правил поведения участников системы (агентов) и последующей эмуляции этих поведений на микроуровне для получения закономерностей поведения исходной системы на макроуровне. Такой подход позволяет моделировать и изучать интегративные (эмерджентные) свойства системы как результата низкоуровневого взаимодействия элементов системы. В данной курсовой работе рассказывается об агентных моделях в компьютерной среде. Агентные модели описывают поведение целого объекта исследования посредством описания поведений локальных элементов.

Курсовая работа носит исследовательский характер. Целью работы является изучение принципов описания агентных моделей в дискретной и непрерывной среде средствами AnyLogic.

Основными задачами курсовой работы являются изучение и реализация агентной модели для случая локального взаимодействия агентов и дискретной среды на примеры моделей «муравей Лэнгтона» и изучение и реализация агентной модели для случая локального взаимодействия агентов в непрерывной среде на примере модели, имитирующей поведение стаи птиц.

Глава 1 Общие положения

1.1 Понятие сложной системы

Термин «система» употребляется в очень широком смысле. Система – это множество взаимосвязанных элементов, которые взаимодействуют для достижения определенных целей. Целью системы называется ее желаемое состояние.

Сложными системами называются системы, состоящие из большого числа взаимосвязанных и взаимодействующих между собой элементов. Сложная система определяется как система, имеющая много независимых элементов, каждый из которых может взаимодействовать с остальными. Понятие сложной системы пользуются в системотехнике, системном анализе, исследовании операций и при системном подходе в различных областях науки, техники и народный хозяйства.

Примеры сложных систем являются: энергетические комплексы, телефонные сети крупных городов, информационные системы, производственные процессы крупного предприятия, системы управления полетом в крупных аэропортах, отраслевые системы управления и др. [2]

При исследовании сложных систем изучаются как части системы и в отдельности, так и вся система целиком. Строгое исследование процесса функционирования сложных систем практически невозможно. Определение аналитической модели сложной системы затрудняется множеством условий, определяемых особенностями работы системы, взаимодействием ее составляющих частей, влиянием внешней среды и т.п.

Например, для исследования поведения большого количества людей на стадионе не следует усреднять какие-либо значения для удобного изучения, поскольку это может привести к недействительным результатам. В данном случае строиться компьютерная модель всех людей на стадионе, чтобы потом можно было изучить поведение толпы [3].

Таким образом компьютерное моделирование является одним из эффективных методов изучения, описания и исследования сложных систем, и позволяет проводить вычислительные эксперименты в тех случаях, когда реальная постановка эксперимента затруднена или может дать неточный результат.

1.2 Агентное моделирование

Имитационное моделирование – это построение компьютерных моделей и проведение экспериментов над ними [4].

Имитационный подход применяют, когда параметров много, зависимости не линейны, система имеет качественно различные состояния (непрерывные процессы прерываются дискретными переходами), траекторию во времени (объект эволюционирует), обладает вероятностным поведением и обратными связями. Имитационный подход незаменим, когда нужно сопроводить модель анимационной презентацией (симуляцией). При создании виртуальных тренажеров, моделей движения транспорта и пешеходов это может оказаться главной задачей моделирования.

AnyLogic – единственное ПО для агентного моделирования профессионального класса. Кроме того, в AnyLogic агентное моделирование комбинируется с дискретно-событийным подходом или системной динамикой [4].

Это современная среда разработки моделей на языке Java с русскоязычным графическим интерфейсом и тщательно продуманной контекстной справочной системой [5].

AnyLogic поддерживает все известные методы имитационного моделирования:

- Системная динамика (используется, если имеется информация только о глобальных зависимостях)
- Дискретно-событийное моделирование (используется, если система может быть описана как процесс)
- Агентное моделирование (используется, если в модели предполагается несколько индивидуальных объектов).

Так же в AnyLogic можно комбинировать несколько методов, если модели присущи несколько особенностей.

Доступность трёх методов моделирования одновременно дает гибкость, необходимую для решения любой поставленной задачи [4].

Моделирование на основе агентов или агентное моделирование – это модельная и вычислительная основа для моделирования динамических процессов, которая включает автономных агентов.

Агентное моделирование используется для моделирования сложных систем, состоящих из большого количества взаимодействующих подсистем. Также

агентное моделирование хорошо применимо в случае, когда слишком трудно или невозможно формализовать поведение системы на глобальном уровне [4].

Под агентом в агентном моделировании понимается элемент модели, который может иметь поведение, память, контакты и т.д. Агенты могут моделировать людей, компании, проекты, автомобили, города, животных, корабли и т.д.

В рамках этого подхода, система рассматривается как совокупность взаимодействующих частей – агентов, каждый из которых действует самостоятельно по заранее определенным правилам и может взаимодействовать с другими агентами. Таким образом, поведение всей системы складывается из взаимодействия ее частей. Изначально агентный подход был разработан для моделирования социальных процессов в человеческих сообществах. Сейчас он активно применяется в самых разных областях: от моделирования процессов фондовых бирж до предсказания распространения инфекционных и сердечно-сосудистых заболеваний [6].

Агентное моделирование сосредоточено на индивидуальных участниках системы. В этом заключается его отличие от более абстрактного метода системной динамики и дискретно-событийного метода, ориентированного на процессы.

Создание стандартной агентной модели в AnyLogic заключается в задании двух типов агентов. К примеру, тип агента Main для описания высокоуровневого объекта, где содержатся все агенты, и тип Person для описания агента. Каждый тип будет подтипом базового типа Agent. Число агентов было бы включено в тип Main как дубликаты объекта "человек" типа Person. Среда может быть определена на уровне типа агента Main для установления свойств среды агентов. Впрочем, вы можете легко определить другие иерархии в вашей агентной модели, например, у вас могут быть компании-агенты, которые содержат служащих-агентов и общаются с потребителями-агентами [4].

Глава 2 Агентные модели в дискретной среде

2.1 Клеточный автомат

Клеточный автомат – дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Включает регулярную решетку ячеек, каждая из которых может находиться в одном из конечного множества состояний, таких как 1 и 0.

Решетка может быть любой размерности. Для каждой ячейки определено множество ячеек, называемых окрестностью. К примеру, окрестность может быть определена как все ячейки на расстоянии не более 2 от текущей. Для работы клеточного автомата требуется задание начального состояния всех ячеек и правил перехода ячеек из одного состояния в другое. На каждой итерации, используя правила перехода и состояния соседних ячеек, определяется новое состояние каждой ячейки. Обычно правила перехода одинаковы для всех ячеек и применяются сразу ко всей решётке [7].

Пожалуй, наиболее частое и развитое направление применения клеточных автоматов — это математическое моделирование динамических процессов. При математическом моделировании физических явлений часто возникает ситуация, когда рассматриваемую задачу нельзя решить аналитически, а ее расчет в виде сложной схемы приводит к появлению различного рода неустойчивостей.

В данный момент клеточные автоматы используются как вычислительный инструмент для широкого спектра различных задач. Они могут упрощать расчеты в тех случаях, когда традиционные подходы приводят к сложным и требующим большого времени вычислениям.

