ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ имени первого президента России Б.Н.ЕЛЬЦИНА».

Институт математики и компьютерных наук
Очное отделение

-

-

Выпускная квалификационная работа

на тему:

«Визуализация и анимация плотностных алгоритмов кластеризации»

-

Допустить к защите Зав.кафедрой доктор физико-математических наук, профессор Волков М.В.	Выполнил: студент 2-го курса группы МГМТ-2 Соломатов А.С. Научный руководитель кандидат технических наук, доцент Авербух В.Л.

Екатеринбург
2017

РЕФЕРАТ

Соломатов А.С. ВИЗУАЛИЗАЦИЯ И АНИМАЦИЯ ПЛОТНОСТНЫХ АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ, выпускная квалификационная работа. Стр.61, рис.6, таб.4, лит.8, прил.5.

Объект исследования: анимация алгоритмов и восприятие визуальных символов человеком.

Цель работы: изучение данной области и разработка визуального представления работы плотностных алгоритмов.

В ходе работы исследована ситуация в области анимации и визуализации алгоритмов на данный момент, разработана модель представления данных, а также построено приложение, отвечающее за всю предлагаемую функциональность, на основе среды разработки «Unity».

ОГЛАВЛЕНИЕ

Введение....4
Постановка задачи....6
Изучаемые области....7
Algorithm animation....7
Описание....7
Компьютерная метафора....23
Data mining....8
Описание....23
Задача кластеризации....23
Алгоритм DBSCAN....24
Проектная часть....7
Предлагаемые метафоры....8
Сведения о выбранной метафоре....8
Описание доступного инструментария....8
Сведения о среде разработки....23
Основные необходимые элементы....24
Сборка проекта....25
Реализация....27
Архитектура программы....31
Вычисление промежуточных данных....31
Интерфейс взаимодействия с пользователем....36
Тестирование....39
Заключение....40
ЛИТЕРАТУРА....41
ПРИЛОЖЕНИЕ....41

Введение

На момент написания данной работы открыт широкий спектр кластеризирующих алгоритмов, большая часть из которых является оптимизацией предыдущих. Разумеется, каждый из них определен для решения узкого круга задач в таких областях как Data Science и математическая статистика, но для их успешного и быстрого изучения не всегда достаточно иметь перед глазами их описания - человеку всегда проще понять (“to grasp”) хорошо изображенный символ, нежели груду текста, описывающую его. Анимация алгоритмов как раз и занимается этим процессом.

В рамках текущей работы мы будем заниматься алгоритмами кластерного анализа, применяющимися в области аналитики данных. Появились данные алгоритмы в начале 30-х годов XX века применительно к области антропологии и психологии, хотя авторство термина “кластерный анализ” относят к математику Триону.

Изучение алгоритмов может быть нетривиальным и для понимания их сути необходимо сделать анимацию таких алгоритмов, чтобы начинающие могли уяснить его, а опытные люди имели возможность шире анализировать и улучшать качество ПО.

Анимация алгоритмов важна и нужна, и именно ей мы будем пользоваться. В дальнейшем в моей работе будут рассмотрены следующие проблемы и области:

1) для начала обсудим некоторые научные области, имеющие к нам отношение - анимацию алгоритмов и data mining;

2) затем постараемся подобрать выразительную метафору для анимации алгоритма;
3) и в конечном итоге приступим к описанию программной части работы и полученных результатов.

1. Постановка задачи

Необходимо:

- Выполнить и продемонстрировать анимацию плотностного алгоритма кластеризации данных
- Предоставить удобный интерфейс для управления данными и контроля за ходом выполнения алгоритма

Для реализации поставленных задач необходимо:

- Разработать метафору для выбранного алгоритма, отражающую его суть без искажения исходного принципа действия
- Провести анализ имеющихся сред графической разработки и выбрать подходящую
- Разработать и описать структуру программ визуализации

2. Изучаемые области

2.1. Algorithm animation

2.1.1. Описание

Анимация алгоритмов, как область, занимается динамическим представлением объектов на разных итерациях их развития, т.е. нацелено предоставить наблюдателю визуальный ряд, включающий в себя данные и операции над ними в абстрактном виде для облегчения понимания принципа работы.

Существует следующее определение:

“Анимацией алгоритмов называется процесс получения абстрактных представлений о данных программы, ее операциях и семантике и создания динамических графических образов этих абстракций. Анимация алгоритмов, в общем, включает в себя и анимацию программ, и визуализацию структур данных, которая обычно требует однозначного соответствия между данными программы и движущимися образами. Однако, анимация алгоритмов обычно понимается шире чем эти две области и дает образы программ, которые углубляют возможности простых графических представлений структур данных. Часто динамические образы алгоритмов, не имеют прямого соответствия с данными программы или с ее блоками, а представляют некоторые абстракции, созданные для объяснения семантики программы.”

Первые работы по анимации алгоритмов сделаны в первой половине 60-х годов, первые системы начали разрабатываться в конце 70-х - начале 80-х. Тогда казалось, что системы анимации алгоритмов решат проблемы, стоящие перед начинающими программистами и, главное, перед

преподавателями в такой дисциплине, как алгоритмизация. В то же время, реальные разработки часто ограничивались небольшим количеством алгоритмов, в которых легко описывались основные алгоритмические операции, в то время как для многих областей не существовало методов какого-либо представления новых алгоритмов.

Развитие области успешно идет последние десятилетия, и на данный момент разработано значительное количество реализаций анимации различных алгоритмов.

Процесс обучения построен на предоставлении структурированной информации и нацелен на как можно более быстрое усвоение материала обучающимся. Могут возникнуть осложнения: зритель может не иметь физической возможности изучать материал с помощью слухового контакта, или не иметь возможности читать. Algorithm animation нацелено упростить понимание такой информации посредством введения более привычных и легко усваиваемых сущностей и их взаимодействия.

При разработке алгоритмов нередко может возникнуть надобность в оптимизации или в сравнении текущего этапа с предыдущим, либо с конкурирующей работой. А т.к. алгоритм с большой вероятностью абстрактен и иногда требует довольно серьезных умственных усилий для представления, как от разработчика, так и от пользователя. В данном случае, удачно подобранная и разработанная визуализация и анимация версии алгоритма может пролить свет не только на то, что недоступно человеческому воображению, но указать на фундаментальные огрехи общей структуры последовательности действий.

Кроме того, раз любой программный продукт по определению снабжен алгоритмами, то применив к ним анимацию, можно гораздо проще взглянуть внутрь “серого ящика”, когда суть работы алгоритма затемняется его программной реализацией, и, пускай даже не найти огрехи, но оценить имеющееся по ряду признаков.

Также стоит разделять понятия «анимации алгоритмов» и «анимации программ». Так первые помогают вникнуть в суть алгоритма в отрыве от реализации, тогда как другие должны пролить свет на работу конкретной программы.

Ввиду вышесказанного несложно будет составить цели данной области.

Цели algorithm animation:

1) *Обучение*

Способствует изучению особенностей работы и устройства отдельных алгоритмов, обучению проектирования алгоритмов и программных систем на их основе.

2) *Оценка и измерение*

Помогает легче увидеть особенности работы программных систем с точки зрения основных алгоритмов их функционирования, а не с точки зрения данной реализации. Кроме этого анимация позволяет сравнить скорости выполнения и другие параметры алгоритмов.

3) *Отладка*

Помогает в отладке эффективности отдельных алгоритмов, базируясь на их оценках, позволяет выбирать лучший вариант алгоритма, необходимый для конкретной области применения.

В литературе указывается, что основные задачи при анимации алгоритмов можно описать следующим образом:

1. Определение (выбор) алгоритмических операций, описывающих алгоритм, но, в то же время, поддающихся анимации.
2. Выбор метафоры анимации.
3. Создание анимационных последовательностей для визуального моделирования этих операций.

Популярные работы по анимации алгоритмов начала 90-х сосредоточились на алгоритмах сортировки, в виду того, что в них четко определяются алгоритмические операции: сравнение и перестановка. Все реализации занимались исключительно анимацией операции перестановки, тогда как операция сравнения считалась самой очевидной,

однако существуют метафоры обеих операций алгоритма, что все-таки делает его очевидным.

При разработке анимации создаем идеализированное представление, пригодное для описания сути алгоритма. Затем продумываем основную идею того, как будет показываться сам процесс - метафору. После - сами события, которые нужно показать зрителю - алгоритмические операции. В последствии требуется разработать макет анимации - черновик, отображающий весь процесс, но жестко зашитый в рамках одного случая (case'a): процесс с логом происходящего. И в конечном итоге, принять решение в пользу одного из макетов и разработать программу, содержащую в себе, как саму анимацию, так и сигналы, уведомляющие о происходящих событиях.

Метафоры - это идеи, предназначенные для сценарирования фильма, который развернется по ходу работы системы; сценарий, описывающий работу алгоритма. Должен быть выбор «героев» (операций) и их действий, которые должны быть последовательны и информативны в плане описания алгоритма.

2.1.2. Компьютерная метафора

В данном случае метафора понимается как использование информации и знаний из одной (исходной) области человеческого опыта для того, чтобы лучше понять и структурировать явления и понятия другой (целевой) области, которая, как правило, является более абстрактной. Метафора делает возможным перенос понятий с уже освоенной предметной области на область неизвестного.

Метафора визуализации рассматривается как основная идеи сближения понятий прикладной области с той или иной образностью. Именно метафоры визуализации лежат в основе видов отображения, проектирование которых, в свою очередь, составляет базу проектирования когнитивной составляющей конкретной специализированной системы визуализации[1].

