МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОПИСАНИЕ ПРЕДПОЛАГАЕМОГО СПОСОБА РЕШЕНИЯ

Студент гр. 5304 _____ Скиба А.С.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2020

Описание метода решения.

Решение поставленной задачи достигается в 3 этапа, для начала архитектурные особенности реализации, описание интерфейса пользователя, далее используемые алгоритмы для реализации конечного приложения.

Архитектурные особенности реализации.

1. Архитектура приложения

Для разрабатываемого приложения была выбрана клиент-серверная архитектура, показанная на рисунке 1.

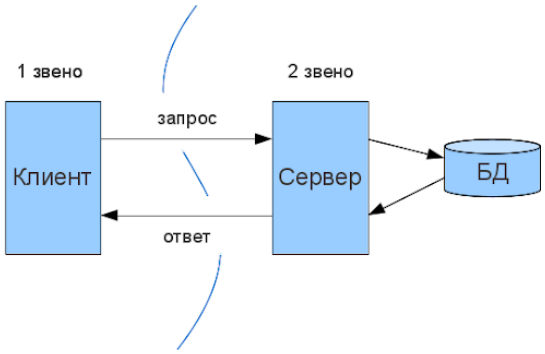

Рисунок 1 - Архитектура приложения

Данная архитектура представляет собой распределенную структуру приложения, в которой четко распределяются сфера исполнения задач – сервер, и сфера запросов на исполнение различного рода задач – клиент. Клиентские программы выполняются на устройствах пользователей. Сервер принимает запросы от клиентов и обрабатывает их, далее отправляет нужную клиенту информацию, на клиенте эта информация отображается в удобном пользователю виде.

Приложение предполагается разрабатывать в виде онлайн-сервиса, то есть конечному пользователю не нужно будет устанавливать дополнительное программное обеспечение, запустить и пользоваться сервисом можно будет из браузера.

2. Предполагаемые технологии для реализации решения.

Предполагается использовать следующий стек технологий:

Для клиентской части приложения:

- HTML – язык гипертекстовой разметки

- CSS – язык стилизации размещаемых на странице элементов
- Javascript (JS) – Основной язык программирования
- Three.js – Библиотека для работы с 3D-графикой в браузере на основе WebGL
- React – Библиотека для создания пользовательских интерфейсов на основе JS

Для серверной части приложения:

- Node.js – язык программирования с синтаксисом JS для построения серверной части приложения
- NPM – утилита для подключения сторонних библиотек для Node.js
- Express – основной фреймворк, предоставляющий обширный набор возможностей для обработки запросов, поступающих с клиентской части.

Также предполагается выбрать MongoDB – система управления базами данных (СУБД), не требующая описания схемы таблиц, то есть база данных будет нереляционной.

Описание интерфейса пользователя.

Пользовательский интерфейс будет разделен на две основные зоны:

Первой зоной является настройка будущего ландшафта, страница предполагает наличие окна для построения графа пользовательских трехмерных моделей, а также несколько панелей, таких как точная настройка характеристик каждого объекта и группы объектов, панель загрузки новых моделей, с которой пользователь сможет перетащить в основное окно загруженную модель, для расположения ее в графе объектов. Далее на странице должны присутствовать функциональные кнопки для продолжения построения и перехода на страницу с будущим, построенным ландшафтом.

Таким образом вторая зона представляет из себя окно, в котором присутствует панель для задания общих характеристик ландшафта и две кнопки, первая предоставляет возможность вернуться назад, вторая начинает рендеринг

поверхности карты. По окончанию рендеринга появляется новая панель для реализации экспорта полученной карты в удобном для пользователя формате, это основная панель всего приложения, на ней будут как минимум такие возможности как выгрузка с объектами, выгрузка без объектов, но с построенными местами для них и выгрузка чистой сетки ландшафта.