Некоторые примеры применения клеточных автоматов:

- Клеточные автоматы предложены для использования в криптосистемах, поскольку их первоначальное состояние сложно найти. Двумерные клеточные автоматы используются для генерации случайных чисел.
- Клеточные автоматы используются в компьютерном моделировании физических процессов.
- Благодаря своей гибкости и универсальности, клеточные автоматы нашли признание в теории искусственного интеллекта, микро- и макробиологии, вычислениях в среде с возможностью сбоев, в системах распознавания языка и изображений [8].

2.2 Модель муравья Лэнгтона

Рассмотрим одну из простейших сложных систем – «муравьем Лэнгтона». Муравей был придуман в 1986 году американским программистом и одним из основателей области искусственной жизни Кристофером Лэнгтон для развития теории и приложений сложных систем и искусственной жизни. Технически это двумерный клеточный автомат, система клеток квадратной решетки, состояния которых обозначаются цветом [9].

Для понимания сути модели «Муравья Лэнгтона» как двумерного клеточного автомата вводится бесконечная решетка, которая может быть закрашена черными или белыми квадратами. На этой плоскости находится сам муравей, который, в зависимости от состояния клетки, на которой находится, может двигаться по следующим правилам:

- При нахождении на черном квадрате – повернуть на 90° влево, изменить цвет квадрата на белый, сделать шаг вперед на следующую клетку.
- При нахождении на белом квадрате - повернуть на 90° вправо, изменить цвет квадрата на чёрный, сделать шаг вперед на следующую клетку.

Эти два простых правила вызывают сложные поведения модели. Так движения муравья можно разделить на три четкие фазы:

- «Простота»: начав движения на абсолютно белой плоскости, муравей во время первых несколько сотен шагов создает очень простые и зачастую симметричные образцы.
- «Хаос»: после несколько сотен шагов появляется большой и нерегулярный, несимметричный образец. Муравей прослеживает псевдослучайный путь приблизительно до 10 000 шагов.
- «Строительство магистрали»: после хаотичного движения, какой бы ни была первоначальная конфигурация, в конце концов муравей непременно принимался за строительство магистрали при помощи все того же 104-шагового цикла.

Все конечные начальные конфигурации, проверенные в конечном счете, сходятся к тому же самому повторному образцу, предполагая, что «шоссе» - аттрактор муравья Лэнгтона, но никто не был в состоянии доказать, что это верно для всех таких начальных конфигураций. Это одна из фундаментальных нерешенных задач теории сложности [9].

2.3 Моделирование муравья Лэнгтона в AnyLogic

При моделировании муравья будем использовать подход агентного моделирования. Такой подход в моделировании дает возможность указать индивидуальность поведения каждого объекта, моделирования эмерджментных свойств системы.

Создание стандартной агентной модели в AnyLogic заключается в моделировании отдельных элементов системы: агента и его среды. По условию муравей находится на бесконечной решетке, которая может быть закрашена черными или белыми квадратами, что определяет среду для муравья - двумерное дискретное пространство.

Двумерное дискретное пространство представляет собой прямоугольный массив ячеек, полностью или частично занятых агентами. В одной ячейке может находиться не больше одного агента. Поддержка этого типа пространства в AnyLogic включает в себя возможности по распределению агентов по ячейкам, их перемещению в соседние или любые другие ячейки, определению того, какие агенты являются соседями (согласно выбранной модели соседства), нахождению свободных ячеек и т.д. [4]

В AnyLogic присутствует две модели соседства: Мурово (8 соседних ячеек) и Евклидово (4 соседних ячейки). На рисунке 2.1 представлены размерности и типы соседства в дискретном пространстве.

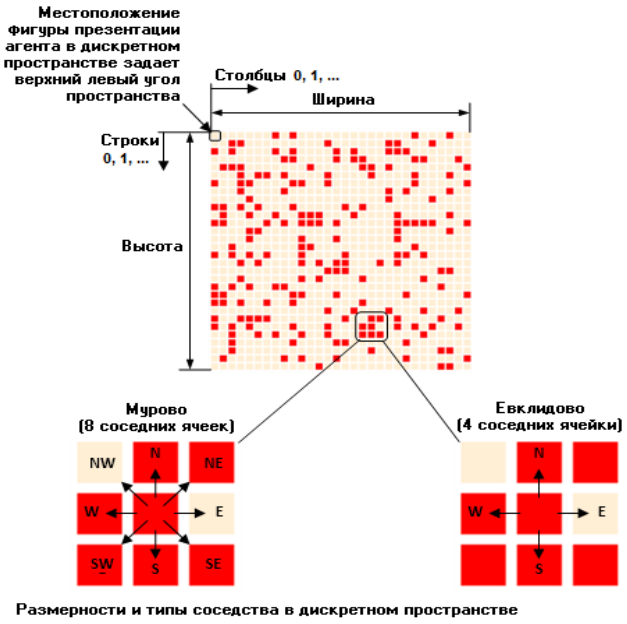


Рисунок 2.1 Размерности и типы соседства в дискретном пространстве [4]

Создадим среду для муравья. В нашем случае средой для муравья будет матрица размером 100×100 ячеек. Щелкая по клетке, можно изменять ее цвет.

Создадим нового агента и назовем его Cell. В окне открывшегося графического редактора класса Cell поместим элемент Прямоугольник в начало координат. Прямоугольник отражает состояние ячейки, поэтому первоначально он будет белого цвета. Теперь введем возможность изменять цвет ячейки по щелчку. Добавим переменную click (тип – boolean, начальное значение – false).

В свойствах прямоугольника в «Действие по щелчку» запишем код:

```
this.rect.setFillColor(red);
click=!click;
if(click==true){this.rect.setFillColor(red);}
else {this.rect.setFillColor(white);};
```

Таким образом мы создали одну клетку среды муравья. Теперь определим саму среду – матрицу, состоящую из 10000 таких ячеек.

Создадим новый класс MyCell. Из окна структуры перетащим элемент класса Cell и зададим количество клеток. В настройках свойства класса MyCell выберем тип пространства - дискретное, размерность 100×100 (100 столбцов и 100 строк) и соответственно Ширину и Высоту среды- 1000 и 1000, тип соседства – Евклидово.

Среда для муравья создана. Теперь построим двухуровневую агентную модель муравья Лэнгмана.

Создадим нового агента и назовем его Ant. С помощью инструментов рисования создадим образ муравья и сгруппируем графические примитивы в фигуру муравья, назвав его pict (рисунок 2.2). Поместим фигуру в начало координат.

