МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**Реализация биоинспирированных алгоритмов для построения карты железнодорожных путей Краснодарского края***.*

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Е. Гиренко

(подпись)

Направление подготовки 02.03.02 — «Фундаментальная информатика и\_\_\_\_\_

(код, наименование)

информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_курс\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_

Направленность (профиль) \_\_\_\_\_\_\_\_\_\_\_\_Фундаментальная информатика и информационные технологии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Научный руководитель

канд. техн. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.А. Приходько

(подпись, дата)

Нормоконтролер

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Нигодин

(подпись, дата)

Краснодар

2022

**РЕФЕРАТ**

Курсовая работа 32 стр., 3 ч., 6 рис., 4 источников.

МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ, БИОИНСПИРИРОВАННЫЕ АЛГОРИТМЫ, ОПТИМАЛЬНАЯ МАРШРУТНАЯ КАРТА, SLIMEMOLD.

Цель работы – реализовать биоинспирированный алгоритм на основе модели роста слизевика Физариум и исследовать возможность построения с помощью его оптимальной маршрутной сети.

Объектом исследования в работе является алгоритм слизевика.

Предметом исследования является алгоритм слизевика Физариум в области построения оптимальных маршрутных сетей.

В качестве методов исследования использовались сравнительный анализ наблюдение, моделирование. В результате работы был изучена пригодность и условия эффективности алгоритма слизевика Физариум для построения оптимальных маршрутных сетей и разработана модель симуляции его работы над данной задачей.

Научная новизна работы заключается в исследовании возможности алгоритмизации задачи построения оптимальной маршрутной сети при помощи биоинспирированных алгоритмов.

По результатам исследования, были изучены необходимый перечень параметров для настройки алгоритма, возможности адаптации его под условия задачи, а также найдены дальнейшие способы его модернизации.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc121830458)

[1 Паттерны поведения слизевика Физариум 6](#_Toc121830459)

[1.1 Описание слизевика 6](#_Toc121830460)

[1.2 Прохождение лабиринта кратчайшим путём 7](#_Toc121830461)

[1.3 Исследования с картой Японии 8](#_Toc121830462)

[2 Построение модели 9](#_Toc121830463)

[2.1 Общее описание 9](#_Toc121830464)

[2.2 Представление агента и локации 9](#_Toc121830465)

[2.3 Цикл выполнения одного шага симуляции 11](#_Toc121830466)

[2.4 Адаптация модели для решаемой задачи 11](#_Toc121830467)

[3 Проведение тестирования реализованного алгоритма 14](#_Toc121830468)

[3.1 Проверка паттернов 14](#_Toc121830469)

[3.2 Моделирование сетей простой формы 16](#_Toc121830470)

[3.3 Результаты для карты ж/д путей Краснодарского края 18](#_Toc121830471)

[ЗАКЛЮЧЕНИЕ 21](#_Toc121830472)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc121830473)

# ВВЕДЕНИЕ

Задача построения маршрута является одной из самых часто встречаемых задач на графах. Из-за её фундаментальности с точки зрения транспорта она будет актуальна всегда. Несколько точек в пространстве, которые необходимо соединить так, чтобы по ним можно было перемещаться определённым образом. Однако эта задача крайне усложняется, когда в неё вносятся дополнительные условия из бизнеса. В нынешних задачах построения маршрутной сети необходимо учитывать экономичность построения, планарность, устойчивость и многие другие понятия, которые значительно усложняют алгоритмизацию. Примером тому является построение железнодорожных путей регионов, в которых все эти параметры играют важную роль. Если для данной задачи использовать алгоритм полного перебора, то его выполнение на сети с большим количество вершин станет слишком долгим и от того менее эффективным, чем использование труда профессионального проектировщика.

Для решения проблемы алгоритмической сложности можно применить другие алгоритмы. Строгие математические не могут быть определены для такого количества параметром. Одним из возможных решений является применение биоинспирированных алгоритмов, основанных на роевом поведении агентов. Такие алгоритмы не являются точными и требуют настройки множества параметров, однако имеют малую алгоритмическую сложность и легко могут быть переиспользованы при изменении условий. Возможность использования таких алгоритмов подтверждена природой, так что основной задачей становится перевод физических механизмов в цифровые.

Одним из примеров решения данной задачи в природе является слизевик Физариум многоглавый. Даже с отсутствующей нервной системой он способен находить кратчайший путь в лабиринте, а также строить оптимальные маршрутные сети, чтобы эффективно перемещать еду по своему телу. Именно его поведение было решено взять за основу для написания алгоритма построения оптимальной маршрутной сети. Для тестирования результатов алгоритма было решено взять карту железнодорожных путей Краснодаского края, чтобы в последствии получилось не только оценить нынешний уровень оптимальности, но и предложить более лучшее решение.

Основная цель работы – реализовать биоинспирированный алгоритм на основе модели роста слизевика Физариум и исследовать возможность построения с помощью его оптимальной маршрутной сети.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить паттерны поведения реального слизевика в данной задаче;
* построить на основе изученного материала симулируемую модель;
* реализовать данный алгоритм и проверить его работу на карте железнодорожных путей Краснодарского края;

Объектом исследования в работе является алгоритм слизевика.

Предметом исследования является алгоритм слизевика Физариум в области построения оптимальных маршрутных сетей.

Информационная база исследования включает в себя несколько видов учебных материалов таких как книги по алгоритму слизевика, статьи от авторитетных сервисов и документация. В качестве методов исследования использовались абстрагирование, наблюдение и моделирование.

Научная новизна работы заключается в том, что предлагается новое решение задачи построения оптимальной маршрутной сети при помощи биоинспирированного алгоритма слизевика Физариума многоглавого.

Практическая значимость исследования курсовой работы заключается в результатах исследования, которые пригодятся для дальнейшей работы над решением поставленной задачи и реализацией полного алгоритма по модели чёрного ящика.