Предварительные макеты с отсутствием большей части вышеописанной функциональности представлены на рисунках 2 и 3:

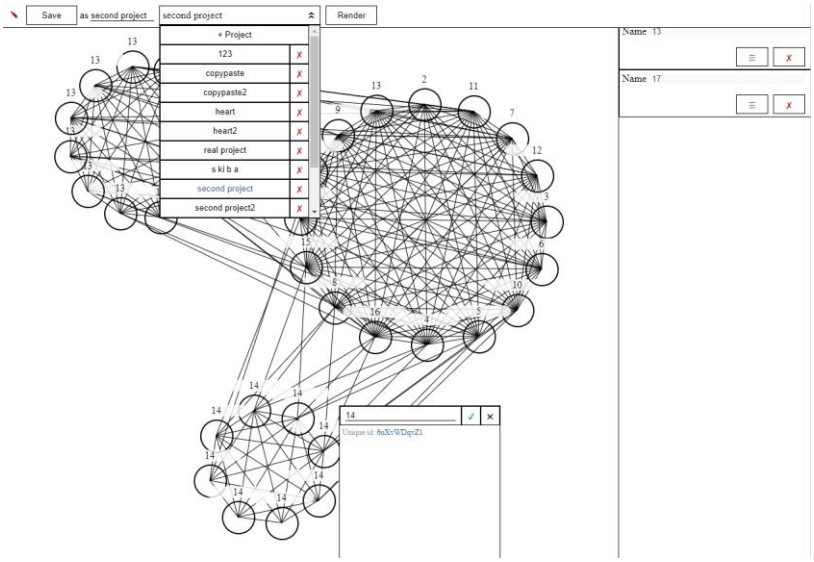

Рисунок 2 – Первое основное окно, настройки объектов

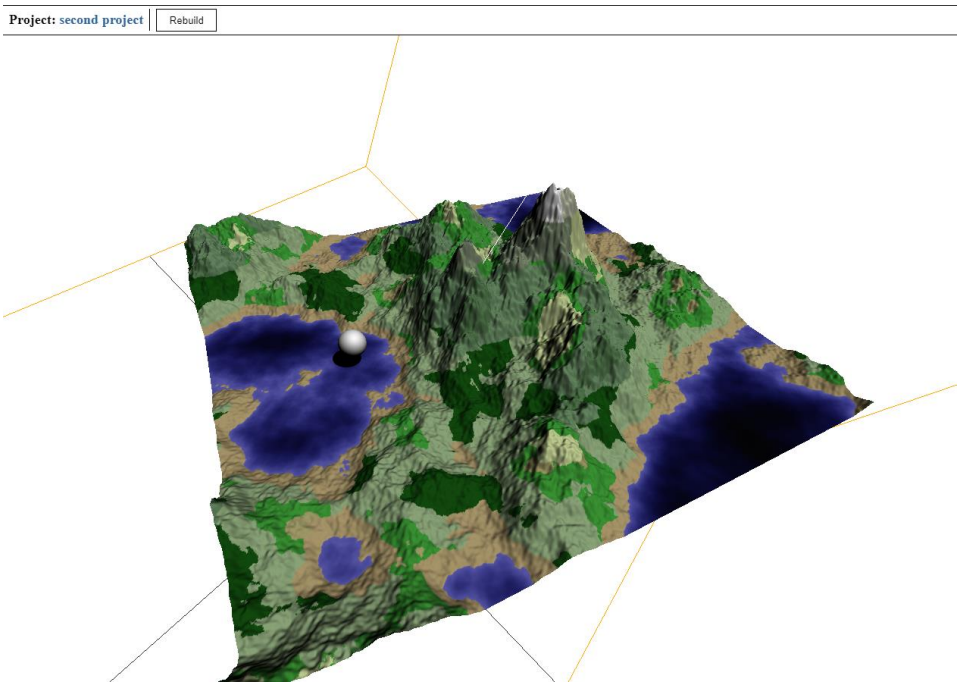

Рисунок 3 - Второе основное окно с настройкой ландшафта

Предполагаемые алгоритмы.

Основными алгоритмическими проблемами являются построение ландшафта и размещение на нем трехмерных моделей в соответствии с пользовательским графом содержащем эти объекты.

Первая проблема может быть решена несколькими уже существующими алгоритмами, на рисунке 3, например, ландшафт построен на основе симплексного шума. Алгоритм на основе симплексного шума строит карту высот, которая потом передается в методы библиотеки Three.js и преобразуется в рельефную сеточную поверхность. Таким же образом работает шум Перлина, но симплексный шум имеет меньшее количество направленных артефактов (рисунок 4), а также имеет меньшую сложность по сравнению с классическим шумом Перлина, $O(n^2)$ вместо $O(n2^2)$.