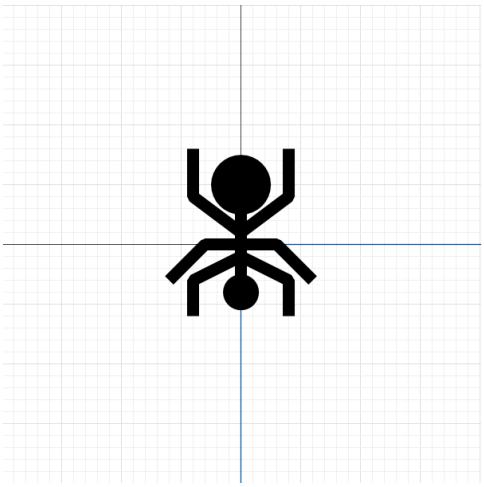


Рисунок 2.2 Изображение муравья

Далее введем параметр flag. Этот параметр будет отображать направление муравья в Евклидовом пространстве (“N”,“S”,“E”,“W”). В свойствах параметра выберем тип String, выбрав значение по умолчанию “N” (север).

На рисунке 2.3 изображены все возможные направления муравья в пространстве.

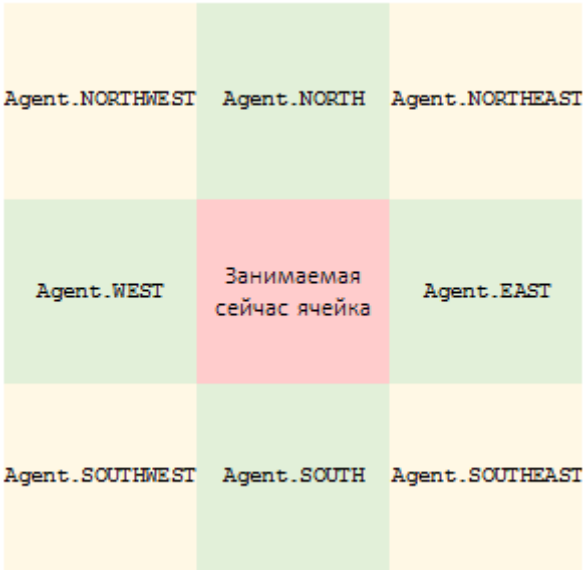


Рисунок 2.3 Иллюстрация направлений муравья в пространстве [4]

Введем еще один параметр dir. Это параметр будет отображать угол поворота фигуры муравья. Тип параметра будет double, а по умолчанию значение будет 0.

В дискретном пространстве один агент может занимать одну ячейку. Агента можно перемещать в любую пустую ячейку. Ячейка определяется строками и столбцами. Для того, чтобы узнать расположения агента в AnyLogic есть встроенные функции:

int getR() - возвращает строку ячейки текущего расположения агента.

int getC() - возвращает столбец ячейки текущего расположения агента.

Для мгновенного перемещения агента в какую-либо пустую ячейку с заданными координатами есть функция jumpToCell(int r, int c), где r - строка ячейки, c - столбец ячейки.

Создадим функцию checkCell(), которая будет определять перемещения агента. С учетом фаз движений модели можно поместить муравья в центр среды и установить правило остановки движения по достижению границ среды. Опишем поведение муравья для одного случая, все остальные будут выглядеть аналогично за счет правил перемещения агента, описанных выше.

Определим правило остановки муравья по достижению границ среды:

int r=cell.getR();

```
 int c=cell.getC();
//остановка, если муравей достиг границ
if ((getC()==1)||( getC()==99)||( getR()==1)||( getR()==99))
{get_Main().pauseSimulation();}
```

Опишем поведение агента. В данном случае направление муравья на север “N” и располагается он в клетке белого цвета. Согласно правилам передвижения, меняем направления муравья на восток “E”, поворачиваем изображения муравья на 90 градусов вправо, меняем цвет ячейки на красный и делаем шаг вперед.

```
if ((getC()==c)&(getR()==r)&(cell.rect.getFillColor()==white)&flag=="N")
{
cell.rect.setFillColor(red);
flag="E";
dir=dir+PI/2; pict.setRotation(dir);
jumpToCell(r,c+1);
break;
 };
```

По аналогии описываются и другие случаи направления агента.

Теперь осталось только настроить действия поведения муравья. В свойствах агента Ant в ячейке “Действия при запуске” установим первоначальные значения для муравья, и действия на шаге: checkCell().

На данном этапе возникает проблема: два агента не могут находится в одной точке одновременно, что не позволяет поместить муравья в среду клеток. Для решения этой проблемы создадим класс MyAnt – среда муравья, параметры которой будут совпадать с параметрами среды. Из своей среды (MyAnt) муравей будет контролировать состояние клеток среды MyCell.

Поместим популяции агентов в myCell и myAnt в Main. Таким образом архитектуру модели муравья Лэнгтона можно изобразить на рисунке 2.4.

Запустив модель можно наблюдать все три стадии поведения муравья (Начальное поведение см. Рисунок 2.5, промежуточное поведение муравья см. Рисунок 2.6). Как и было сказано выше, конечным поведением муравья стало строительство «магистрали» (см. Рисунок 2.7).

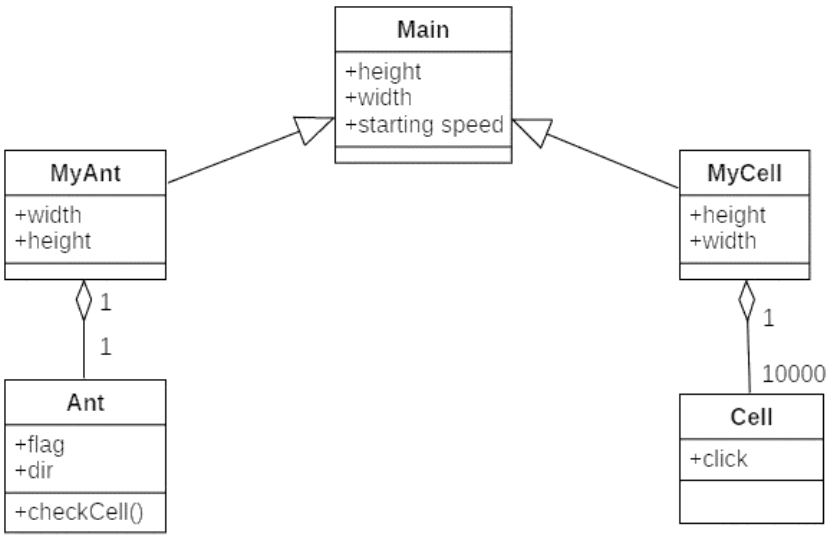


Рисунок 2.4 Диаграмма классов реализации агентного подхода моделирования муравья Лэнгтона в AnyLogic.