# Паттерны поведения слизевика Физариум

## Описание слизевика

Физарум многоглавый живет в сырых местах, имеет ярко-желтую окраску и питается, переваривая бактерии, грибные споры и микробов. Вегетативное тело физарума представляет собой многоядерный протопласт, не имеющий клеточной оболочки. Такое образование называют синтицием, а тело организма – плазмодием. Он перемещается подобно гигантской амёбе, как бы перетекающей по поверхности. Слизневик использует так называемые «челночные перемещения». Его протоплазма постоянно перетекает сначала вперед, а потом назад. Один такой «двигательный» цикл занимает около двух минут и при этом скорость достигает 0,1—0,4 мм в минуту. Плазмодий активно перемещается в направлении источников пищи, более влажных мест и навстречу току воды, но избегает света. Движущие силы токов плазмы в плазмодии еще сравнительно мало изучены. Однако существует предположение, что движение связано с изменением вязкости специального белка — миксомиозина — при взаимодействии с АТФ.

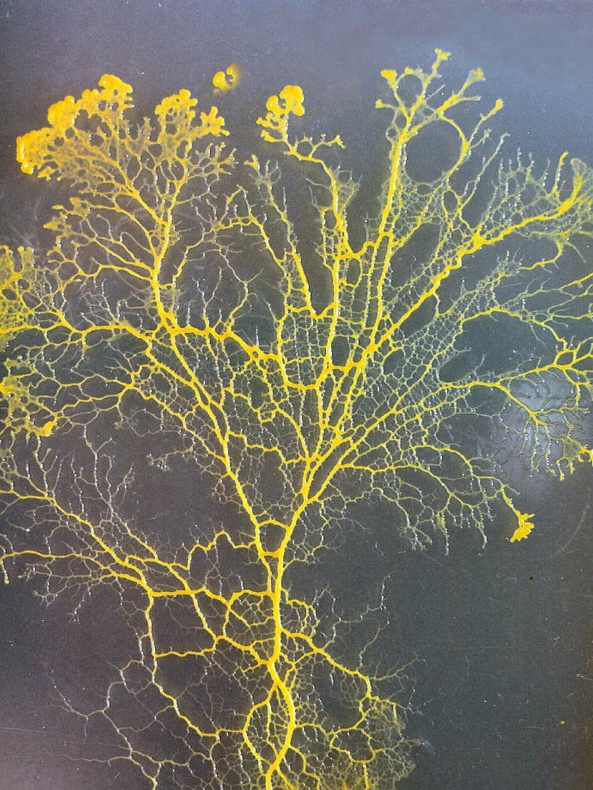


Рисунок 1 – Внешний вид физариума многоглавого

В прозрачном прикраевом слое цитоплазмы, свободном от органелл, с помощью электронного микроскопа были обнаружены чрезвычайно тонкие нити, находящиеся в непосредственном контакте с оболочкой. Было высказано предположение, что сокращение этих нитей также связано с токами цитоплазмы и движением плазмодия. Токи цитоплазмы в плазмодии можно непосредственно наблюдать под микроскопом. При этом в направлении движения у плазмодия возникают выросты, напоминающие псевдоподии простейших животных, и общий объем цитоплазмы всегда оказывается большим на переднем по движению конце плазмодия.

## Прохождение лабиринта кратчайшим путём

Плазмодии физарума многоглавого способны выбирать кратчайшее расстояние между источниками пищи. В экспериментах маленькие кусочки плазмодиев физарума помещали в лабиринт. Когда они заполняли всё пространство лабиринта, у входа и выхода помещали два блока с измельчёнными овсяными хлопьями. В течение четырёх часов цитоплазменные тяжи в тупиковых и более длинных ходах утончались и исчезали. Ещё через четыре часа плазмодий сформировал единственный утолщённый тяж по самому короткому пути между источниками пищи. Авторы работы сделали вывод о наличии у физарума примитивного интеллекта, хотя плазмодий обладает не более чем паттернами поведения. Тем не менее в некоторых случаях плазмодии выбирают более длинный путь, так как выбор пути происходит в один шаг, без просчитывания всех возможных решений. Кроме того, поведение миксомицета в лабиринте можно описать в терминах градиента пищевых сигналов.

## Исследования с картой Японии

Японцы взяли карту железнодорожного транспорта, на которой отметили города овсяными хлопьями, а самого слизевика поместили в районе Токио. Препятствия в виде водоёмов, рек и возвышенностей учёные обозначили источниками света, зная о том, что слизевики не любят яркого освещения. Фузариум добрался до всей пищи, составляя при этом самые кратчайшие маршруты. Ему хватило 26 часов, чтобы почти практически воссоздать существующую железнодорожную сеть. Пока неизвестно, будут ли использованы эти разработки, но Шнобелевскую премию учёные за них уже получили. К слову, эксперимент был повторен учёными Великобритании, Испании, Португалии, которые получили те же результаты.

Изображение выглядит как карта

Автоматически созданное описание

Рисунок 2 − Прохождение по карте Японии

Именно это исследование и вдохновило меня на создание модели данного организма для решения задачи построения оптимальной маршрутной карты.

# Построение модели

## 2.1 Общее описание

Как было видно в обзоре на слизевик, каждая его клетка действует по определённому алгоритму, на который влияют другие клетки. Причём в процессе роста принимают активно участие только те клетки, которые находятся на периферии, остальные же лишь поддерживают структуру слизевика. Это означает, что его можно представить в виде множества агентов, которые буду представлять собой именно активные клетки. В такой реализации один шаг алгоритма будет означать один шаг каждого из агентов.