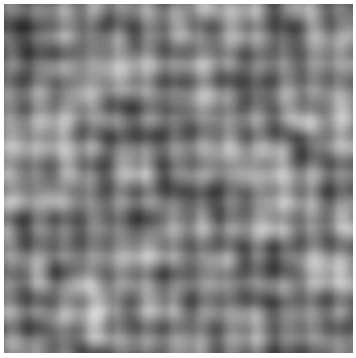

Рисунок 4 - Квадратные артефакты шума Перлина

Но данные алгоритмы относятся к одному и тому же типу, построение карты высот на основе шумовых функций. Существует, например, также известный алгоритм Diamond-Square, который уже основывается на фракталах. Данный алгоритм работает в два шага Diamond и Square соответственно или же ромб и квадрат. Ход построения карты высот описан на рисунке 5.

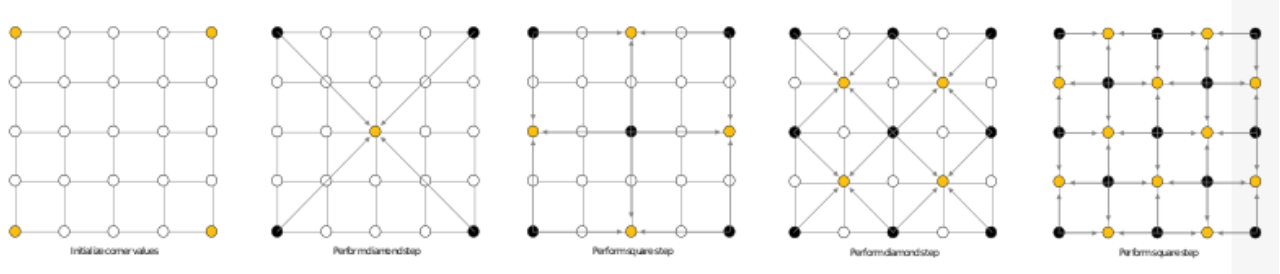

Рисунок 5 - Алгоритм Diamond-Square

На первом шаге выбираются случайным образом значения крайних (желтых) точек, далее поочередно выполняются ромбовидные и квадратные вычисления средних точек от квадрата и ромба соответственно с прибавлением некоторого случайного значения.

На данный момент предполагается использовать один из этих способов для реализации начального рельефа карты.

Второй проблемой является размещение пользовательских объектов из графа на карте. Для начала стоит вообще определить для чего будет использоваться именно граф в данном приложении.

Предполагается, что граф, который будет составлять сам пользователь с помощью интерфейса первого окна, сможет определить отношения между объектами на карте. К примеру, между объектами можно будет определить отношения вероятного расстояния друг от друга, отношение множественности одного объекта вокруг другого (деревья вокруг дома, трава вокруг дерева и дома и т.д.).

Таким образом на основе полученного графа и характеристик каждого объекта в частности предполагается изменять карту высот графа, например, вырезать уравнивать или дополнять нужные участки, данный алгоритм находится в разработке, он и является основополагающим всего приложения, так как с одной стороны позволит выделить действительно нужные характеристики объектов для построения псевдослучайного ландшафта, а с другой стороны позволит менять в соответствии с этими характеристиками карту высот, данная часть будет полезна в дальнейшем для разработчиков и дизайнеров для автоматизации создания не только рельефа, но также и готового живого мира.

Выводы из сравнения.

Таблица сравнения явно указывает на несколько проблем, присущих данным рассмотренным приложениям. Первой проблемой API, то есть приложение можно использовать только для выгрузки готовых трехмерных моделей, таким образом программистам, желающим использовать постоянную генерацию карт в своих приложениях приходится писать свои алгоритмы для

нее. Следующей важной проблемой является изолированность приложений от входных данных, ни одно не поддерживает генерацию ландшафта на основе пользовательских объектов, а это является не маловажной частью для получения естественной карты, так как постфактум придется в другом приложении изменять карту для расположения домов, деревьев и других возможных объектов.

Таким образом, разрабатываемое приложение должно обладать возможностью экспорта, предоставлять полный или частичный бесплатный доступ, работать без дополнительного ПО, предоставлять готовую карту на основе пользовательских трехмерных моделей, предоставлять API для получения карты внутри программного кода, а также позволять экспортировать готовую модель в различных, общепринятых форматах.