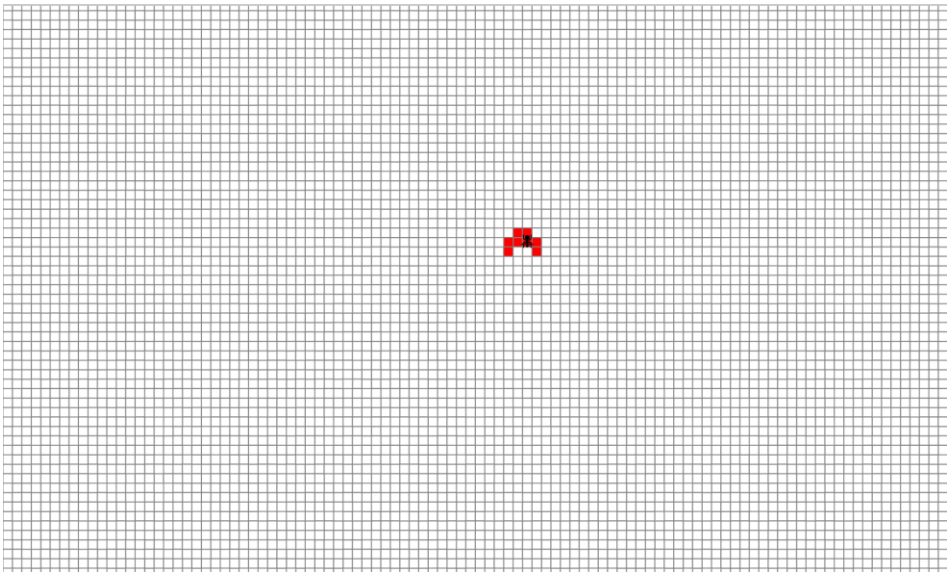


Рисунок 2.5 Начальное поведения муравья Лэнгтона.

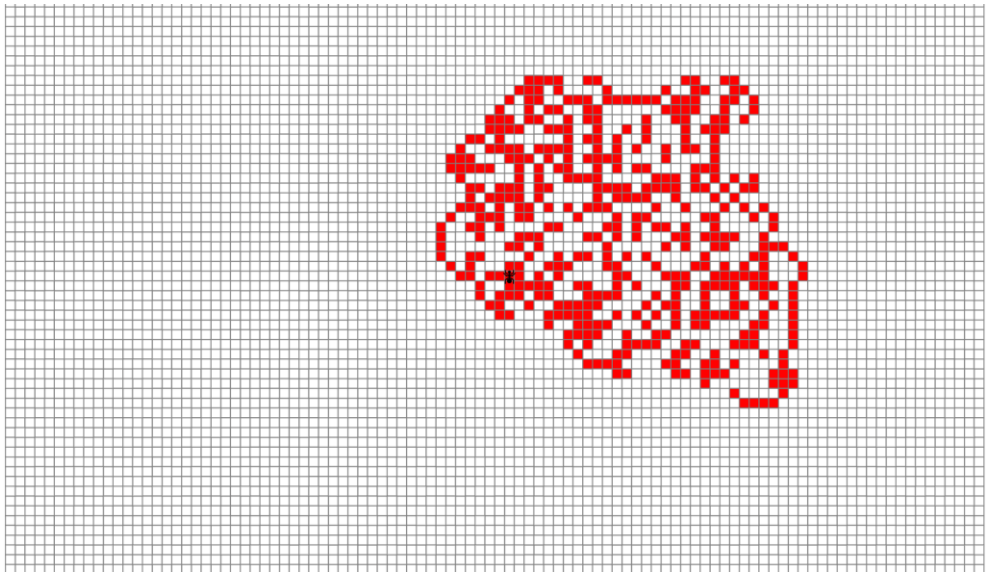


Рисунок 2.6 Промежуточное поведение муравья Лэнгтона.

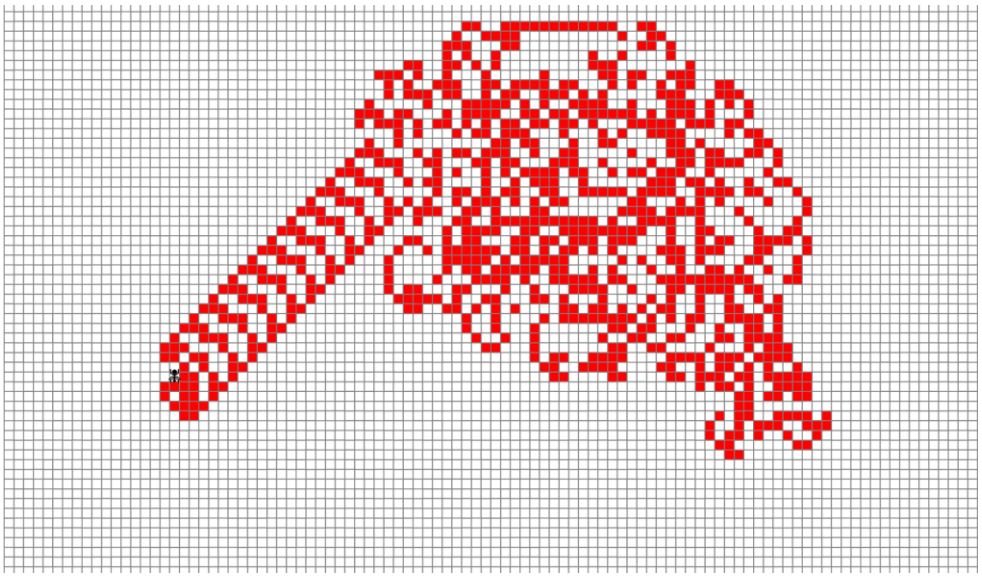


Рисунок 2.7 «Строительство магистрали» муравьем Лэнгтона.

2.4 Модель Sugarscape

Одним из базовых примеров клеточного автомата является модель Sugarscape.

Sugarscape - это модель для искусственного интеллектуального социального моделирования на основе агентов, которые соблюдают некоторые правила поведения, представленным Джошуа М. Эпштейном и Робертом Экстеллом [11].

Все вариации модели Sugarscape включают агентов (жителей), среду (двумерную сетку) и правила, регулирующие взаимодействие агентов друг с другом и средой [11].

Оригинальная модель, представленная Дж. Эпштейном и Р., основана на сетке ячеек 50x50, где каждая клетка может содержать различные количества сахара. На каждом шагу агенты осматриваются, находят ближайшую клетку, наполненную сахаром, двигаются и питаются сахаром. Они могут оставлять загрязнения, умирать, размножаться, передавать информацию, обменивать или заимствовать сахар, генерировать иммунитет или передавать болезни - в зависимости от конкретного сценария и переменных, определенных при настройке модели.

Сахар в симуляции можно рассматривать как метафору ресурсов в искусственном мире, с помощью которого экзаменатор может изучать влияние социальной динамики, такой как эволюция, семейное положение и наследование, на население [13].

Подробнее рассмотрим модель с базовыми правилами поведения агентов.

Окружающая среда представляет собой сетку 50 × 50, которая оборачивается, образуя тор. Ячейки сетки имеют как уровень сахара, так и емкость сахара c. Уровень сахара в клетке - это количество единиц сахара в клетке, а емкость сахара c - это максимальное значение, которое уровень сахара может принимать в этой клетке.

Емкость сахара фиксирована для каждой отдельной клетки и может быть различной для разных клеток. Пространственное распределение сахарных мощностей изображено на рисунок 2.8.