Сам слизевик в процессе своего роста помечает то пространство, где он уже прорастал, чтобы туда более не расти. Кроме того, необходимо смоделировать процесс утолщения тех трубок, по которым еда доставляется наикратчайшим образом. Источники пищи, которые в эксперименте с Японией заменяли города, стимулировали слизевик расти к ним и создавать между ними сети. Так как агенты представляют собой активные клетки, то источники пищи станут генераторами агентов. Эти факторы позволят растущей модели строить лишь те пути, которые будут вести к вершинам, создавая сеть.

В данной работе рассматривается двумерная плоскость из-за специфики рассматриваемой задачи, однако достаточно легко можно перевести всех агентов и плоскость в трёхмерное пространство.

## Представление агента и локации

Агент представляет собой материальную точку, существующую в координатной плоскости и имеющую привязанные к её координатам три материальные точки – сканеры. Агент передвигает своё тело и поворачивает в зависимости от показателей сканеров, которые считывают информацию о следе с карты. След – это некое вещество, которое оставляет на своём месте в локации агент. Агенты хранятся в динамическом массиве ссылок на объекты класса. Каждый агент характеризуется параметрами:

* 1. Координатный вектор – в какой точке сейчас находится агент;
  2. Пиксельный вектор – вектор, указывающий в какой точке дискретной локации находится агент. Необходим для ускорения вычисления.
  3. Вектор движения – вектор, направление которого является лицевой частью агента, а длина – длина шага. То есть это вектор разницы между следующим положением и предыдущим.
  4. Векторы левого, центрального и правого сенсоров – векторы, обозначающие на сколько смещены сенсоры.

Локация является двумерной матрицей ограниченного размера, в клетках которых хранится след, оставляемый агентами. Чтобы определить клетку, в которой агент должен оставить след, его координаты проецируются на карту округлением своих координат до целочисленных значений индексов. Аналогично поступают и сенсоры при выборе клетки, из которой они считываю значение следа. Таким образом, чем больше агентов прошло по определённому маршруту, тем больше там будет следа, тем более вероятно, что другие агенты выберут именно этот маршрут. Чтобы неэффективные маршруты исчезали с карты след подвержен диффузии и разложению. Из-за разложения значение следа в каждой клетке уменьшается каждый шаг в определённое количество раз. Диффузия разделяет значение следа, находящегося в одной клетке, на восемь соседних клеток. Локации могут различаться размерами, возможностью допускать несколько агентов в одной клетке, а также зацикленностью координат.

Угол поворота агентов, угол поворота сенсоров, удалённость сенсоров от тела, скорость движения, количество следа за один шаг, размер диффузии, сила разложения, стартовая популяция настраиваются изначально в модели и после используются как в одном хранилище настроек. Алгоритм реализован так, что изменения в настройки можно внести между шагами активной симуляции.

## Цикл выполнения одного шага симуляции

Изначально определённым образом агенты появляются в координатной плоскости. Показатель следа в каждой клетке равен нулю. Каждый агент последовательно начинает выполнять свои действия.

Агент пытается двинуться вперёд на величину своего вектора движения. Если это удаётся, он оставляет в своём месте след, иначе он поворачивается в случайном направлении.

Следующим шагом агент активирует свои сенсоры, каждый из которых вычисляет значение следа в своей клетке. Если наибольшее значение показал левый сенсор, то агент поворачивается налево, аналогично с правым вектором. Если центральный вектор показал наибольшее значение, поворот не осуществляется. Для ускорения алгоритма поворот реализуется через умножение вектора движения и векторов сенсоров на матрицу поворота.

Рисунок 3 – Шаги алгоритма

После выполнения всех действий агентами динамический массив агентов перемешивается, чтобы поведение агентов не было зависимо от их индекса в массиве. После этого алгоритм повторяется вновь.

## Адаптация модели для решаемой задачи

Чтобы рассматриваемая модель решала задачу построения маршрутной сети необходимо добавить генераторы, время жизни и функцию смерти. Основные параметры остаются те же, кроме популяции на старте – она становится равной нулю.

Генераторы – массив точек в локации, которые генерируют агентов со случайным изначальным поворотом или с поворотом из очереди. Генераторы характеризуются координатами, очередью углов и количеством агентов, генерируемых за шаг. Очередь углов – это очередь, в которую при столкновении одного из агентов с другим генератором, добавляется угол, направленный в сторону другого генератора. Если очередь не пуста, то генератор первоначально берёт угол из очереди и создаёт агента с данным направлением, иначе он продолжает генерировать агентов со случайным поворотом. Необходимость этой очереди заключается в том, чтобы при нахождении агентами других генераторов эти агенты начнут появляться в более пригодном положении, чтобы вновь прийти в область того генератора. При построении карты генераторы имитируют вершины графа, причём генераторы каждый ход оставляют под собой след, равный частоте генерации агентов умноженный на след за шаг одного агента.

Агенты приобретают ещё одну настройку – время жизни. Этот параметр уменьшается на один каждый шаг. Когда он достигает нуля, агент в квадрате со стороной в пять клеток со своим телом по центру уменьшает след на количество следа в ход. Таких образом в области его смерти клетки могут приобрести даже отрицательный след. После этого агент удаляется и исчезает из массива.

Если же за свою жизнь агент дошёл до другого генератора, который не является его родным, то он исчезает из массива без активации функции смерти.

Это необходимо для того, чтобы наибольшее количество следа оставалось на путях, которые ведут к другим генераторам. Пути, которые ведут в тупик, через некоторое время приобретут отрицательный показатель следа и по ним более не будут ходить агенты. То есть агенты будут собой создавать рёбра между вершинами. Также функция диффузии в тестах была изменена на диффузию в виде креста и с меньшим коэффициентом.