Пользователь системы имеет дело с конкретными графическими выводами (*displays*), которые получаются, когда виды отображения (*views*) “наполняются” реальными данными. Графические выводы являются воплощением абстрактного понятия вида отображения. Смена значимых и значащих картинок (графических выводов) при возможном взаимодействии с изображением является внешней стороной визуализации. Задача проектировщика - разработать набор видов отображения, обладающий такими свойствами, как системность и иерархичность. Идея сближения понятий, лежащая в их основе (метафора), может не быть явно описана в процессе проектирования, но, так или иначе, присутствует в сознании проектировщика.

1 Semiotic way on generation of a computer visualization theory

Целью метафоры визуализации является создание наглядных зрительных образов, манипуляции с которыми как-то соответствуют операциям над модельными объектами. Составляющими метафоры визуализации являются ее образность и предписываемые ею действия, как по изменению визуальных образов, так и по манипуляциям пользователей с визуальными объектами. Метафора задает контекст, помогающий правильной интерпретации элементов данного языка визуализации, выявлению значения визуального текста.

Разработка метафоры – эмпирический процесс и опирается на личный опыт исследователя – и критика представления будет исходить из тех же побуждений.

Таким образом, метафора визуализации обеспечивает понимание отображаемых сущностей прикладной области, а также участвует в создании новых сущностей на базе внутренней логики самой метафоры.

Разумеется, не стоит списывать со счетов мысль, что плохо выбранная метафора может, напротив, усложнить или даже разрушить исходное представление алгоритма. Так не стоит отображать на экране слишком много сущностей и не следует давить на зрителя огромным количеством их взаимодействий. В идеале достаточно определить пару-тройку простых, желательно абстрактных, типов объектов и концентрировать внимание наблюдателя на, пускай и не слишком конкретных, но интуитивно понятных действиях.

2.2. Data mining

2.2.1. Описание

Данная область занимается изучением процессов определения схожестей на больших наборах данных. Включает в себя такие подобласти, как Artificial Intelligence (Искусственный интеллект), Machine learning (Машинное обучение) и статистику. Появилась в конце 80х – начале 90х годов после семинара Григория Пятецкого-Шапиро, который интересовался вопросом: можно ли автоматически находить определённые правила, чтобы ускорить некоторые запросы к крупным базам данных. Тогда же был предложен термин — Data Mining, в который заложена метафора «добычи данных», отображающая суть области.

Большие наборы данных без обработки, как правило, бесполезны, поэтому люди стали искать возможности их структурировать или модернизировать по определенным алгоритмам, которые и являются основными инструментами Data mining.

Преподается данная область, как правило, на спецкурсах в университетах и, в большинстве случаев, на картинках, либо сплошным текстом оригинальных статей и их переводов. В то время как большими данными в наши дни оперируют все крупные компании мира, содержащие информацию о своих клиентах, больных, плательщиках и пр.

Задача Data mining - [желательно автоматический] анализ большого объема данных с целью получить новую, доселе неизвестную информацию, включающую в себя паттерны; группы данных, объединенных определенными признаками (кластерный анализ - cluster analysis); зависимости (последовательный анализ паттернов - sequential

pattern mining) или аномалии. В общих чертах, главная цель области - извлечь и показать информацию в понятном для последующего восприятия виде.

Основные задачи Data mining можно разбить на 6 групп:

1. *Anomaly detection* - определение специфических, выделяющихся среди других, данных, имеющих потенциальную возможность определить ошибки в алгоритмах, хранилищах и т.д.
2. *Dependency modeling* – поиск связей (зависимостей) между данными.

   Н-р: сбор информации по тематикам роликов, которые просматривают пользователи YouTube – пытаемся доказать предположение, как вариант, если пользователь любит смотреть политические видео, то ему придутся по вкусу и передачи про путешествия
3. *Clustering* (кластеризация) – поиск схожих между собой данных и объединением их в группы (кластеры) по заранее заданным признакам.
   Н-р: определение принадлежности людей к определенным народностям по географическому положению (мало правдоподобно, но имеет право на существование)
4. *Classification* (классификация) – обобщение текущих данных для применения к новым (последующим) входам.
   Н-р: классификация биологических показателей допрашиваемых для последующих заключений об искренности людей, дававших показания
5. *Regression* (регрессия) – поиск функции, определяющей выборку с наименьшей погрешностью.
6. *Summarization* (обобщение) – предоставление компактной визуализации или отчета о входных данных.

Методов и терминов в данной области, также, немало. Наиболее популярные из них:

1. Neural networks (Нейронные сети) – в сущности, объекты, состоящие из частиц (нейронов) и связей между ними. Могут содержать множество слоев и пропускать через себя многочисленные данные с целью построения определенного вывода о входных данных. В настоящее время очень широко распространены.
2. Regression analysis (Регрессионный анализ) – метод определения влияния набора переменных на одну другую.
3. Cluster analysis (Кластерный анализ) – (см. *кластеризация*)
4. Decision trees (Деревья решений) – использует дерево для отображения решений и их возможных последствий, включая шанс исхода, и ресурсозатратность.
5. Genetic algorithms (Генетические алгоритмы) – в основном используются для вывода решений оптимизации поиска, полагаясь на биологические операции, как мутация, переход и селекция.

2.2.2. Задача кластеризации

Задача кластеризации - создать разбиение имеющегося множества на подмножества таким образом, чтобы элементы одного подмножества существенно отличались от элементов других подмножеств по определенному ряду признаков, передаваемых на вход. Относится к ряду задач обучения без учителя.

Поставить ее можно следующим образом:

Пусть X — множество объектов, Y — множество номеров (имён, меток) кластеров. Задана функция расстояния между объектами $\rho(x,x')$. Имеется конечная обучающая выборка объектов $X^m = \{x_1,\dots,x_m\} \subset X$. Требуется разбить выборку на непересекающиеся подмножества, называемые ***кластерами***, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ, а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X^m$ приписывается номер кластера y_i.

Алгоритм кластеризации — это функция $a\colon X \to Y$, которая любому объекту $x \in X$ ставит в соответствие номер кластера $y \in Y$. Множество Y в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного *критерия качества* кластеризации.

Кластеризация отличается от классификации тем, что метки исходных объектов y_i изначально не заданы, и даже может быть неизвестно само множество Y.

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин:

- Не существует однозначно наилучшего критерия качества кластеризации. Все они могут давать разные результаты.
- Число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым критерием.
- Результат кластеризации существенно зависит от субъективно выбираемой метрики.

Цели кластеризации:

- Обеспечить понимание данных путём выявления кластерной структуры. Разбиение выборки на группы схожих объектов позволяет упростить дальнейшую обработку данных и принятия решений, применяя к каждому кластеру свой метод анализа.
- Сжатие данных. Если исходная выборка избыточно большая, то можно сократить её, оставив по одному наиболее типичному представителю от каждого кластера.
- Обнаружение аномалий. Выделяются нетипичные объекты, которые не удаётся присоединить ни к одному из кластеров.

Алгоритмы кластеризации можно разделить на несколько **категорий**:

1. Алгоритмы разбиения
   - Разбивают элементы по максимальному сходству элементов - в общем случае используют некоторый управляющий элемент не из выборки
   - Наиболее используемые: K-means, K-medoids, CLARANS

2. Иерархические
   - Выполняют последовательную иерархическую декомпозицию, разделяя построенное дерево данных на N подмножеств до тех пора, пока каждое подмножество не будет состоять из 1 элемента
   - Н-р: BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), CHAMELEON

3. Плотностные
   - Основаны на предположении, что плотность элементов внутри кластера больше, чем снаружи
   - Н-р: DBSCAN, SUBCLU, OPTICS

4. Сетевые
   - Пространство объектов разбивается на конечное число ячеек, образующих сетевую структуру, в рамках которой выполняются все операции кластеризации - не зависят от кол-ва объектов данных

- Н-р: CLIQUE, MAFIA

5. Модельные
   - Предполагают, что имеется некоторая математическая модель кластера в пространстве данных и стремятся максимизировать сходство этой модели и имеющихся данных
   - Н-р: EM (Expectation-maximization) algorithm

6. Концептуальные
   - Определяют кластеры как группы объектов, относящейся к одному классу или концепту, определенному набору пар атрибут-значение
   - Н-р: COBWEB

Данные алгоритмы не элементарно понять в виду того, что большинство из них является модернизацией (зачастую нетривиальной) предыдущих, особенно для дилетантов в этой области, на которых мы и ориентированы. Но, вместе с тем, они широко используются в анализе данных, и, считаю, они необходимы к изучению интересующимся[2].

2 Data mining - Wiki

2.2.3. Алгоритм DBSCAN

В процессе исследования был выбран плотностный алгоритм **DBSCAN** (A Density-Based Algorithm for Discovering Clusters), предложенный в 1996-м году в совместной работе 4-х докторов наук Martin Ester, Hans-Peter Kriegel, Joerg Sander и Xiaowei Xu института компьютерных наук Мюнхенского Университета.

Был выбран именно DBSCAN, так как он является основой для других, уже разработанных, плотностных алгоритмов (н-р SUBCLU), и является необходимым для понимания сути его модернизаций.

DBSCAN был разработан с целью:

a) уменьшить сложность выбора входных параметров
b) увеличить эффективность при определении кластеров на больших пространственных базах данных
c) позволить определять кластеры случайной формы

За одно, в итоге оказалось, что алгоритм в десятки раз эффективнее уже ранее созданного и широко используемого алгоритма CLARANS (Clustering Large Applications based on RANdomized Search).

Пространственные БД (SDBS – Spatial Database Systems) – это базы данных, оптимизированные для хранения и запросов данных, представляющих геометрические объекты. Обычно, такие БД хранят в себе такие объекты, как линии, точки, полигоны. Но есть и БД, являющиеся хранилищем для более сложных 3-мерных моделей, триангулярных изометрических сеток и пр.