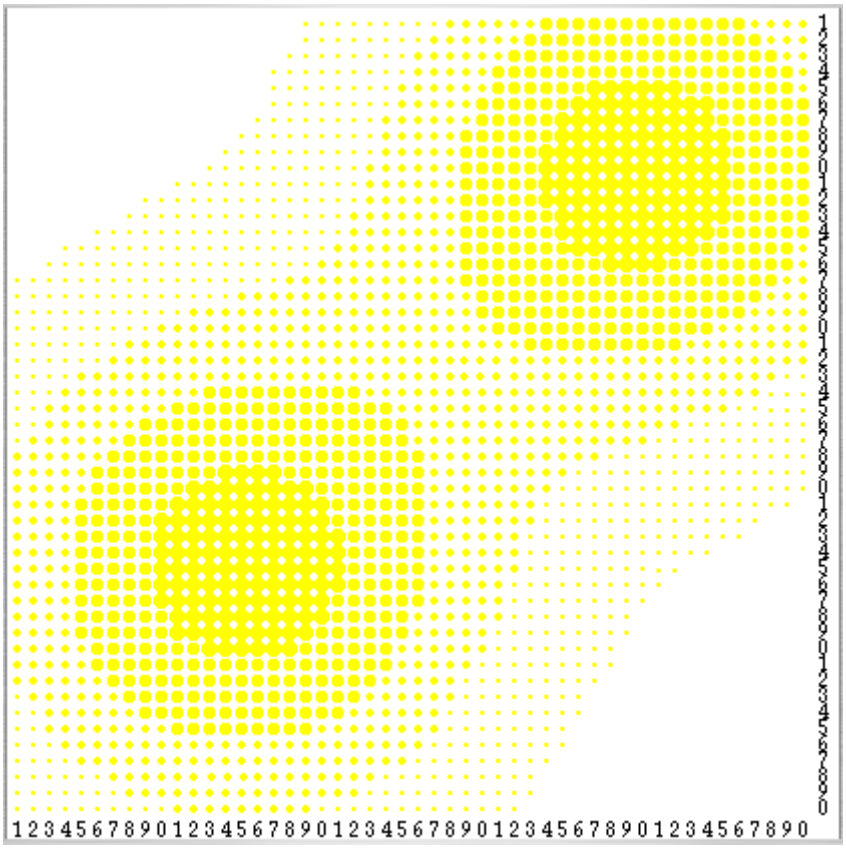


Рисунок 2.8 Среда Sugarscape [12]

В каждой клетке сахар накапливается со скоростью α единиц за шаг до емкости клетки c.

Каждый агент наделен индивидуальными (пожизненными) характеристиками, которые определяют его навыки и способности выживания в среде Sugarscape:

- Поле зрения v, который представляет собой максимальное количество ячеек, которые агент может видеть в каждом из четырех основных направлений решетки: север, юг, восток и запад.
- Скорость метаболизма m, которая представляет собой единицы сахара, которые агент сжигает за шаг.
- Максимальный возраст max_age - максимальное количество шагов, которые может прожить агент.

Агенты также обладают способностью накапливать сахар w. Накопленный сахар агента увеличивается в конце каждого шага за счет собранного сахара и уменьшается за счет метаболизма агента. Два агента не могут занимать одну и ту же ячейку в сетке.

Поведение агентов определяется следующими двумя правилами:

Правило движения агента:

Агент рассматривает незанятые клетки в своем поле зрения (включая ту клетку, на которой он стоит), определяет клетки с наибольшим количеством сахара, выбирает ближайшую (случайным образом, если их несколько), перемещается туда и собирает весь сахар в клетке. В этот момент накопленный сахар агента увеличивается за счет собранного сахара и уменьшается за счет метаболизма агента m. Если в этот момент накопленный сахар агента не больше нуля, то агент умирает.

Правило замены агента:

Всякий раз, когда агент умирает, его заменяет новый агент возраста 0, помещенный в случайно выбранную незанятую ячейку со случайными атрибутами v, m и max-age и случайным накопленным сахаром w. Все случайные числа взяты из равномерных распределений с диапазонами, указанными в таблице 2.1.

Длина решетки L	50
Распределение сахарных емкостей в клетках	См. Рисунок 2.7
Скорость накопления α	1
Количество агентов	250
Начальный накопленный сахар агента в диапазоне	От 5 до 25 единиц
Скорость метаболизма агентов m в диапазоне	От 1 до 4 единиц
Поле зрения агентов в диапазоне	От 1 до 6 единиц
Максимальный возраст агентов в диапазоне	От 60 до 100

Таблица 2.1 Параметризация модели

Первоначально каждая ячейка Sugarscape содержит уровень сахара, равный ее емкости сахара *c*, и 250 агентов создаются в случайном незанятом начальном местоположении и со случайными атрибутами (с использованием равномерного распределения, указанного в таблице 2.1).

Таким образом архитектуру модели Sugarspace можно изобразить на рисунке 2.9.

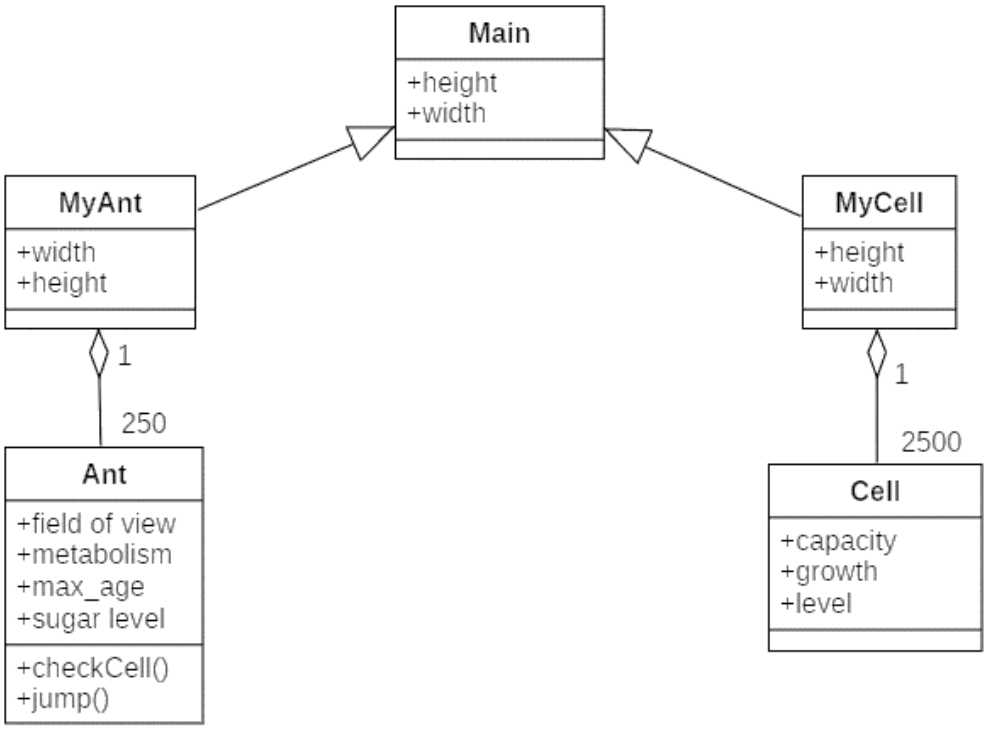


Рисунок 2.9 Диаграмма классов реализации агентного подхода моделирования Sugarspace.