С учётом этих изменений общий шаг алгоритма можно представить блок-схемой:

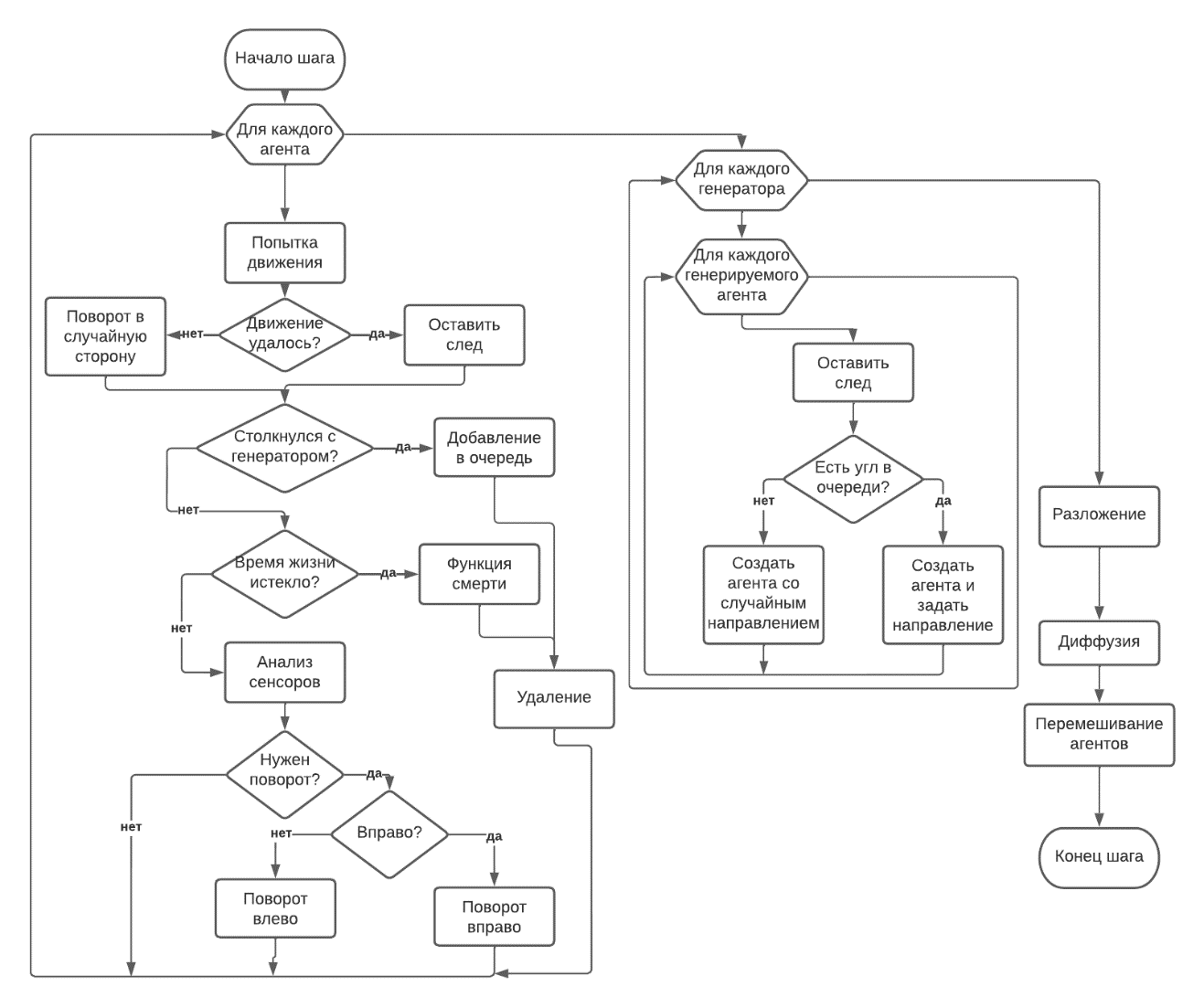


Рисунок 4 **−** Блок-схема шага адаптированного алгоритма

# Проведение тестирования реализованного алгоритма

## 3.1 Проверка паттернов

Поведение, описанное выше, является крайне простым. Каждый отдельно взятый агент способен видеть лишь область недалеко от себя и передвигаться к наибольшему значению по сенсорам. Сложное поведение, образование неких паттернов возможно только при большом количестве агентов, когда все они начинают действовать как одна система. Именно эти образующиеся паттерны было решено изучить, прежде чем начинать работу по построению маршрутной сети.

Модель обладает крайне большим количеством параметров и изменение каждого влечёт за собой изменения в поведении. Изначально появляющиеся случайно агенты способны как сгруппироваться в кучки или остаться в хаотичном состоянии, так и начать образовывать сложные структуры, похожие на сети.

Для визуального наблюдения каждые несколько шагов формируется снимок нынешнего положения агентов, относительно матрицы локации. Цвет агентов подбирается случайно. Яркость света означает количество агентов, которые находятся в одной точке. На рисунках представлены изображения и параметры симуляции для наиболее оптимальных параметров, выведенных в процессе тестирования.