CLARANS относится к стандартным разбиенческим алгоритмам, в основе которого лежит алгоритм K-medoids, однако его модернизация быстро работает на тысячах объектов. Но т.к. авторы предлагали вызывать CLARANS для каждого K от 2 до N по разу, то и скорость выполнения сильно зависит от большого N, т.к. имеет ввиду O(n) вызовов всего алгоритма. Кроме того, из явных недостатков стоит выделить необходимость хранения всех объектов в памяти во время выполнения, что увеличивает время обработки - так при увеличении количества данных скорость отработки алгоритма падает экспоненциально, когда у DBSCAN стабильно линейна.

Суть же DBSCAN в предположении, что внутри кластера плотность данных выше, чем вне него; а также, что в зонах шума плотность ниже, чем в любом из кластеров. Т.е. основная идея в том, для каждой точки в окрестности определенного заранее радиуса должно содержаться хотя бы минимальное количество точек, суть плотность в окрестности точки должна быть равна или выше определенного порогового значения. Форма

же определяется выбором функции d(p,q), определяющей расстояние

между двумя точками p и q. Н-р, в случае 2-мерного пространства, форма будет прямоугольная.

Описание:

При имеющемся наборе данных V, радиусе поиска R и заранее заданным числом N:

1. Задаем счетчик C - номер текущего кластера

2. $\forall v0 \in V$ выполняется:

a. Если в окрестности R вершины v0 cv >= N, где cv - кол-во

вершин $v \in V$, все вершины в данной окрестности присваиваем кластеру, иначе начинаем поиск заново для следующей вершины

b. Если поиск удался и вершины вошли в кластер, продолжаем поиск от вершин, принадлежащих данному кластеру до тех пор, пока поиск не провалится - в этом случае счетчик увеличиваем на единицу

В оригинальной статье[3], которую я взял и перевел, алгоритм описывается следующим образом (переведен):

DBSCAN(*МнвоТочек, Eps, МинТчк*)
НАЧ
    ИдКластера := СледИд();
    ПОКА *i* < *МнвоТочек*.длина НЦ
        Точка := *МнвоТочек*.взять(*i*);
        ЕСЛИ *Точка*.ИдКластера = НЕТ_КЛАСТЕРА ТО
            ЕСЛИ ***Кластеризировать***(*МнвоТочек, Точка,*
                ИдКластера, Eps, МинТчк) ТО
                ИдКластера := СледИд(*ИдКластера*);
        i := *i* + 1;

3 A Density-Based Spatial Clustering of Application with Noise (Henrik Bäcklund, Anders Hedblom, Niklas Neijman)

КЦ

КОН

МнвоТочек - это либо база данных, либо или кластер, обнаруженный при предыдущем запуске алгоритма. *Eps* и *МинТчк* являются глобальными параметрами, отражающими плотность: первый - радиус окрестности поиска, второй - минимальной количество точек, необходимое для образования кластера. Метод *МнвоТочек*.взять(*i*) возвращает *i*-й элемент из *МнвоТочек*. Наибольшая роль отведена функции *Кластеризировать*, которая изложена ниже:

Кластеризировать(*МнвоТочек, Точка, ИдКластера, Eps, МинТчк*) : Boolean
НАЧ
ИсслТочки := *МнвоТочек*.запросОбласти(*Точки, Eps*);
ЕСЛИ *ИсслТочки*.длина < *МинТчк* ТО НАЧ
МнвоТочек.изменитьИдКластера(*Точка*, ШУМ);
ВОЗВРАТ Ложь;
КОН
ИНАЧЕ НАЧ
МнвоТочек.изменитьИдКластера(*ИсслТочки, ИдКластера*);
МнвоТочек.удалить(*Точка*);
ПОКА *ИсслТочки* != Пусто НЦ
ТекущТчк := *ИсслТочки*.Первый();
Результат := *МнвоТочек*.запросОбласти(*ТекущТчк, Eps*);
ЕСЛИ *Результат*.длина >= *МинТчк* ТО
ПОКА i < *Результат*.длина) НЦ
РезультатТчк := *Результат*.взять(i);
ЕСЛИ *РезультатТчк*.ИдКластера
ВХОДИТ В { НЕТ_КЛАСТЕРА,
ШУМ } ТО НАЧ

```
                                    ЕСЛИ РезультатТчк.ИдКластера =
                                НЕТ_КЛАСТЕРА ТО
                                    ИсслТочки.добавить(РезультатТчк);
                                МнвоТочек.измИдКластера(РезультатТчк,
ИдКластера);
                                КОН
                                i := i + 1;
                            КЦ
                        ИсслТочки.удалить(ТекущТчк);
                КЦ
                ВОЗВРАТ Истина;
        КОН
КОН
```

Вызов метода *МнвоТочек*.запросОбласти(*Точка*, *Eps*) возвращает список точек, попадающих в *Eps*-окрестность *Точки*. Запросы области могут использовать пространственные методы, такие как R-деревья, предполагающие эффективную обработку нескольких типов

пространственных задач. Высота R-дерева - O(log n) для базы данных из

n-точек в худшем случае, и запрос с “малой” областью должен пройтись только ограниченное количество путей, *m* в дереве.

Так как точек в Eps-окрестности ожидается немного на фоне всего множества данных, сложность выполнения запроса для одной области -

O(log n). Для каждой точки мы имеем, по меньшей мере, один запрос

области. Отсюда, сложность алгоритма DBSCAN -O(n * log n).

Поле *ИдКластера* точек, помеченных как ШУМ, могут быть изменены в ходе работы алгоритма, если они достижимы для других точек по плотности, что происходит для граничных точек кластера. Эти точки не добавляются в *ИсслТочки*, так как нам уже известно, что точка с *ИдКластера* = ШУМ не является точкой-источником поиска.

Может произойти ситуация, когда точка стоит между двумя

кластерами C1иC2, находящимися близко друг к другу. Тогда эта точка

способна быть граничной для обоих кластеров, в противном случае оба кластера будут равны. В этом случае такая точка присваивается первому кластеру. Исключая этот случай, результат DBSCAN не зависим от порядка точек в данных.

Преимущества:

1. Не нуждается в выборе заранее предугаданного кол-ва кластеров на выходе
2. Удобен в работе с базами данных (для чего и был создан)
3. Безразличен к форме кластера на выходе (может получится, что угодно)
4. Устойчив к выбросам
5. Если данные хорошо изучены, задать параметры R и N не составляет большого труда

Недостатки:

1. Чувствителен к плотности данных (если в большинстве случаев недостижимы - алгоритм бесполезен)
2. Сложен в использовании многомерных данных
3. Усложняет задачу визуализации при использовании многомерных данных

3. Проектная часть

3.1. Предлагаемые метафоры

В процессе работы опытным путем было разработано несколько метафор анимации. Для алгоритма DBSCAN были предложены:

Наводнение

Суть: для заданного набора двумерных данных, на каждой итерации поиска во время применения алгоритма проводятся следующие процедуры:

1. Пусть есть Rb,Rv,N ϵ N, где Rb- радиус поиска, Rv- максимальный радиус вершины, N- минимальное кол-во вершин, необходимое для включения в кластер
2. Корневую вершину v0(вокруг которой ищутся близлежащие) считаем водяным источником
3. Остальные вершины считаются просто предметами, легко впитывающими воду (н-р пробки)

4. Вокруг источника размещаем барьер радиуса Rb

5. Активируем источник и заполняем размеченную зону водой - вследствие содеянного, близлежащие вершины (предметы) разбухают до заранее вычисленного размера (с радиусом Rv), минимально необходимого для “закупорки” источника v0

6. Разбухшие вершины выталкиваются со своих мест и стремятся образовать вокруг источника плотный барьер, не позволяющий протекать воде

7. Спускаем воду через дырки, в которых стояли все вершины, кроме v0

8. Если вода вокруг источника есть, считаем, что поиск прошел удачно, меняем цвет всех вершин в радиусе Rbи полагаем их объединенными в кластер
9. Откатываем разбухшие вершины на место и возвращаем им исходный размер

(см. рис. 1)