Глава 3 Модель поведения стаи птиц

3.1 Описание агента

Данная модель (Boids) имитирует поведения стаи птиц в нескольких измерениях. Эта модель широко используется в играх для имитации хаотического или похожего на жизнь группового движения.

Алгоритм Boids был создан Крейгом Райнольдсом (Craig Reynolds) в 1986-м году как модель перемещения стаи птиц. В его основе лежат три следующих правила:

- Каждая птица старается не приближаться к другим птицам на расстояние меньшее некоторой заданной величины (правило разделения);
- Каждая птица старается выбрать свой вектор скорости наиболее близким к среднему вектору скорости среди всех птиц в своей локальной окрестности (правило выстраивания);
- Каждая птица старается расположиться в геометрическом центре масс своей локальной окрестности (правило сближения).

Первое правило предназначен для того, чтобы избежать столкновений среди птиц, второе — координирует скорость и направление полета, третье — заставляет каждую птицу не отрываться от стаи (а в идеале — расположится внутри стаи). Поведения птицы изображено на рисунке 3.1.

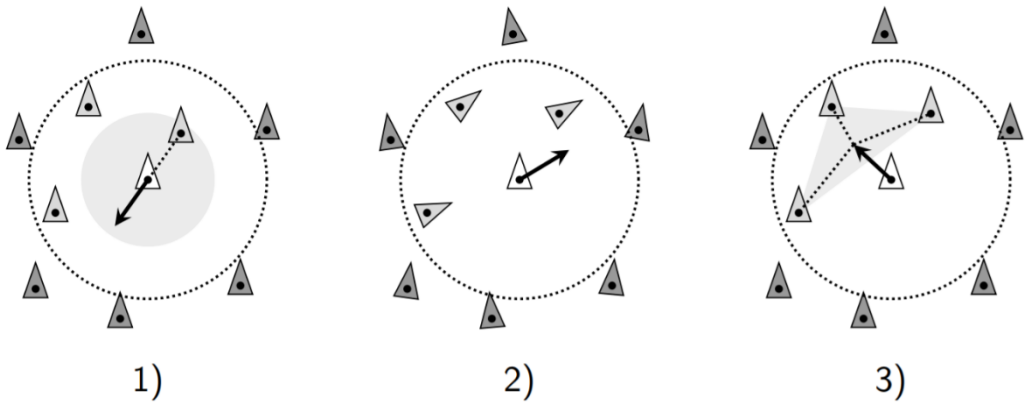


Рисунок 3.1 Правила поведения стаи птиц.
1) Правило разделения 2) Правило выстраивания 3) Правило сближения

Поверхность, на которой происходит движение птиц, представляет из себя “закольцованный” квадрат (т.е. поверхность тора). Все птицы перемещаются с одной и той же постоянной по модулю скоростью.

Могут быть добавлены более сложные правила, такие как обход препятствий и поиск цели.

Движение птиц может быть охарактеризовано как хаотичное или упорядоченное. Поведения птиц, такие как разделение стаи и воссоединение после преодоления препятствий, могут рассматриваться как возникающие.

Модель Boids может использоваться для непосредственного управления и стабилизации команд простых беспилотных наземных транспортных средств или микроавтобусов в робототехнике. Также модель часто используется в компьютерной графике, обеспечивая реалистичное представление стай птиц и других существ, таких как стаи рыб или стада животных. Первой анимацией, созданной с помощью модели, в фильме “Stanley and Stella in: Breaking the Ice” (1987), после чего дебютировал полнометражный фильм «Возвращение Бэтмена» (1992) с компьютерными копьями летучих мышей и армиями пингвинов, поведение которых было задано моделью Boids [15]. Данная модель использовалась в видеоигре 1998 года Half-Life для существ, похожих на птиц.

3.2 Реализация модели Boids в AnyLogic

При моделировании будем использовать подход агентного моделирования.

По условию агенты-птицы находятся на поверхность тора, поэтому тип пространства в AnyLogic будет непрерывный.

Взаимодействие агентов в агентных моделях может быть реализовано различными способами. Если связи между агентами достаточно постоянны, то агенту нужно запоминать тех агентов, которые состоят с ним в какой-то связи.

AnyLogic предоставляет встроенный механизм поддержки "плоских" связей, таких, как друзья, социальные контакты и т.д. Любой агент может иметь определенное число связей - ссылок на других агентов, обитающих в той же среде.

Взаимодействие агентов в агентных моделях может быть реализовано различными способами. Если связи между агентами достаточно постоянны, то агенту нужно запоминать тех агентов, которые состоят с ним в какой-то связи.

Связи в сети создаются отдельно вызовом функции applyNetwork(). Повторный вызов функции удаляет все существующие связи и создает новые соответственно текущим настройкам сети.

Создадим новый класс Main, который будет представлять среду для стаи птиц. Тип пространства выберем непрерывный, ширину и высоту среды установим 500 единиц. Радиус соединения установим 40, а действия перед выполнением шага – функция applyNetwork().

Теперь создадим новый класс, который будет представлять из себя агента-птицу, назовем его Bird.

Для данного класса определим такие параметры, как heading – текущее направление агента в радианах, x и y – текущее положение агента в двухмерном пространстве, crowded- минимальная дистанция между птицами и максимальный возможный поворот: max_turn= 0.05, max_turn_away = 0.02.

По условию все птицы перемещаются с одной и той же постоянной по модулю скоростью, поэтому в свойстве класса в разделе начальная скорость установим значение 2. Начальное значение текущего положения агента будут вычислять встроенные функции getX() и getY(), таким образом начальное положение агента задаем пользователь при расположении агентов в среде. Начальное значение направления агента heading будет выбрано случайным образом из промежутка (0…2π) heading = uniform() * 2 * Math.PI. Минимальную дистанцию между птицами установим 5 единиц.

Теперь создадим функцию move(), которая будет перемещать агента в новое положение. В теле функции напишем код:

```
x += getSpeed() * cos(heading);
y += getSpeed() * sin(heading);
// the 2D space is made a torus
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);
```

Функция getSpeed() – возвращает скорость агента, которая у всех птиц определяется из поля начальной скорости в классе среды Main.

Суть функции очень проста – сначала мы вычисляем новое положение агента, после чего проверяем, попадает ли это положение в границы среды, и если это не так, то корректируем его. После этого мы перемещает агента в новое положение с помощью встроенной функции setXYZ, и устанавливаем новые направление агента, которое будет корректироваться функцией adjust_heading().

В теле функции adjust_heading запишем код:

```
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 = ((Bird) a).heading;}
}
If (dist < crowded){
separate(closest_head);}
else{
align();
cohere();}
```

Данная функция обновляет текущее направление агента в соответствии со всеми правилами поведения. В функции сначала определяются ближайшие агенты,

если один из них слишком близко – разворачиваем агента-птицу. В противном случае выстраиваем и сближаем агентов в соответствии с правилами поведения. Здесь используются три функции, представляющие собой правила поведения птицы, описанные выше:

- Align - правило выстраивания
- Cohere - правило сближения
- Separate - правило разделения.