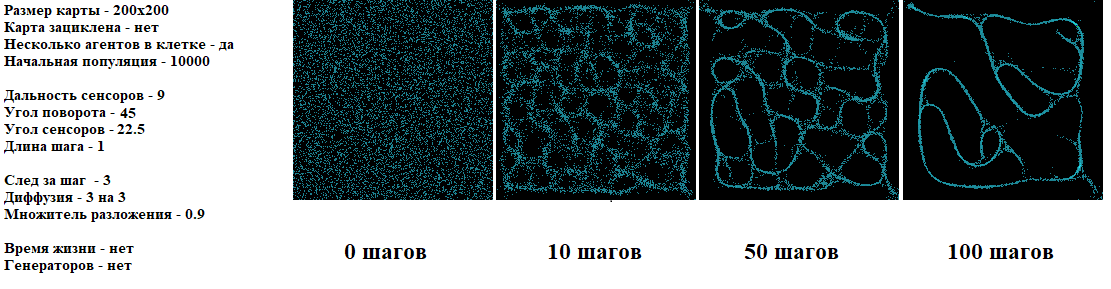


Рисунок 5 **−** Стандартная симуляция на замкнутой карте

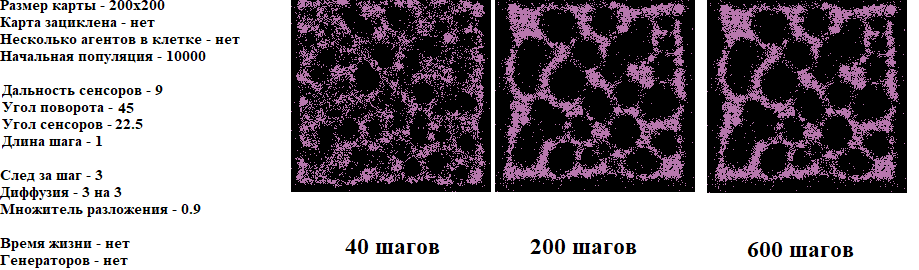


Рисунок 6 **−** Запрет находиться в одной клетке нескольким агентам

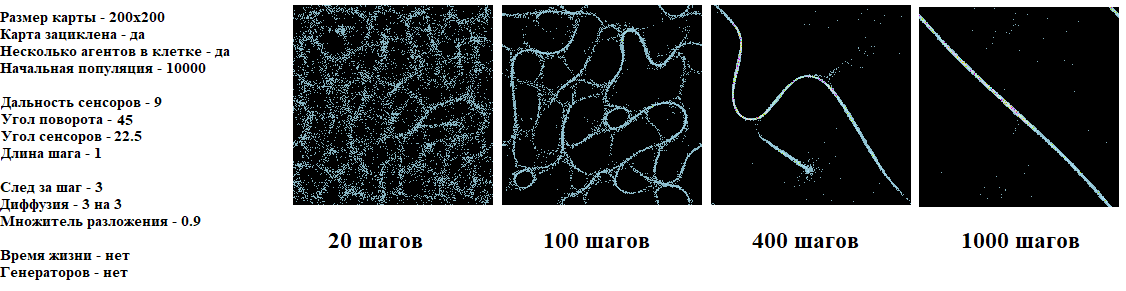


Рисунок 7 **−** Зацикленная карта

Следует заметить, что для образования сложных и правильных структур настраиваемые параметры не должны быть слишком завышенными или заниженными. Впрочем, зачастую интересно смотреть то, как на подобные ненормальные параметры реагирует модель.

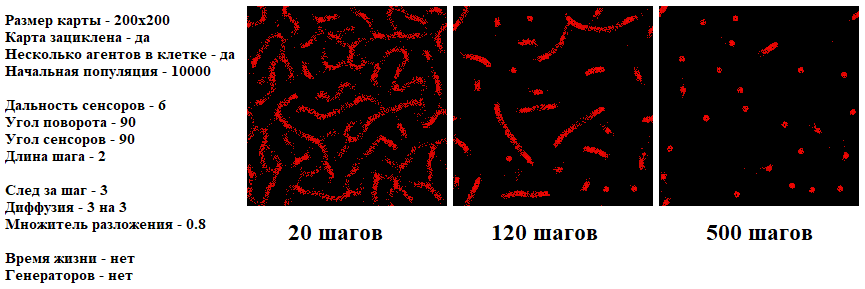


Рисунок 8 **−** Прямые углы сенсоров и поворота

Тестирование различных настроек показало, что поведение модели очень разнообразно и зависит от настраиваемых параметров. Вырисовывающиеся каждый раз сети уникальны и, если собрать изображения в gif или видео формат, достаточно интересны для любого наблюдателя. Исследование влияния каждого параметра на паттерны заслуживает отдельной научной работы. Для целей данной курсовой были найдены подходящие параметры для формирования сетей. Поиск оптимальных параметров для выполнения задачи построения маршрутной карты тоже заслуживает отдельного исследования.

## Моделирование сетей простой формы

Прежде чем переходить к реальной карте было принято решение протестировать несколько простых фигур, на которых достаточно легко определить наличие маршрутной сети. Главные особенности этих симуляций в том, что здесь имеются генераторы, агенты могут умереть со временем, форма диффузии изменена, а начальная популяция равна нулю.

В этих симуляциях требуется, чтобы случайно бредущие агенты по карте смогли находить другие генераторы, связываться с ними и после, когда уже агенты из своих тех построят ребро, соединяющее генераторы, сохранять целостность этих рёбер со временем.

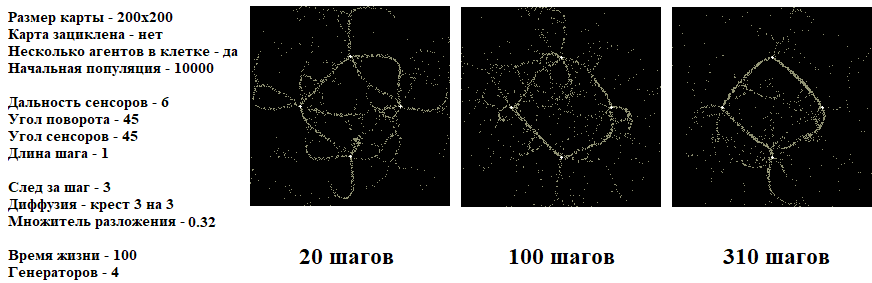


Рисунок 9 **−** Проверка построения сети в виде ромба

Во время тестов выяснилось, что на построение сети и крепкость связи крайне сильно влияет коэффициент разложения. Другие параметры тоже имеют свою силу, влияющую на поведение паттернов движения агентов однако для того, чтобы увидеть это влияние, требуется гораздо большие изменения, доходящие иногда до границ разумного. К примеру, дальность сенсоров не сильно влияет на построение результата, отчасти из-за того, что здесь должны формироваться обычные прямые линии.