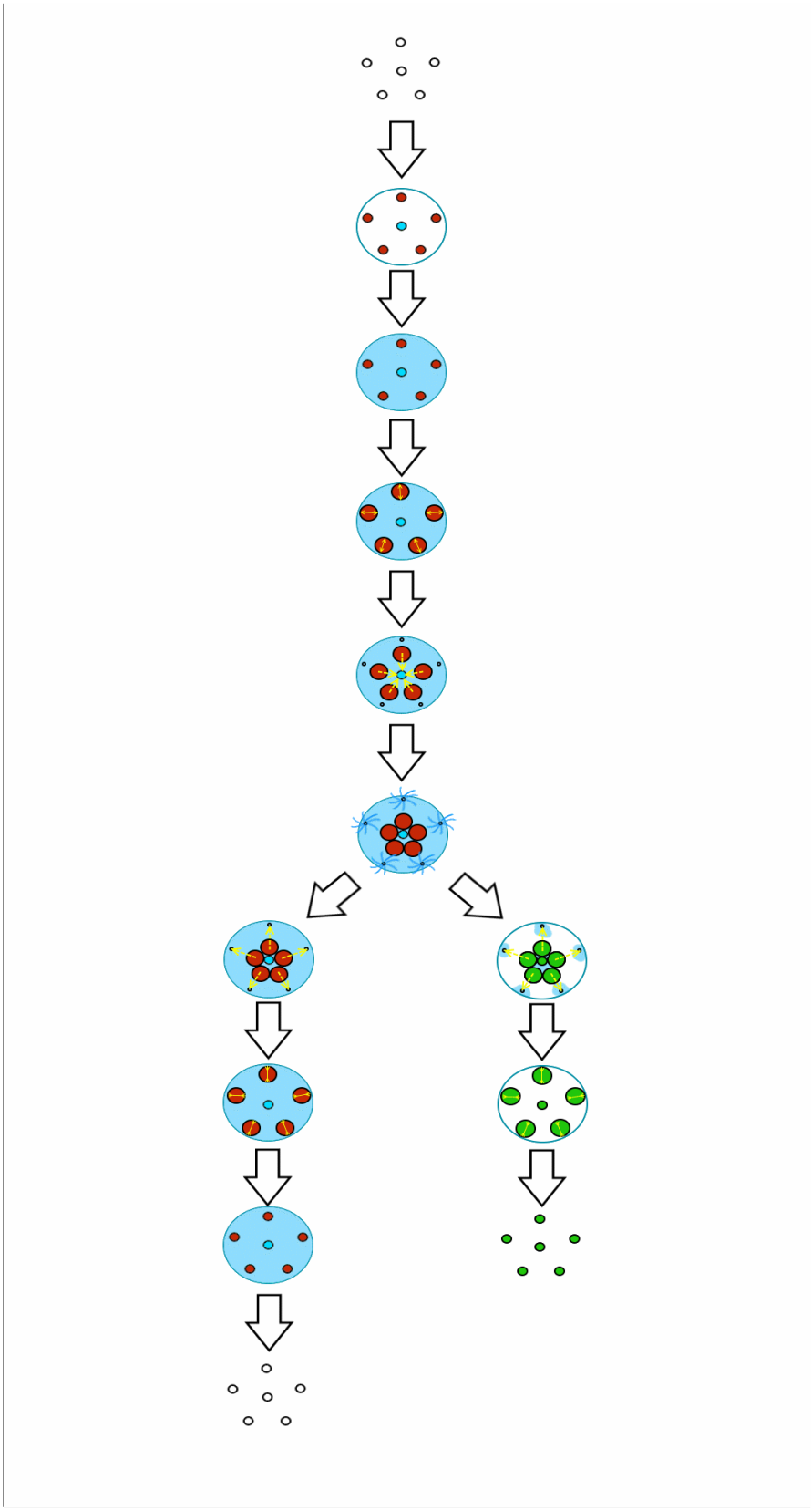


Рис.1. Псевдоблок-схема работы алгоритма по описанию выше

Алгоритмические операции:

1. Выбор вершины (точки поиска)
2. Проверка окрестности на удовлетворение условиям
3. Присвоение кластеру

Из перечисленных анимируются только первые две.

Преимущества:

1. Хорошо выделяются действующие лица на каждом шаге выполнения
2. Достаточно демонстрации всего 2-х случаев: успешный и потерпевший неудачу поиск - для понимания принципа отбора вершин в кластер

Объединение и защита

1. Пусть есть Rb,N ϵ N, где Rb- радиус поиска, N- минимальное кол-во вершин, необходимое для включения в кластер
2. Все вершины считаем некоторыми абстрактными существами (н-р бактериями)
3. Корневую вершину v0(вокруг которой ищутся близлежащие) считаем начальником
4. Начальник созывает близлежащие вершины (отображаем звуковую волну радиусаRb)
5. Вершины, попавшие под звуковую волну, считаем объединенными и огораживаем их зоной радиуса Rb

6. Создаем множество анти-вершин противоположного цвета из

Nэлементов

7. Множество анти-вершин “сталкиваем” с только что объединенными вершинами (если происходит столкновение, погибают обе)
8. Если после столкновения в зоне осталась хотя бы одна анти-вершина, поиск считаем неудачным и разъединяем объединенные вершины
9. Возвращаем вершины в исходное положение

Алгоритмические операции:

1. Выбор вершины (точки поиска)
2. Проверка окрестности на удовлетворение условиям
3. Присвоение кластеру

Преимущества:

1. Наблюдателем легко определяются действующие лица
2. Скоротечный ход событий, не затуманивающий разум

Костры (цепная реакция)

Метафора исходит из предположения самого алгоритма: вершины должны находится максимально плотно, чтобы объединяться в кластер.

1. Пусть есть Rb,N ϵ N, где Rb- радиус поиска, N- минимальное кол-во вершин, необходимое для включения в кластер
2. Все вершины считаем пачками хвороста или факелами (или другими горючими веществами)
3. Вершину, считающуюся инициатором объединения (вокруг которой начинаем вести поиск), поджигаем
4. Все вершины, входящие в зону радиуса Rb, ”подхватывают” огонь
5. Если в указанной зоне число вершин < N, костер считаем нежизнеспособным и такие вершины гасим
6. В конце поиска все смежные горящие вершины считаются объединенными в кластер

Алгоритмические операции:

1. Выбор вершины (точки поиска)
2. Проверка окрестности на удовлетворение условиям
3. Присвоение кластеру

Преимущества:

1. Элементарно визуализируется и анимируется
2. Скоротечный ход событий, не затуманивающий разум

Также, пока выбирался подходящий алгоритм для анимации, была предложена метафора для k-средних:

Собаки и кости

1. Пусть есть набор вершин V, и натуральное число N- желаемое кол-во кластеров
2. Все вершины считаем костями
3. В случайно выбранных местах располагаем Nцентроидов, отображаемых в виде собак
4. На каждой итерации собаки огораживают зону, в которой находятся ближайшие к ним кости
5. Собака (центроид), к которой кость (вершина) находится ближе по расстоянию, считается правой и забирает территорию себе
6. В конце итерации все центроиды располагаются в центре размеченной зоны
7. Повторяем итерации до тех пор, пока предыдущий результат не будет равен текущему

Алгоритмические операции:

1. Определение центроидов
2. Определение позиции центроидов на следующей итерации
3. Перераспределение вершин между центроидами

Преимущества:

1. Мир метафоры знаком наблюдателю ввиду естественности происходящего на экране
2. Действующие лица вызывают интерес у зрителя, но реакция зависит от качества реализации их анимации

Описанные метафоры для алгоритма DBSCAN имеют свои преимущества, однако, варианты “Объединение и защита” и “Цепная реакция” подразумевают слишком быстрое исполнение, вследствие чего хоть результат выполнения и будет достигнут, скорее всего, не будет осмыслен аудиторией за время выполнения. Также, для данных метафор нетривиально подобрать логичное объяснение стадии деинициализации, когда все данные должны вернуться в исходную позицию – неестественный возврат состояния непременно встретит критику наблюдателя.

Метафора “Собаки и кости” привлекательна и проста для понимания, но ввиду критичности ума зрителя не способна сконцентрировать его внимание на суть алгоритма, т.к. наблюдатель будет следить исключительно за качеством анимации действующих лиц: их перемещения, естественных действий вроде подбора и охраны костей и т.д.

Поэтому, считаю, что свой выбор будет справедливо остановить на метафоре “Наводнение” в виду неспешности исполнения и концентрации зрительского внимания именно на принципе работы алгоритма, а не на красоте картинки и качестве исполнения.

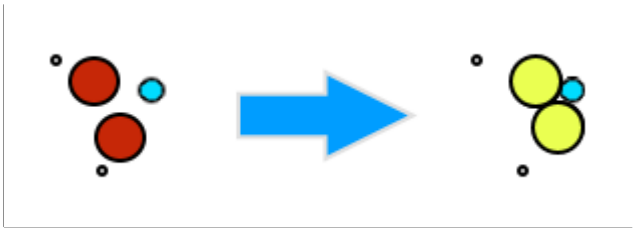
3.2. Сведения о выбранной метафоре

При том, после 5-го шага есть возможность построить барьер вручную, чтобы пользователю в процессе интеракции было проще понять цель образа - показать достаточность в количестве данных. Для этого, необходимо предусмотреть несколько нюансов:

1. Не учитывать скольжение по воде, т.к. в противном случае строить барьер будет крайне неудобно

2. При сочленении вершин выделять их особым цветом, чтобы было понятно, что вершины образуют единую “стену”

Рис.2. Пример смежения



3. Есть вероятность возникновения ситуации, когда пользователю придется образовать цельное кольцо и при вставке последнего элемента, не требуется препятствовать вставке (н-р отталкивание соседних вершин при их сочленении) - смена положения вершины вручную заключается в механике “указать желаемую точку -> сблизиться”, при том, при нажатии левой кнопки мыши показывать штрихом результирующую позицию (не реализовано).

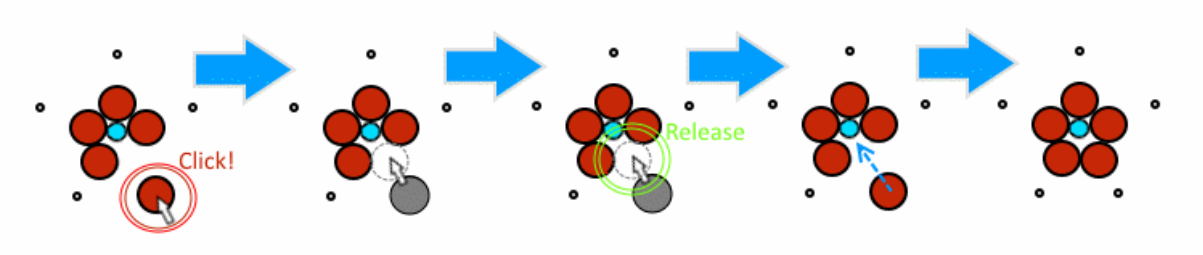


Рис. 3. Пример ручного перемещения

4. Также возможен случай, что пользователь по невнимательности не сможет построить барьер, например, если вершин слишком много или если их тяжело сопоставить - на сей счет есть кнопка “Auto”, завершающая процесс автоматически, либо деинициализирующая процесс, если построение невозможно по причине нехватки данных (вершин), сопровождая это сообщением в логе

Также, при проектировании стоит заострить внимание на самом барьере, а именно каким образом он будет строиться. Логично предположить, что для того, чтобы вода не распространялась далеко за пределы источника, вокруг него необходимо построить многоугольник, причем не важно, какой формы - вершинами в данном случае будут служить точки соприкосновения “распухших” вершин.