Сначала рассмотрим функцию align(), которая меняет направление птицы, выстраивая его с направлением соседних агентов. В теле функции напишем код:

```
double avg_head = 0, avg_x = cos(heading), avg_y = sin(heading);
for(Agent a : getConnections()){
avg_x += cos(((Bird)a).heading);
avg_y += sin(((Bird)a).heading);
}
avg_x /= getConnectionsNumber()+1;
avg_y /= getConnectionsNumber()+1;
avg_head = atan2(avg_y, avg_x);
turn_towards(avg_head);
```

Здесь мы перебираем всех агентов-птиц, которые находятся в радиусе видимости, вычисляем их среднее направление, после чего функцией turn_towards выравниваем направления с другими направлениями. Теперь определим функцию turn_towards(others_heading). В теле функции напишем код:

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

Данная функция выравниваем направления с другими направлениями, т.е. определяем разницу в направлениях и поворачиваем направление стаи.

Теперь определим функция cohere(), которая работает похожим способом: вычисляет среднее направление стаи и регулирует направление агента-птицы, чтобы тот летел в нужном направление.

```
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);
```

Отличием от функции align() является то, что она регулирует направление агента не по среднему направлению соседей, а в центр масс соседних агентов-птиц.

Опишем функцию separate(others_heading), в теле которой запишем код:

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

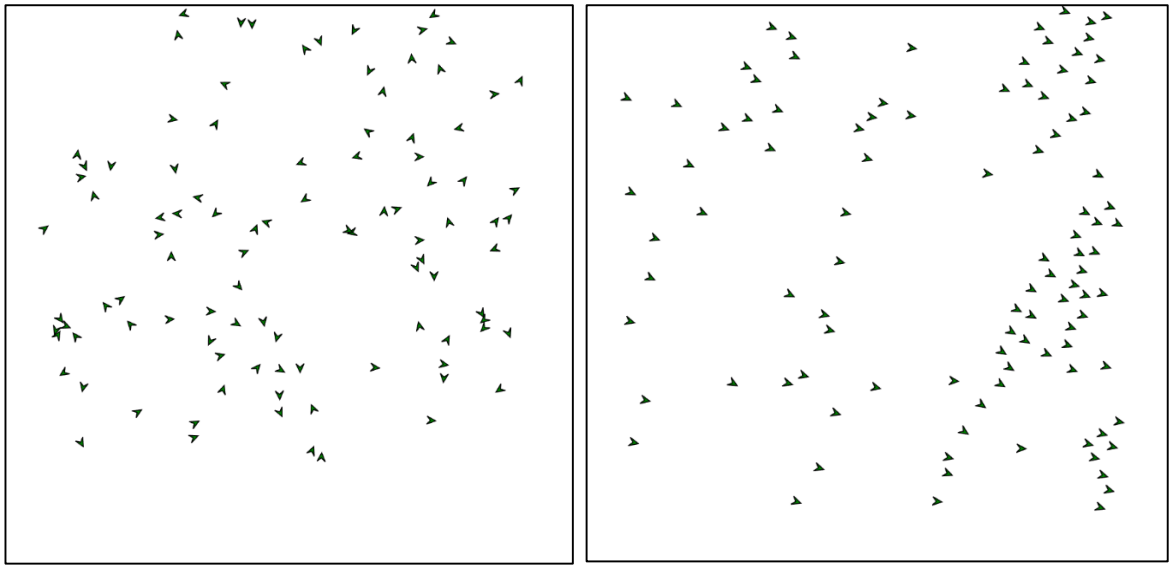
Функция subtract_headings определяет разницу между двумя направлениями и нормализует разницу от $-\pi$ до π. Функция separate разворачиваем агента от скопления птиц. В этой функции мы определяем разницу в направлениях и разворачиваем агента от скопления соседей.

Осталось только описать функцию subtract_headings:

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

Как видно функции, определяющие три правила поведения очень похожи и просты, однако этого достаточно для описания поведения сложной системы поведения стаи птиц.

При запуске модели все птицы случайно расположены в непрерывном пространстве со случайным направлением (см. рисунок 3.1.1), однако с течением времени они выстраиваются примерно в одном направлении (см. рисунок 3.1.2).



1) 2)

Рисунок 3.1 Положение агентов в модели Boids

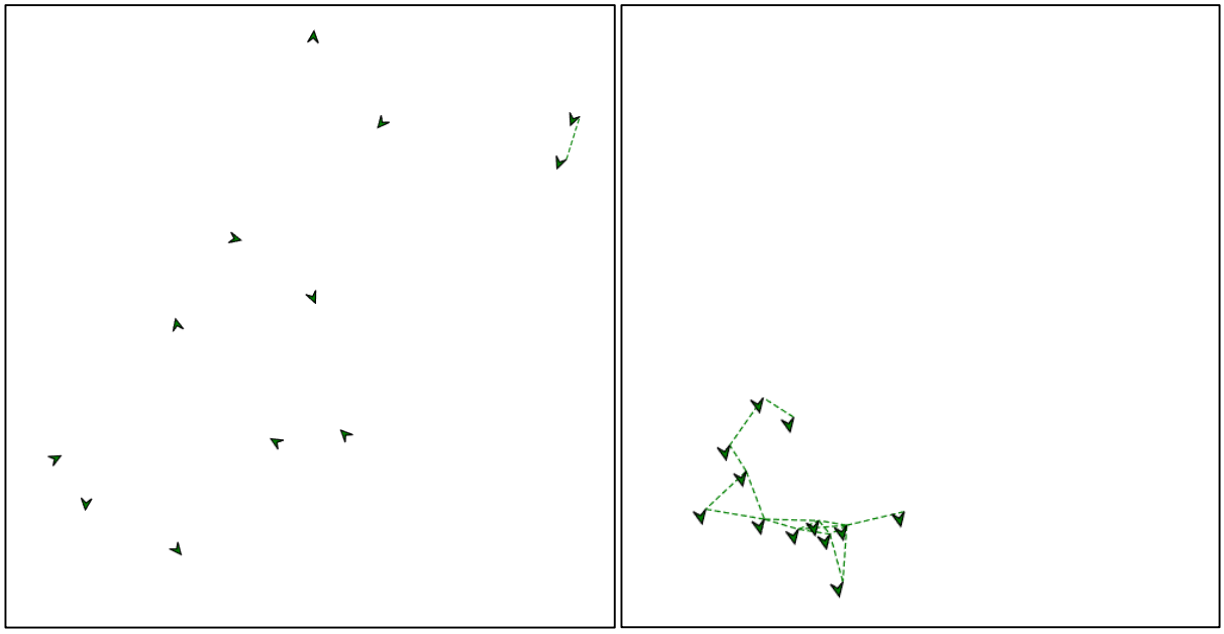
1) Начальное положение 2) Конечное положение

Наблюдая за конечным состоянием модели возникает вопросы: можно ли подкорректировать начальные правила поведения агентов так, чтобы, расположив птиц в одну цепочку, они продолжали двигаться как единое целое; можно ли подкорректировать правила поведения агентов, чтобы перейти от индивидуального поведения агентов к групповому.