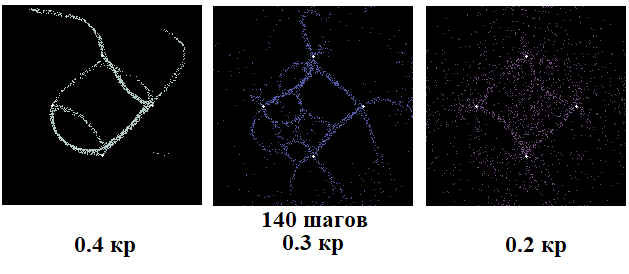


Рисунок 10 **−** Влияние коэффициента разложения

На приведённом выше рисунке все иные настройки, кроме коэффициента разложения одинаковы. Показан лишь один из шагов симуляций, но его достаточно для обзора, ибо в будущем структура сети разительно не менялась. Лишние линии из случая с коэффициентом в 0.4 со временем исчезали, а в симуляции с коэффициентом 0.2 они так и не смогли сформироваться. Более интересен случай с коэффициентом 0.3, в нём процесс формирования новых линий не останавливался за всё время симуляции, причём, появившиеся лишние рёбра через некоторое время сливались с более крупными рёбрами. Но это слияние не происходило безболезненно для структуры – появлялись сильные волнения в ней, что могло дойти до разрыва ребра. В те моменты, когда формировались лишь необходимые 4 ребра, структура держалась недолго и через некоторое время одно из рёбер разрывалось на несколько слабых потоков, которые собой разрушали целостность других рёбер.

При изменении структуры наблюдалось то же самое поведение. Дополнительно выяснилось, что высокий коэффициент разложения не всегда является эффективным по той причине, что формируемый граф в скором времени может стать деревом, что не всегда является оптимальным вариантом построения сети. Необходимо сохранять баланс этого параметра.

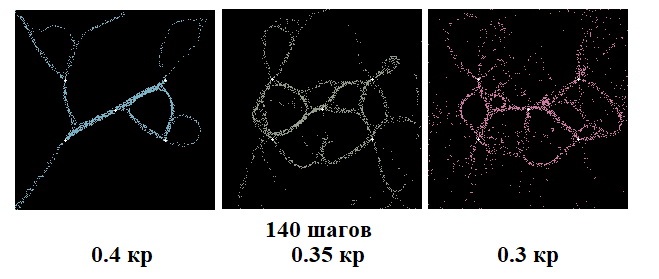


Рисунок 11 **−** Влияние коэффициента разложения при ином графе

## Результаты для карты ж/д путей Краснодарского края

Исходя из прошлых тестов было решено провести несколько экспериментов с разными коэффициентами разложения и выбрать наилучший полученный результат. В будущем планируется настроить программу, чтобы она сама выбирала самый оптимальный вариант из всех представленных с самостоятельным изменением параметров.

Карта железнодорожных путей Краснодара достаточно обширная и имеет множество станций на пути следования маршрутов, поэтому при переводе её в граф было решено брать в качестве вершин только либо большие населённые пункты, либо пункты, находящиеся на пересечении нескольких маршрутов, но при этом сохранялось некое минимальное расстояние между пунктами, чтобы не допустить близкого нахождения генераторов друг к другу.

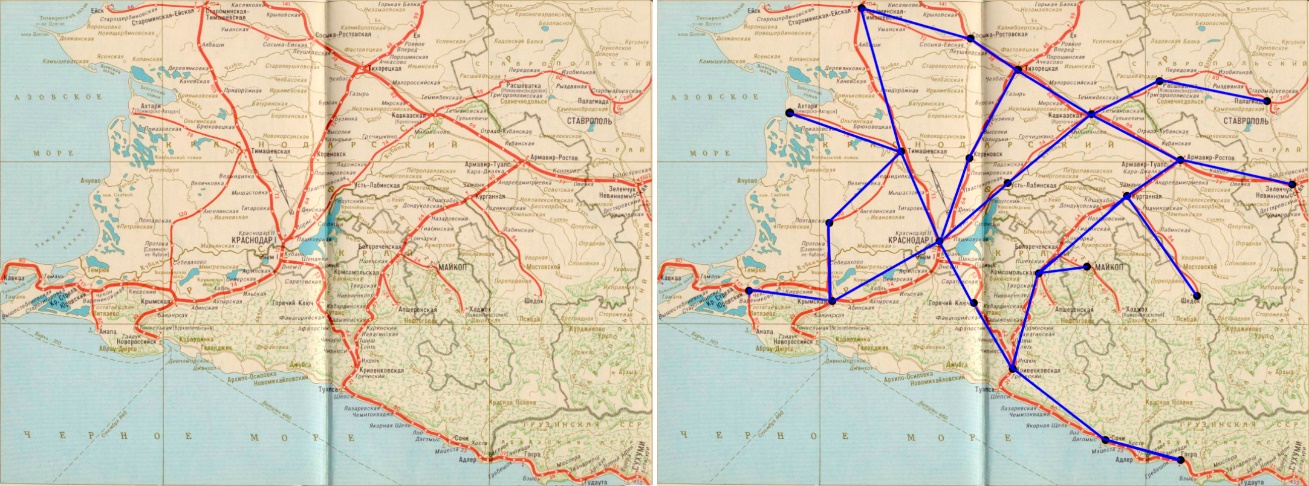


Рисунок 12 **−** Построение вершин по карте железнодорожных путей

Аналогично всем предыдущим тестам было решено перевести симуляцию в матрицу размером 200 на 200. Итоговое количество вершин – 24. Частота каждой вершины равна 2 агентам за шаг, но городу Краснодару было выставлено число 4. Также было замечено, что, несмотря на значительное увеличение вершин и количества агентов, один шаг замедлился всего лишь примерно в 2 раза.