В случае автоматического построения барьера достаточно выстроить все вершины в кольцо, т.к. это идеальный и в то же время легко понятный случай, а значит для данного режима необходимо:

1) выбрать радиус будущей вершины-источника воды (Rsrc);

2) вычислить радиус, до которого “разбухнут” остальные вершины в

окрестности источника (Rv);

3) вычислить радиус кольца вокруг источника - суть барьера (Rc).

В данном случае, изначальный радиус вершины источника увеличивается в 1,5-2 раза для выделения его среди других вершин и меняет цвет, близкий к водяному - голубому. После, отмечаем поле радиуса

Rbвокруг источника и делим эту окружность от центра на n- секторов, где

n - кол-во вершин, попавших в окрестность. Затем вычисляется минимальное расстояние, на котором должны находиться “барьерные” вершины от источника - это и будет нашим радиусом в общем случае

считаем его равным3Rsrc, и в конечном итоге на этом расстоянии

вставляем воображаемую окружность, “вмещающуюся в рамки радиусных лучей” (см. рис) - радиус этой окружности и будет нашим Rv.

Однако, сразу возникнет вопрос, что делать если кол-во вершин в окрестности $n < 3$? Ведь 2 и менее окружностей не могут образовать замкнутую цепь - данным случаем придется пренебречь, кроме того, он не критичен.

Если же пользователь считает необходимым вручную построить барьер, мы его не ограничиваем четкой формой будущего “забора”, несмотря на подсказки при выделении вершин.

3.3. Описание доступного инструментария

Ввиду стремительного развития игростроения, с течением времени на свет появляется все больший ассортимент инструментов, упрощающих их разработку. Мультиплатформенную поддержку одно время возглавляла среда «cocos-2dx» от китайских разработчиков, позволяющая вести разработку игровых приложений для почти всех доступных популярных систем на языке C++. Несмотря на активное развитие, в данный момент считается устаревшим в виду необходимости выполнения излишней ручной работы и неудобно разделенной функциональности.

Также довольно популярным является фреймворк «Phazer», поддерживающий только приложения для web'а. Разработка ведется на языке JavaScript. Набор инструментов имеет содержательную документацию и большое количество примеров, позволяющих собрать свое приложение из их кусков.

Вот уже несколько лет в качестве стандарта на вершине стоит среда «Unity», которая и была избрана для разработки системы анимации в силу ее простоты, низкого порога вхождения, богатого инструментария, широкого спектра дополнительных утилит и доступной поддержкой как среди официальных разработчиков, так и от любителей.

3.4. Сведения о среде разработки

3.4.1. Основные необходимые элементы

Хоть «Unity» и позволяет разрабатывать продукты без помощи скриптов, но даже не обращая внимания на огромные его возможности, сложные задачи невозможно решить без помощи языка программирования (поддерживаются C# и JavaScript, которые и были использованы в процессе). Так некоторые элементы анимации были реализованы при помощи “поведений” (“behaviours”) - основных объектов для написания сценариев: имеют почти 2 десятка типов событий, позволяющих легко вклиниваться в мельчайшие детали процесса, основные из них:

1) Start () - вызывается при инициализации компонента, содержащего поведение
2) Update () - вызывается каждый раз при обновлении кадра

Для простого скриптинга этого вполне достаточно, но все равно стоит отметить, что жизненный цикл приложения разделен на следующие итерации (и в каждой из них имеет управляемые события):

1) Инициализация;
2) Исполнение физики;
3) Обработка устройств ввода;
4) Игровая логика;
5) Рендеринг исполняемой сцены;
6) Рендеринг Gizmos (в основном это объекты для отладки);
7) Рендеринг элементов интерфейса пользователя;
8) Конец кадра;
9) Возможная остановка приложения;
10) Фаза включения (вызывается если поведение было прикреплено в течение текущего кадра);

11) Терминация процесса (завершение работы приложения).

При моделировании данных для анимирования алгоритма основной задачей является вычисление всех будущих характеристик объектов и осуществление их плавного перехода между кадрами.

Среда «Unity» предоставляет возможности по ручному созданию, хранению и воспроизведению анимации, при том позволяя при анимировании изменять широкий спектр характеристик объекта, выстраивая их в логический ряд кадров по градации/деградации. В качестве примера можно выделить объект Animator, предоставляющий разработчику возможности выстраивать конечный автомат состояний аниматора, настраивать переходы между состояниями, типы воспроизведений и прочее. На текущий момент данный объект является действующим и широко используется почти всеми разработчиками.

Однако, в ходе работы выяснилось, что, во-первых, данный объект обладает неприятной неисправностью, блокирующей изменение характеристик объекта, у которого в текущий момент уже запущена анимация, либо закончена и остановлена на последнем кадре (не выставлен флаг LoopPlay). Вследствие чего пришлось избавиться от возможности приукрашивания промежуточных анимаций вершин.

С другой стороны, есть компонент Animation, являющийся предшественником более продвинутого аниматора, но предоставляет более скудный набор функциональности, поэтому хоть и заложен в программный продукт, как использующийся, но не воспроизводим.

«Unity» поддерживает сценарии на языках C# и JavaScript, широко распространенных, но в силу знакомства выбор пал на C#.

3.4.2. Сборка проекта

Сборка осуществляется посредством редактора среды «Unity» во вкладке File -> Build Settings, содержащей 2 окна:

- Список загружаемых в собираемый проект сцен (файлы *. unity)
- Настройки сборки: выбор платформы и тип сборки (для разработчика, выпуск и пр.)

См. рис. 7.

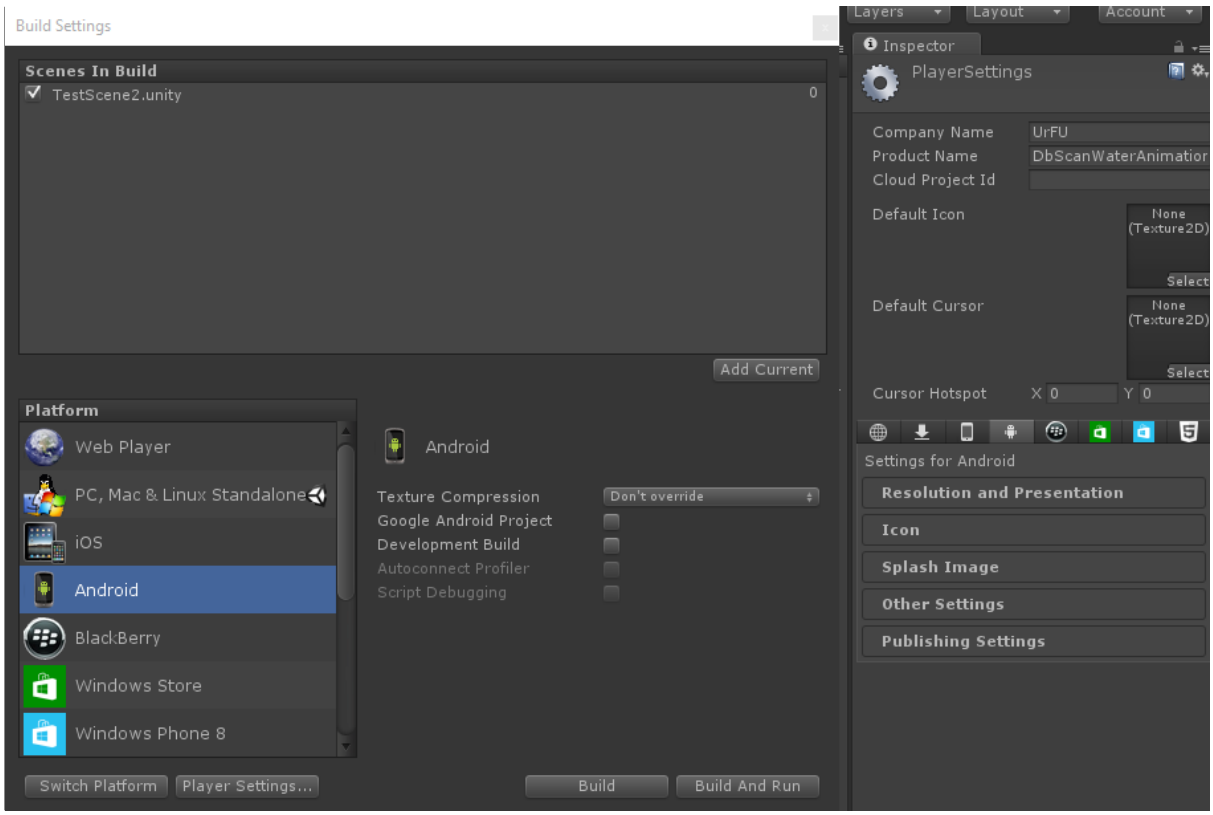


Рис.7. Интерфейс сборки

Также панель Inspector для Player Settings позволяет выполнить общие настройки собранного приложения: изменить разрешение, настройки окна, иконки, экран загрузки и пр.

4. Реализация

4.1. Архитектура программы

Программное обеспечение, реализующее вышеописанную анимацию состоит из:

1) Управляемых сущностей;
2) Управляющих сущностей;
3) Поведений.