3.3 Корректировка правил поведения модели Boids

Конечное состояние модели показывает, что агенты-птицы могут располагаться в одну цепочку и двигаться как одно целое. Возникший вопрос: можно ли подкорректировать базовые правила поведения модели так, чтобы, расположив агентов-птиц в одну цепочку, они продолжали двигаться как одно целое, будем решать практическим способом.

Для начала поочередно будет комбинировать три правила поведения агентов, чтобы осталось или одно, или два любых правила поведения. Комбинация правил таким образом делается для того, чтобы при каких-нибудь условиях зацепиться за желаемый результат системы. Данная практика ник чему не привела, конечное состояние агентов стало или полностью хаотичным движением, или движением в одной группе, при этом нарушая первоначальный строй (см. Рисунок 3.3).



1) 2)

Рисунок 3.3 Конечное состояние модели
1) Хаотичное движение агентов 2) Движение агентов как одно целое

Отталкиваться от движения агентов-птиц в одной группе не имеет смысла, т.к. результат поведения модели, приближенный к желаемому, больше допускается при начальных правилах поведения стаи птиц.

Поскольку в начале движения у каждой птицы случайное направление движение, из-за чего они успевают «разлететься», нарушая строй.

Зафиксируем несколько птиц для регулирования направления в начальный момент времени.

Введем параметр fix, который будет указывать на принадлежность к фиксированным (fix=1) или движущимся (fix=0) птицам. Теперь будем комбинировать начальное расположение фиксированных птиц в цепочке и правила перемещения (см. Рисунок 3.4). Данная практика опять ни к чему не привела: фиксированные птицы существенно не влияют на поведение модели.

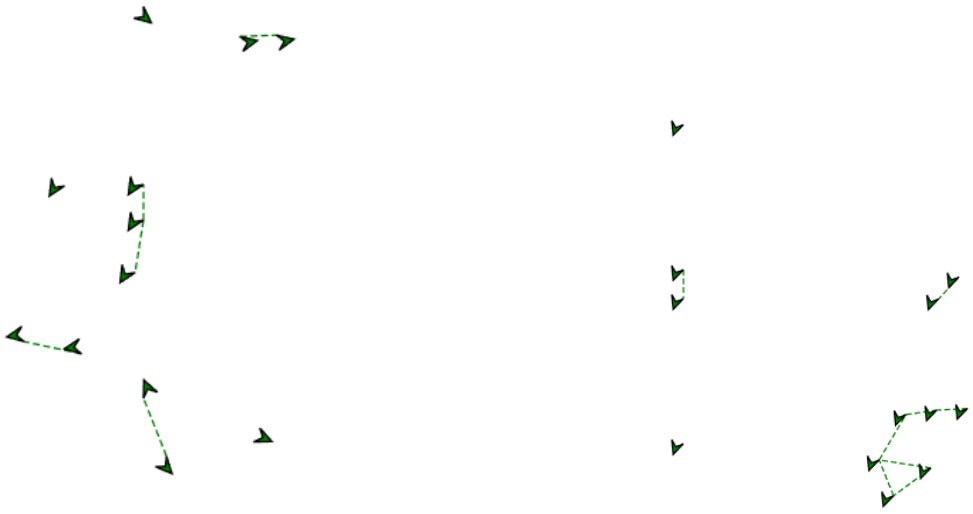


Рисунок 3.4 Состояние модели с фиксированными агентами

Проведя все вышеперечисленные модификации модели стаи птиц можно прийти к выводу, что правила поведения модели слишком простые, чтобы из можно было откорректировать для достижения поставленной цели.

Заключение

На основе проведения теоретических и экспериментальных исследований в работе можно сделать вывод, что агентное моделирование является наиболее перспективным методом исследования поведения сложных систем.

В данной курсовой работе была изучена, описана и реализована в среде AnyLogic агентная модель для случая локального взаимодействия агентов и дискретной среде: модель «муравья Лэнгтона», изучение и описана агентная модель «Sugarspace».

Изучена и реализована в среде AnyLogic модель для случая локального взаимодействия агентов в непрерывной среде на примере модели, имитирующей поведение стаи птиц. Также в работе приведены попытки модификации поведения модели стаи птиц с целью реализации заданного интегрального свойства системы через локальное поведение ее участников.

Список использованных источников

1. Щеглова Н.Л., Курс лекций по прикладному системному анализу
2. Понятие сложной системы | Электронная библиотека Librano.ru /6-2-ponyatie-slozhnoy-sistemy-teor_ekonom_inf_sys/
3. Ефимов И. Н. Е 912 Ефимов ИН, Морозов ЕА, Селиванов КМ Компьютерное моделирование динамических систем.-Ижевск. – 2014.
4. Справочная система Anylogic. [Электронный ресурс] – Режим доступа: https://www.anylogic.ru
5. Куприяшкин А. Г. Основы моделирования систем: учеб. пособие //Норильск: НИИ. – 2015.
6. Киселев И. Н. Агентное моделирование сердечно-сосудистых систем человека.
7. Справочная система Wikiwand. [Электронный ресурс] – Режим доступа: https://www.wikiwand.com/ru/Клеточный_автомат
8. Справочная система Neuronus. Клеточные автоматы. Часть III. Применение клеточных автоматов. [Электронный ресурс] – Режим доступа: http://neuronus.com/theory/ ca/654-kletochnye-avtomaty-chast-iii-primenenie-kletochnykh-avtomatov.html
9. Муравей Лэнгтона - Величайшие математические задачи - Иэн Стюарт. [Электронный ресурс] – Режим доступа: http://ogrik2.ru/b/ien-styuart/velichajshie-matematicheskie-zadachi/26401/muravej-lengtona/27
10. Справочная система knowledgr. Муравей Лэнгтона [Электронный ресурс] – Режим доступа: Ru.knowledgr.com
11. J. M. Epstein, R. Axtell, Growing artificial societies: social science from the bottom up, 1996
12. Epstein and Axtell's Sugarscape [Электронный ресурс] – Режим доступа: http://jasss.soc.surrey.ac.uk/12/1/6/appendixB/EpsteinAxtell1996.html
13. "Agents at Work". CIO Insight. 1 (27): 43. 1 June 2003. ISSN 1535-0096. Retrieved November 11, 2010.(Retrieved from ABI/Inform Document ID: 347271391)
14. Cui, Zhihua; Shi, Zhongzhi (2009). "Boid particle swarm optimisation". International Journal of Innovative Computing and Applications. 2 (2): 77–85. doi:10.1504/IJICA.2009.031778.
15. Lebar Bajec, Iztok; Heppner, Frank H. (2009). "Organized flight in birds"