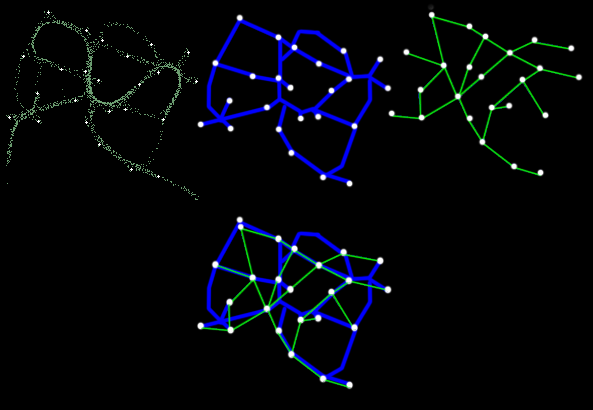


Рисунок 13 **−** Маршрутная сеть железнодорожных путей Краснодарского края

Данный результат был достигнут при 0.399 коэффициенте разложения. Первая фотография – получившееся изображение, выведенное программой, вторая – смоделированный граф по фотографии. На третьей фотографии ещё раз повторён реальный граф, а на последней для наглядности сравнения наложены полученный алгоритмом граф и реальный. Как можно заметить у агентов получилось сформировать связный граф с достаточной избыточностью рёбер, чтобы считать данную маршрутную сеть устойчивой. Причём даже более устойчивой, нежели реальная карта железнодорожных путей Краснодара. Если рассмотреть получившуюся карту, то можно увидеть, что при выключении любой вершины, кроме крайних, удобные для перемещения циклы в графах сохраняются. Например, если выключить вершину Краснодара в реальной карте, то граф станет деревом, что крайне не удобно.. Но также можно наблюдать проблемы, связанные с близким расположение некоторых генераторов друг к другу – агенты скапливаются в одну точку где-то посередине этих вершин. Кроме того, далеко не все рёбра являются прямыми между двумя точками, впрочем, со временем они должны стянуться в прямые линии. Однако стоит учитывать тот факт, что в симуляции не учитывались другие населённые пункты, особенности рельефа и непроходимые места. Также не получилось сделать так, чтобы через город Краснодар проходило как можно больше дорог. Но несмотря на все эти недочёты, получившаяся карта достаточно хорошо решает поставленную задачу построения оптимальной маршрутной карты. Если в будущем появится возможность перевести подбор необходимых параметров на программу, а не программиста, то её уже можно будет использовать для реальных задач.

# ЗАКЛЮЧЕНИЕ

В работе были изучены основные паттерны действий слизевика Физариум.

В работе были поставлены и решены следующие задачи:

* изучены паттерны поведения реального слизевика в задаче построения карту железнодорожного транспорта Краснодара;
* построена на основе изученного материала симулируемая модель;
* реализован алгоритм работы слизевика и протестирована его работа на карте железнодорожных путей Краснодарского края.

По результатам тестирования можно сказать, что алгоритм показал хорошие и удовлетворительные результаты по построению сетей, однако для использования его в профессиональной деятельности ему ещё далеко по причине сложности подбора оптимальных настроек симуляции. Впрочем, само существование данного алгоритма доказывает, что можно решать проблему построения оптимальной маршрутной сети без помощи алгоритмов полного перебора.

Для карты железнодорожных путей Краснодарского края алгоритм показал намного более устойчивое построение путей, намного меньшее количество висячих вершин, но и больше количество рёбер. Точно сказать, какая из получившихся сетей является лучшей, нельзя, покуда не будут выведена единая метрика для сравнения, но точно можно сказать, что полученная алгоритмом карта не сильно уступает реальной.

За время исследования было замечено, что алгоритм в достаточно большом количестве мест можно улучшить или дополнить, чтобы он лучше выполнял поставленную задачу. Также в сами настройки симуляции можно ввести ещё несколько объектов, таких как генераторы отрицательного следа, непроходимые участки и многое другое. Особенно будет интересно направление улучшения алгоритма в том, чтобы он самостоятельно подбирал оптимальные настройки для каждой карты вершин, а после и сам вырисовывал получившиеся рёбра, то есть полностью автоматизировать процесс построения оптимальной маршрутной карты и сделать из него модель чёрного ящика. Этим всем и, возможно, многим другим стоит заняться уже в следующей работе.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Moritz Physarum Slime Mold [Электронный ресурс]. – URL: https://entagma.com/physarum-slime-mold/ (дата обращения 02.11.2022).

2 Sage Jenson Physarum [Электронный ресурс]. – URL: https://cargocollective.com/sagejenson/physarum (дата обращения 07.11.2022).

3 Википедия Физариум многоглавый [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Физариум\_многоглавый (дата обращения 02.12.2022).

4 Jeff Jones Characteristics of Pattern Formation and Evolution in Approximations of Physarum Transport Networks [Электронный ресурс]. – URL: https://direct.mit.edu/artl/article-abstract/16/2/127/2650/Characteristics-of-Pattern-Formation-and-Evolution?redirectedFrom=fulltext (дата обращения 12.11.2022).

**ПРИЛОЖЕНИЕ   
Код программы**

SlimeMoldClass.h

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <math.h>

#include "omp.h"

#include <tuple>

#include <vector>

#include <stdio.h>

#include <algorithm>

#include <queue>

#include <string>

#include <random>

#include <iterator>

#include <iostream>

#include <fstream>

#include <chrono>

#pragma once

using namespace std;

typedef long long it;

typedef double ft;

const double PI = acos(-1.0);

class SlimeAgent;

class SlimeMoldSimulation;

class Location;

class SlimeAgent {

public:

//здесь настройки

SlimeMoldSimulation\* settings;

//вектор нынешней позиции

vector<ft> positionVector;

// нынеший занимаемый пиксель

vector<it> pixelVector;

//вектор, куда должно быть сделано следующее движение

vector<ft> moveVector;

//вектор, где находится сенсоры

vector<ft> leftSensorVector;

vector<ft> centerSensorVector;

vector<ft> rigthSensorVector;