Основные объекты представлены в таблице:

Пространство имен: **Algorithm**

Название	**Описание**
Vertex	Сущность, представляющая вершину Является оберткой для UnityEngine.GameObject, предоставляя пользователю интерфейсы для управления необходимыми для анимирования данными: ○ Ссылку на объект структуры VertexData, предоставляющую данные о вершине, как геометрическом объекте ○ Ссылки на управляющие объекты для анимирования вершины ○ Ссылку на текущий кластер, ассоциированный с вершиной ○ Свойство-обертку для упрощения управления компонентом SpriteRenderer ○ Методы отображения и сокрытия вершины от наблюдателя
VertexData	Структура, предоставляющая данные о вершине, как о геометрическом объекте Включает: ○ Ссылка на исходный UnityEngine.GameObject ○ Ссылки на Transform-объекты визуальных составных частей вершины: собственно

	вершину, область поиска и воду ○ Радиус вершины (единичный - без учета масштаба) ○ Видимый размер ○ Позиция (shortcut) ○ Ссылка на объект UnityEngine.Animator
VertexCursor	Сущность-оболочка для объекта, представляющего курсор поиска: ○ Ссылка на исходный UnityEngine.GameObject ○ Ссылки на объекты-аниматоры
Cluster	Сущность, представляющая кластер: ○ Уникальный идентификатор кластера ○ Цвет вершин, входящих в кластер
BarrierData	Собирательная структура, для хранения данных о барьере, получаемом в процессе анимации: ○ Радиус объекта-источника ○ Радиус вершин в области поиска (“раздутых” вершин) ○ Радиус окружности, строящейся вокруг источника (расстояние от источника до вершины после построения барьера) ○ Список ссылок на позиции вершин в барьере
AnimationStage	Перечисление, собирающее список идентификаторов итераций анимации алгоритма: ○ Отсутствие анимации ○ Инициация курсора ○ Движение курсора ○ Инициализация области поиска ○ Итерация наводнения области поиска ○ Разбухание вершин ○ Выстраивание вершин в барьер ○ Кластеризация ○ Деинициализация (возвращение в исходные позиции) ○ “Посев семян” ○ Деаллокация курсора

Табл. 1. Управляемые сущности

Название	**Описание**
DataOrdinator	Класс, предоставляющий данные и интерфейс управления текущим вводом: ○ Список ссылок на вершины, участвующие в анимации ○ Ссылка на текущую вершину-источник ○ Список ссылок на вершины, попавшие в область поиска ○ Ссылка на объект-структуру BarrierData ○ Метод для переключения на следующую итерацию алгоритма
VertexInstantiator	Класс-помощник, предоставляющий интерфейс-оболочку для создания вершин со всеми прилагающимися суб-объектами. Принимает на вход prefab-заготовки этих объектов.
ClusterManager	Singleton-объект, предоставляющий интерфейс для управления кластерами: ○ Счетчик когда-либо созданных кластеров ○ Список ссылок на существующие кластеры ○ Интерфейсы создания и удаления кластеров
MathUtils	Вспомогательный класс для промежуточных вычислений: ○ Интерфейс определения вхождения вершин в область поиска ○ Интерфейс вычисления BarrierData
UIManager	Вспомогательный класс для управления пользовательским интерфейсом: ○ Ссылки на текстовые поля, слайдеры, кнопки, текстбоксы и объект-поле логирования ○ Промежуточные обработчики кнопок и слайдеров
AppManager	Сущность, предоставляющая интерфейс управления выполнением приложения
LogManager	Класс, предназначенный для вывода логов во время выполнения алгоритма в специальный элемент интерфейса
AreaScaleAnimManager	Класс, предоставляющий управление анимацией масштабирования области поиска: ○ Метод для оборачивания вершины-источника в объект

	○ Данные по исходному и желаемому масштабу ○ Свойство текущего масштаба ○ Метод проигрывания кадра анимации (вперед или назад) ○ Методы отображения и сокрытия объекта
WaterAnimManager	Класс управления анимацией наводнения
SwellAnimManager	Класс управления анимацией разбухания вершин
SwellAnimOrdinator	Обобщающий класс-обертка, упрощающий управление одновременной анимацией разбухания вершин
MoveAnimManager	Класс, анимирующий передвижение вершин: ○ Ссылка на анимируемый объект ○ Текущая и позиция-цель ○ Метод воспроизведения кадра анимации (вперед/назад)
UninitAnimManager	Класс, предоставляющий интерфейс для покадрового воспроизводящий всей анимации в обратном порядке и возвращения всех объектов в исходное состояние

Табл.2. Управляющие сущности

Также в программа предоставляет интерфейс для удобного создания и запуска визуальных тестов:

Пространство имен: **Algorithm.Testing**

Название	**Описание**
TestDataProvider	Класс, предоставляющий набор методов, возвращающих ссылку на объект *TestLauncher*, предварительно оборачивая в него данные.
TestLauncher	Класс, предоставляющий интерфейс запуска тестов, принимая на вход: ○ Список вершин ○ Радиус поиска ○ Минимальное кол-во вершин, необходимых для образования кластера
TestManager	Singleton, предоставляющий доступ к данным о выполнении тестов.

Табл.3. Интерфейсы тестирования

Название	**Описание**
AppPreparationBehavior	Скрипт, инициализирующий приложение необходимыми для работы данными
AlgorithmAnimationBehavior	Скрипт-поведение, отвечающий за инициацию процесса анимации
VxCreationBehavior	Поведение, реализующее управление генерацией вершин
TestPanelBehavior	Поведение, предоставляющее редактору методы-обработчики элементов интерфейса тестирования
InitCursorBehavior	Скрипт, инициализирующий курсор
CursorMoveBehavior	Осуществляет поведение курсора
InitAreaBehavior	Поведение, осуществляющее анимацию шага инициализации и подготовки к следующему этапу
DelugeAreaBehavior	Поведение, анимирующее наводнение области поиска и подготавливает программу к следующему этапу
SwellingBehavior	Скрипт, реализующий поведение программы во время анимации разбухания вершин
MovingBehavior	Поведение перемещения вершин во время выстраивания барьера вокруг вершины-источника. Если барьер построить нельзя, моделируется неполное кольцо
ClusterAssignBehavior	Поведение, исполняющее управление кластеризацией в промежуточном этапе анимации алгоритма
UninitiatingBehavior	Сценарий-поведение, анимирующее возвращение всех участвовавших визуальных объектов в исходное состояние
SetSeedsBehavior	Реализует итерацию “посева семян” при удачно совершившимся поиске
DeallocCursorBehavior	Анимация деаллокации курсора

Табл. 4. Сценарии поведения

4.2. Вычисление промежуточных данных

В ходе реализации анимации алгоритма на основе выбранной метафоры возникла проблема образования барьера на основе данного количества вершин внутри области поиска. На сей счет, в случае не ручного его построения, сошлись на том, что достаточно будет строить кольцо вокруг вершины-источника поиска - в этом случае достаточно показать наблюдателю всего 2 случая итерации поиска, в которых вершины образуют кластер и не образуют соответственно ввиду нехватки звеньев. Так как подбор оптимального расстояния от вершины-источника до барьерных вершин, причем с учетом радиуса “разбухших” вершин задача не тривиальная, кроме того “съедает” много ресурсов ЦП, принялось решение задать это расстояние по-умолчанию.

Так, радиус окружности барьера вычисляется:

$Rb = Rv / \sin(\pi / \max(N, Nb))$, где Nb- кол-во вершин, попавших в область поиска, Rv- радиус барьерной вершины.

Rv=Rv0*K / max(N,Nb), где Rv0- радиус вершины-источника, K-

коэффициент масштабирования ~= 43,8.

Координаты барьерных вершин вычисляются при помощи матрицы поворота с учетом текущей позиции:

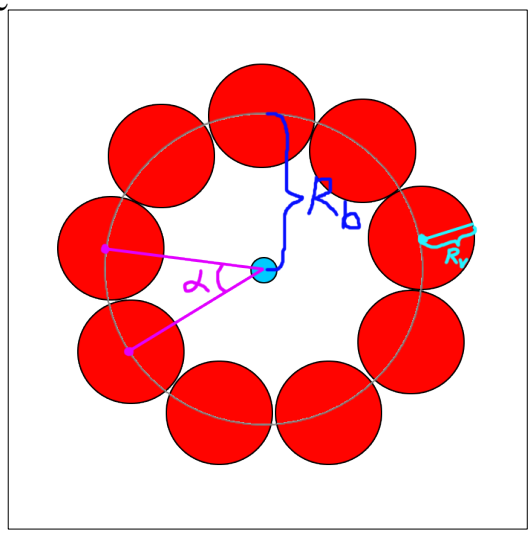
$$x'=x*\cos(\alpha) - y*\sin(\alpha)$$

$$y'=x*\sin(\alpha) + y*\cos(\alpha)$$

, где α - угол между центрами окружностей барьерных вершин.

См. рис 4.

