Санкт-Петербургский государственный университет

Гимранова Рената Маратовна

Выпускная квалификационная работа

Методы генерации ландшафта карт в разработке игр на примере Unreal Engine 4

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Математическое и программное обеспечение вычислительных машин»

Научный руководитель:
Кандидат технических наук,
доцент кафедры технологии программирования
Блеканов Иван Станиславович

Рецензент:
Кандидат физико-математических наук,
доцент кафедры механики управляемого движения
Шиманчук Дмитрий Викторович

Санкт-Петербург
2020

Содержание

Введение

Игровая индустрия является одной из самых больших и важных составляющих современной индустрии развлечений. Влияние и вовлечение потребителя в интерактивное окружение с помощью видеоигр не остается незамеченным уже много лет, благодаря чему сегмент компьютерных игр выделяется среди других видов развлечений. Индустрия видеоигр - крупный сектор экономики, который неразрывно связан с индустрией производства компонентов для персональных компьютеров. Важная часть индустрии — это непосредственно разработка игр. Создание игры является частью жизненного цикла продукта, состоящего из производства, распространения и использования его потребителем.

Разработка игры — сложный процесс, требующий слаженной работы всей команды, которая может включать в себя множество специалистов разных профессий: аналитиков, программистов, дизайнеров и других. Успешные проекты могут быть реализованы как большой командной, которая может позволить себе потратить на разработку большие деньги, так и независимыми энтузиастами, чему способствует появление бесплатных платформ для разработки, площадок для краудфандинга и возможностей для распространения своего продукта.

Одним из важных этапов разработки любой игры является разработка уровня или уровней, на которых будет происходить игровой процесс. Обычно этим занимаются дизайнеры уровней под руководством дизайнеров игры. Таким образом, вид и даже стиль игровой карты могут меняться в зависимости от жанра и стилистики игры. Например, для игры в двухмерном пространстве, разрабатываемой в стиле рисованной мультипликации, нет смысла создавать трехмерный уровень и полностью заполнять его контентом. С развитием технологий во многих играх разных жанров начала

использоваться реалистичная графика, позволяющая игроку более глубоко погружаться в игровой процесс. Как следствие, необходимо создание реалистичного игрового уровня.

Основанием уровня служит его ландшафт. Такие игровые движки, как Unity или Unreal Engine, предлагают готовые средства для его “лепки”. Эти средства представляют из себя набор инструментов скульптинга (например, вытягивание, сглаживание, эрозия, распрямление и т.п.). Благодаря данным инструментам, человек, отвечающий за создание уровня, может создать ландшафт нужного вида, который будет использоваться в дальнейшей разработке. Однако, участие человека в создании ландшафта требует определенных временных затрат в связи с тем, что использование вышеупомянутых средств требует мануальной работы с ними. В случаях, требующих быстрого создания уровня для дальнейшей разработки или даже для последующей игры на нем можно использовать процедурную генерацию — автоматическое создание контента с помощью алгоритмов. При разработке игр процедурная генерация используется для генерации материалов и текстур или для расположения объектов на игровом уровне и наполнения его контентом. Также ее можно применить и для генерации ландшафта. В качестве основного алгоритма для процедурной генерации ландшафта можно выбрать множество алгоритмов, например, создание шума.

Постановка задачи

Целью данной работы является реализация решения для процедурной генерации ландшафта для использования при разработке игр на движке Unreal Engine 4 [16].

Сгенерированный с помощью такого решения ландшафт должен представлять из себя участок земли, который содержал бы на себе возвышенности и углубления, но который при этом позволял бы строить систему навигации для перемещения по данному ландшафту искусственного интеллекта и размещать на себе ключевые для игрового процесса объекты, или участок земли, который можно было бы использовать как базу для последующего создания полноценного уровня на нем.

Задачи работы

Для достижения цели были поставлены следующие задачи:

- Провести обзор существующих решений;
- Провести обзор алгоритмов, использование которых возможно для процедурной генерации ландшафта;
- Реализовать выбранные алгоритмы;
- Провести тестирование реализованных алгоритмов и сравнить качество их работы.