//время жизни

it timeToLife;

//из какого генератора рождён

it teamIndex;

//Вектор направления при появлении

pair<ft ,vector<ft>> startAngle;

SlimeAgent();

void setUp(ft, it, it, vector<ft> , vector<ft>, vector<ft>, vector<ft>, vector<ft>, SlimeMoldSimulation\*);

//движение

void moveTurn();

//сканирование

void skanTurn();

//старение

it checkGeneratorsTurn();

//смерть

void deadTurn(it);

// функция хода

void makeFullTurn();

private:

template<typename T>

vector<T> vSum(vector<T>& x, vector<T>& y);

template<typename T>

vector<T> vMult(vector<T>& x, vector<vector<T>>& r);

// определяет направление поворота

it activateSensors();

// поворот векторов скорости и сенсоров

void rotate(bool isRigth);

//сдвиг и установка пикселя, если это возможно

bool move();

// нанесение следа на карту

void makeDeposit();

};

class Location {

public:

//здесь настройки

SlimeMoldSimulation\* settings;

vector<vector<ft>> trailMap;

vector<vector<bool>> agentMap;

Location();

Location(it x, it y, SlimeMoldSimulation\* set);

vector<it> getSizes();

void castDecay();

void castDiffusion();

bool canMakeMove(vector <it>& xy, vector <it>& oldxy);

vector<it> getPixelOnCoord(vector <ft>& xy);

bool checkMatrix(it i, it j);

private:

it xSize;

it ySize;

};

class SlimeMoldSimulation {

public:

Location location;

vector<SlimeAgent\*> particles;

vector<pair<vector<ft>, it>> generators;

vector<queue<ft>> generatorsQueue;

vector<vector<ft>> leftRotationMatrix;

vector<vector<ft>> rightRotationMatrix;

ft depositPerStep;

ft chanceOfRandomChangeDirection;

//используемые для создания

ft sensorAngle;

ft sensorOffsetDistance;

it sensorWidth; //пока автоединица, не используется

ft stepSize;

it population;

it startTimeToLife;

it diffusionSize; // пока не робит

ft decayFactor; //mult

bool isPeriodicBoundary;

bool isCanMultiAgent;

SlimeMoldSimulation();

void setLocation(it xSize, it ySize);

void setUp(it ttl, it sp, ft sod, it sw, ft sa, ft ra, ft ss, ft dps, ft corcd, it dif, ft dec, bool ipb, bool icma);

void startSimulation(vector<ft> startPosition);

SlimeAgent\* generateAgent(vector<ft> startPosition, ft startAngle, it);

private:

vector<SlimeAgent\*> generatePopulationInPixel(it count, vector<ft>& startPosition, it teamIndex);

vector<SlimeAgent\*> generatePopulationRandomPositions(it count, vector<it> sizes);

void makeStep();

void outputInConsoleTrayMap();

void outputInConsoleAgentMap();

void outputInFileAgentMap();

bool updateSettingsFromFile();

void outputInBmp(bool);

};

main.cpp

#include <SlimeMoldClass.h>

int main()

{

//ввод настроек теперь мерез файл settings в директории проекта

SlimeMoldSimulation sim = SlimeMoldSimulation();

sim.setLocation(200, 200);

sim.startSimulation({ 45.5, 45.5 });

}

Часть SlimeMoldClass.cpp

void SlimeMoldSimulation::startSimulation(vector<ft> startPosition) {

if (updateSettingsFromFile()) {

it count = 0;

std::random\_device rd;

std::mt19937 g(rd());

ft timeForOneIteration1, timeForOneIteration2;

ft sumTime = 0;

while (true) {

timeForOneIteration1 = omp\_get\_wtime();

makeStep(); // 450 // 340 - если параллельно

timeForOneIteration2 = omp\_get\_wtime();

count++;

timeForOneIteration1 = timeForOneIteration2 - timeForOneIteration1;

cout << count << endl << timeForOneIteration1 << endl;

sumTime += timeForOneIteration1;

cout << sumTime/count << endl << endl; // 12

shuffle(particles.begin(), particles.end(), g); // 3

if (count % 30 == 0) {

bool isUpdated = updateSettingsFromFile();

outputInBmp(isUpdated); // 130 - txt // 5 - bmp

}

}

}

}

void SlimeMoldSimulation::makeStep() {

int i;

#pragma omp parallel for

for (i = 0; i < particles.size(); i++) {

particles[i]->moveTurn();

}

vector<SlimeAgent\*> stillAlive;

#pragma omp parallel

{

vector<SlimeAgent\*> stillAlivePrivate;

#pragma omp for nowait // в конце цикла потоки не ждут друг друга, а идут дальше

for (i = 0; i < particles.size(); i++) {

particles[i]->timeToLife--;

it generatorIndex = particles[i]->checkGeneratorsTurn();

if (generatorIndex != -1 || particles[i]->timeToLife <=0) {

particles[i]->deadTurn(generatorIndex);

delete particles[i];

}

else {

stillAlivePrivate.push\_back(particles[i]);

}

}

#pragma omp critical // данная строка выполняется по очереди

stillAlive.insert(stillAlive.end(), stillAlivePrivate.begin(), stillAlivePrivate.end());

}

particles = stillAlive;

#pragma omp parallel for

for (i = 0; i < particles.size(); i++) {

particles[i]->skanTurn();

}

#pragma omp parallel

{

vector<SlimeAgent\*> newPart;

#pragma omp for nowait

for (i = 0; i < generators.size(); i++) {

newPart = generatePopulationInPixel(generators[i].second, generators[i].first, i);

location.trailMap[it(generators[i].first[0])][it(generators[i].first[1])]+=generators[i].second\*depositPerStep;

#pragma omp critical

particles.insert(particles.end(), newPart.begin(), newPart.end());

}

}

location.castDiffusion();

location.castDecay();

}