Рис. 3. Обозначения на картинке



Так это выглядит в коде:

```
BarrierData CalculateBarrierData(IList<Vertex> vertices, Vertex sourceVertex)
{
    int circlePartsCount = Mathf.Max (vertices.Count, DataManager.Instance.MinClusterizableVerticesCount);
    float anglePerElementRad = 2 * Mathf.PI / circlePartsCount;

    float sourceVxRadius = sourceVertex.Data.VisibleSize.x / 2;
    float currentDistanceToVxCenter = sourceVxRadius;
    float vxRadius = 0;

    const float SCALE_KOEFF = 6 * 7.3f;
    vxRadius = sourceVxRadius * (SCALE_KOEFF / circlePartsCount);
    float distanceToVxCenter = vxRadius / Mathf.Sin(Mathf.PI / circlePartsCount); // external circle radius
    currentDistanceToVxCenter = distanceToVxCenter;

    Vector3 startVxPosition = new Vector3 (sourceVertex.Data.Position.x +
```

```
currentDistanceToVxCenter, sourceVertex.Data.Position.y, 1);
        var normalizedStartVxPosition = new Vector3 (startVxPosition.x -
sourceVertex.Data.Position.x, startVxPosition.y -
sourceVertex.Data.Position.y, 1);

        var futurePositions = new List<Vector3> ();
        futurePositions.Add (startVxPosition);
        for (int i = 1; i < vertices.Count; ++i) {
                float rotationAngleRad = anglePerElementRad * i;
                var nextVxPosition = new Vector3 (normalizedStartVxPosition.x
* Mathf.Cos (rotationAngleRad) - normalizedStartVxPosition.y * Mathf.Sin
(rotationAngleRad),       normalizedStartVxPosition.x * Mathf.Sin
(rotationAngleRad) + normalizedStartVxPosition.y * Mathf.Cos
(rotationAngleRad), 1);
                nextVxPosition = new Vector3(nextVxPosition.x +
sourceVertex.Data.Position.x, nextVxPosition.y +
sourceVertex.Data.Position.y, 1);
                futurePositions.Add (nextVxPosition);
        }
        return new BarrierData (
                sourceVxRadius,
                vxRadius,
                currentDistanceToVxCenter,
                futurePositions);
}
```

Кроме того, как было отмечено, вычисления ведутся только по тем вершинам, которые замечены в области поиска. Вот код, вычисляющий их:

```
IEnumerable<Vertex> GetVerticesInsideSearchArea(IEnumerable<Vertex>
vertices, Vertex sourceVertex)
{
	foreach (var searchVertex in vertices) {
		if (searchVertex.Equals (sourceVertex)) {
			continue;
		}

		float distance = Vector2.Distance (
			sourceVertex.Data.Object.transform.position,
			searchVertex.Data.Object.transform.position);

		var areaRadius = (RectTransform)
sourceVertex.Data.AreaCircleTForm;
		if (distance < areaRadius.rect.width * areaRadius.lossyScale.x /
2) {
			yield return searchVertex;
		}
	}
}
```

4.3. Интерфейс взаимодействия с пользователем

Интерфейс программы содержит минимум необходимых элементов управления и прост в понимании (см. рис. 5).

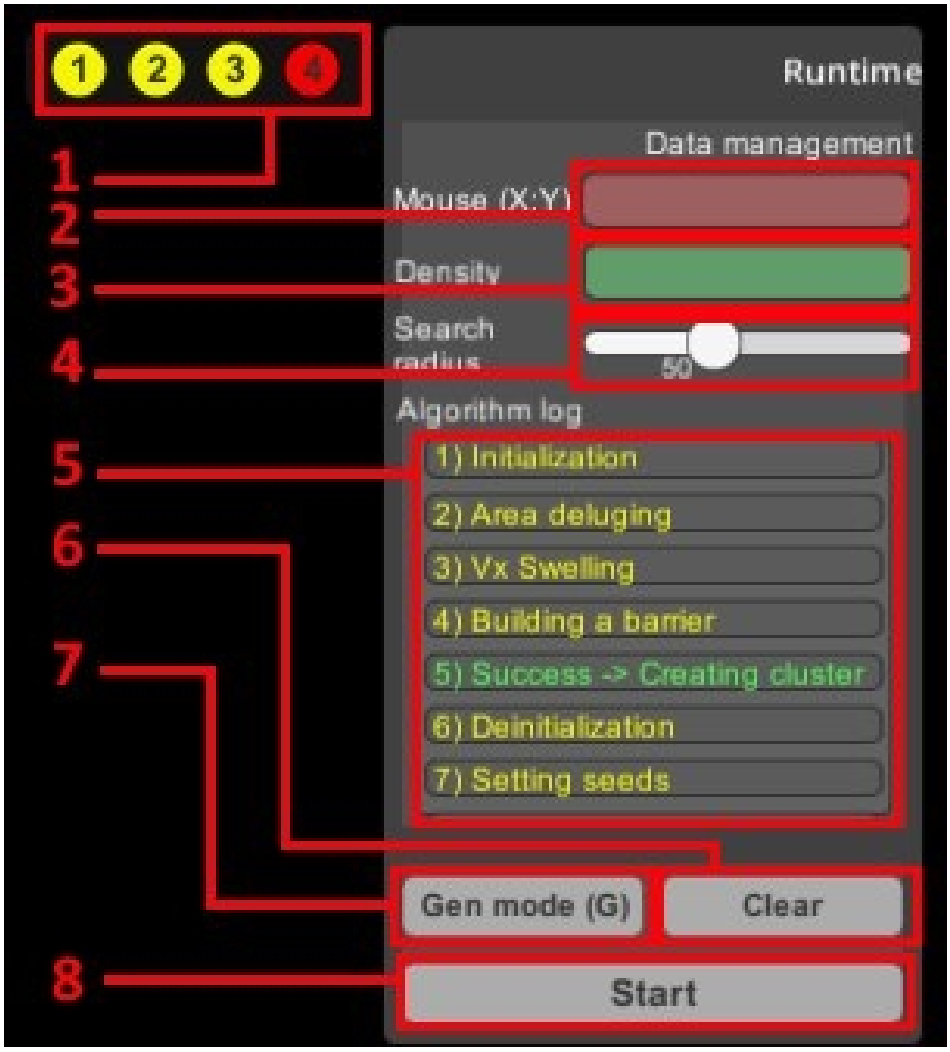


Рис.5. Интерфейс приложения:

1. Панель выбора тестовых наборов с данными
2. Поле с информацией о местонахождении курсора
3. Поле ввода плотности
4. Бегунок, управляющий значением размера области поиска
5. Панель с логами выполнения алгоритма
6. Кнопка очистки данных
7. Кнопка активации ручного режима генерации данных
8. Кнопка запуска

Панель с тестовыми наборами данных предоставляет 4 варианта граничных случаев работы алгоритма без необходимости генерировать из вручную:

- Нерасширяемая кластеризация (когда в процессе поиска «посев семян» не выходит за рамки одной вершины) со случаями образования кластера и идентификацией шумов
- Расширяемая кластеризация («семена» выходят далеко за рамки курсора) с наличием шумов
- Перекрывающие друг друга кластеры
- Критический случай работы приложения

Поле ввода плотности позволяет вводить только числа, превышающие 2, т.к. метафора не подразумевает выполнение с меньшим числом необходимых для построения кластера вершин.

Бегунок управления размером области поиска позволяет выбрать значение в диапазоне [20; 100] (в пикселях), т.к. слишком малый или слишком большой размер использовать не имеет смысла.

Панель логов дополняется в процессе выполнения алгоритма. Является прокручивающимся по вертикали списком.

Кнопка очистки данных удаляет все вершины, сгенерированные пользователем, либо автоматически.

Кнопка активации режима генерации данных позволяет пользователю вручную создавать вершины на поле. Активация также возможна по нажатию кнопки «G» на клавиатуре. Данный элемент

управления был необходим в виду одновременной обработки средой нажатий на элементы интерфейса и создания вершин.

Также, перед запуском работы алгоритма интерфейс содержит тестовую вершину, отображающую примерные характеристики вершин в будущем процессе (см. рис. 6):

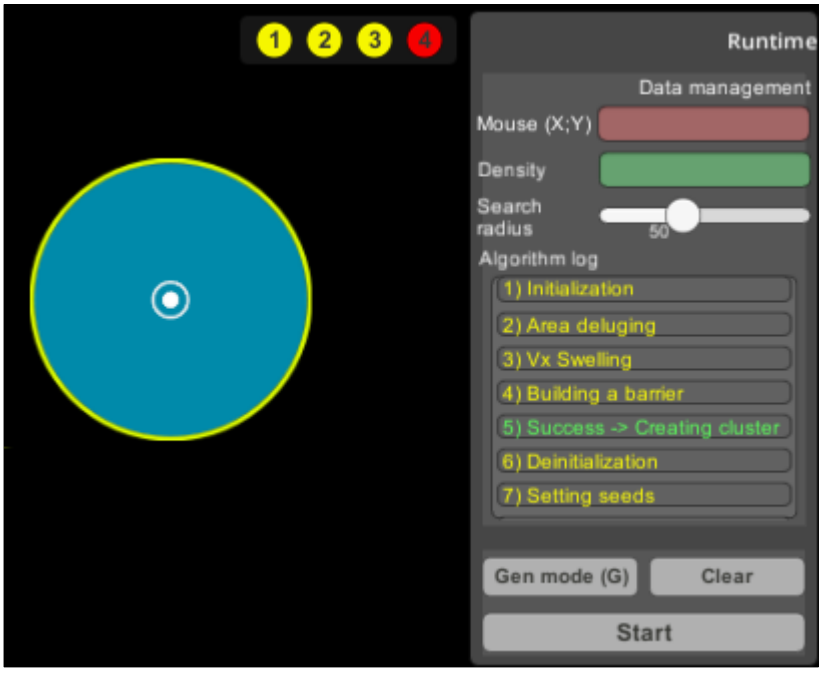


Рис.6. Тестовая вершина:

- Желтое – граница области поиска
- Голубое – вода внутри этой области
- Белый полый круг вокруг вершины – курсор
- Белый кружок в центре – сама вершина до обработки

5. Тестирование

Проверка корректности выполнения алгоритма и его анимации проводилась исключительно вручную, но с использованием тестовых наборов данных (см. раздел 4.1).

Основную проблему вызывала ошибка, возникающая в случаях, если центр вершины лежит ровно на границе области поиска: программа теряет контроль и теряет управление и требует перезагрузки, так как «Unity» не завершает работу приложения в случае фатальной ошибки.

Также стоит снова отметить, что в среде разработки есть неисправность, не позволяющая использовать включенные в нее элементы, отвечающие за анимацию.

В общем и целом, приложение работает корректно за исключением случая, описанного выше, но он возникает крайне редко, что не мешает использованию.

Заключение

Таким образом, на основе проведенных исследований было построено приложение, в котором реализовано большинство поставленных задач. Исследованы вопросы предоставления информации пользователю, простроена простейшая модель данных с последующей ее реализацией, проведено тестирование, вся реализованная функциональность является работоспособной. Создано приложение, предоставляющее возможность изымать необходимую информацию. Однако задуманная изначально функциональность приложения реализована не полностью.

В процессе выполнения были изучены азы работы со средой разработки «Unity», принципы построения компонентов графики и задача кластеризации алгоритмов.

Ввиду того, что приложение находится на начальном этапе развития, перспектив для его модернизации предостаточно, но основная техническая часть уже реализована.

ЛИТЕРАТУРА

[1] - A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu (Institute for Computer Science, University of Munich)

http://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[2] - A Density-Based Spatial Clustering of Application with Noise

Henrik Bäcklund, Anders Hedblom, Niklas Neijman

http://staffwww.itn.liu.se/~aidvi/courses/06/dm/Seminars2011/DBSCAN(4).pdf

[3] - Data mining

https://en.wikipedia.org/wiki/Data_mining

[4] - Semiotic way on generation of a computer visualization theory

Averbukh Vladimir Lazarevich

[5] - Development of Visualization-Animation Software for Learning Transportation Algorithms

Ivan P. Makohon

[6] - Critical Analysis on Algorithm Visualization Study

Ahmad Affandi Supli, Norshuhada Shiratuddin, Syamsul Bahrin Zaibon

[7] – Clusterization algorithms

Nikolenko Sergey

[8] – Mathematical methods on learning by precedents

Vorontsov K.V.

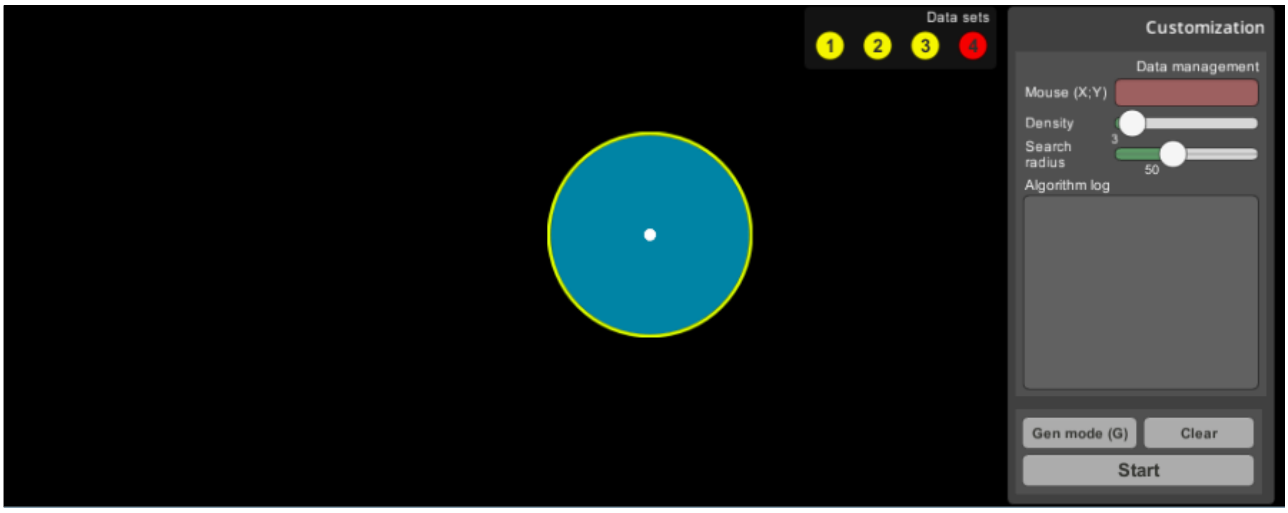
ПРИЛОЖЕНИЕ

Отрывок из код одной из итераций анимации, показывающий, как примерно реализован процесс (выполняется раз в кадр):

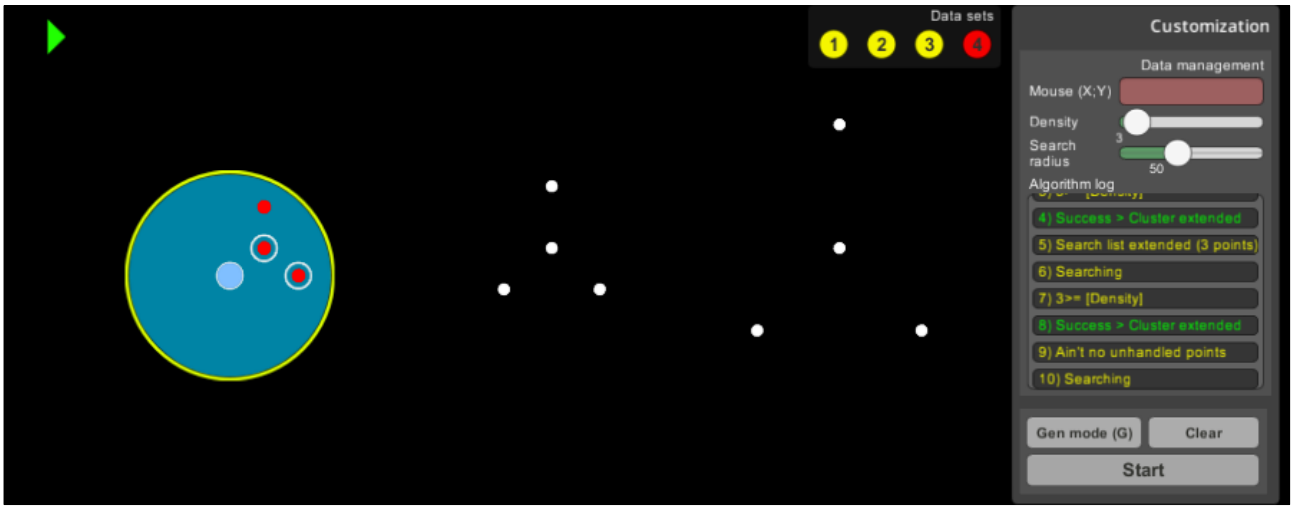
```
bool isAnimFinished = true;
var srcVx = dataManager.Ordinator.CurrentVx;
foreach (var seed in dataManager.Ordinator.VxSeeds) {
	isAnimFinished &= seed.SeedScaleAnimManager.PlayFrame (true);
}
isAnimFinished &= srcVx.SeedScaleAnimManager.PlayFrame (false);
if (isAnimFinished) {
	if (dataManager.Ordinator.SetNext () != DEFAULT_INDEX) {
		AreaScaleAnimManager.Instance.SetVx
(dataManager.Ordinator.CurrentVx);
		AreaScaleAnimManager.Instance.ShowArea ();
		if (dataManager.Ordinator.VxSeeds.Count > 0) {
			dataManager.CurrentStage = AnimationStage.INIT_AREA;
		} else {
			dataManager.Ordinator.VxSeeds.Enqueue
(dataManager.Ordinator.CurrentVx);
			dataManager.CurrentStage =
AnimationStage.CURSOR_MOVING;
	}
} else {
		dataManager.CurrentStage = AnimationStage.NONE;
	}
}
```

Скриншоты работы приложения:

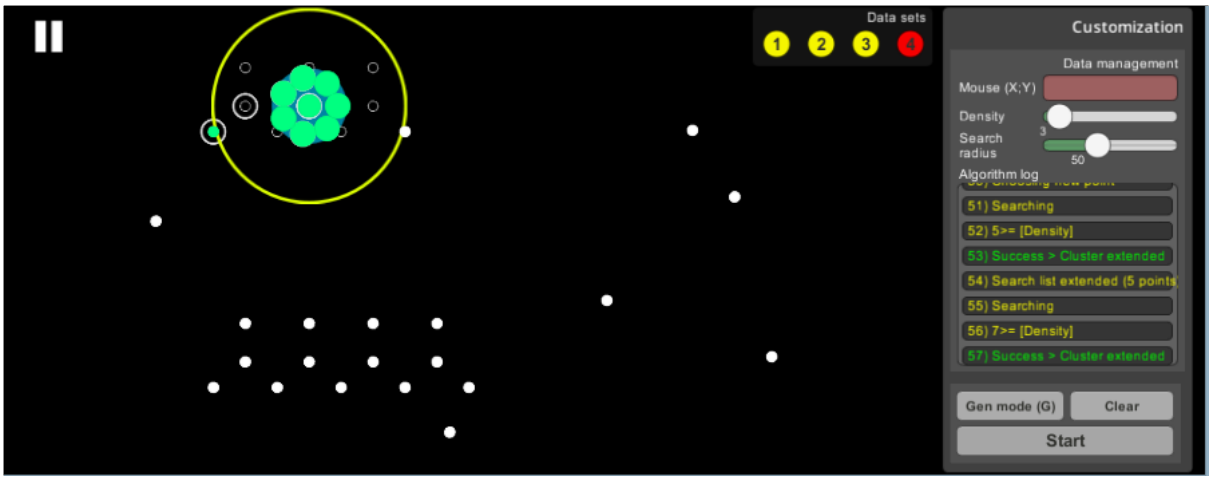
1) Состояние до запуска алгоритма



2) Процесс поиска (начало)



3) Объединение кластеров



4) Демонстрация результата