Глава 1. Обзор предметной области

1.1 Игровая индустрия как сфера научной деятельности

С развитием технологий игровая индустрия начала оказывать большое влияние на многие сферы жизни людей, начиная отраслью развлечений и заканчивая экономикой. Разработка игр рассматривается как задача не только для разработчика, способного придумать и запрограммировать необходимые механики, но и для ряда других специалистов.

Как следствие, при разработке игры любой сложности команда разработчиков может столкнуться с рядом проблем. Разным аспектам разработки посвящают литературу и статьи, которые публикуются, например, в журналах, посвященных геймдеву, и на научных ресурсах.

Многие исследования, освещающие игровую индустрию, посвящены не только проблемам, связанным непосредственно с программированием, но и другим сферам, затрагивающим разработку. Например, одним из рецензируемых журналов, в котором публикуются теоретические и эмпирические исследования об играх и культуре, окружающей их, является Games and Culture, выпускаемый американской издательской компанией Sage Journals. Журнал освещает политические, экономические и социальные стороны игровой индустрии с разных точек зрения. Множество статей оттуда освещают психологические и культурные аспекты сферы разработки игр и людей, жизнь которых тем или иным способом связана с ней; например, проблемы этики и моральные выборы, предоставляемые игрокам освещаются Ф.Г. Босманом в его статье [2]. Некоторые статьи могут также помочь разработчикам понять то, как разные игровые механики могут повлиять на опыт игрока в будущем — например, насколько интуитивно понятны будут действия, которые необходимо совершить, и сколько времени они займут [3].

Руководствуясь подобными материалами, подтвержденными экспериментально или научно, команда разработчиков будет иметь возможность принять те или иные решения.

Не так много журналов и издательств посвящено техническим проблемам, с которым могут столкнуться разработчики в процессе создания игры, однако, многие задачи освещаются статьями, публикуемыми на научных ресурсах. Так, например, упоминаются проблемы с производительностью графических интерфейсов и решения этих проблем при написании игровых движков с применением различных языков программирования [10].

Посвященные отдельным системам игры статьи, написанные несколько лет назад, до сих пор остаются актуальными. На системе искусственного интеллекта, написанного для игры F.E.A.R., вышедшей в 2005 году, благодаря статье [7], написанной программистом, принимавшим участие в создании этого ИИ, можно почерпнуть вдохновение для собственного ИИ. Несмотря на использование конечного автомата с тремя внутренними состояниями, разработчикам удалось достигнуть реалистичного поведения ИИ, благодаря использованию планирования и правильно сформированного списка действий, которые интеллект может предпринять.

Проблемам, связанным с многопользовательскими играми, подразумевающими под собой использование сети Интернет, также посвящено множество статей. Некоторые из них, например, разбирают такую проблему, как задержка на стороне клиентов, которыми в данном случае являются игроки. Ян Бернье в своей статье, посвященной этому [1], рассматривает базовую архитектуру игры, основанной на модели Клиент/Сервер и предлагает в качестве решения вышеупомянутой проблемы метод Lag Compensation.

Кроме публикации большого количества материалов, посвященных разработке игр, в некоторых странах университеты вводят образовательные программы, посвященные данному направлению. Студенты, поступившие на подобные программы, могут сфокусироваться на изучении проектирования видеоигр (например, на геймдизайне и методах создания контента) или непосредственно на разработке игр (в этом случае они могут заняться написанием собственных игровых движков, систем искусственного интеллекта и т.п.).

Итак, сфера разработки игр сейчас достаточно влиятельна и популярна для ее освещения в различных источниках с разных точек зрения и с разными подходами.

1.2 Обзор существующих решений

В связи с актуальностью направления и востребованностью наличия процедурной генерации при разработке игр многих жанров, существуют решения, представленные в открытом доступе или доступные для покупки, которые можно применить для работы с ландшафтом и его генерацией в Unreal Engine 4 (далее UE4). Они созданы пользователями или командами разработчиков, не принимающими участие в разработке игрового движка и реализованы в виде подключаемых к нужному проекту плагинов, позволяющих после подключения использовать свой функционал.

Voxel Plugin [17] предлагает систему генерации объемного ландшафта с возможностью использования системы визуального программирования UE4 Blueprints. Данная реализация позволяет изменять ландшафт в рантайме. Плагин реализован в двух версиях - базовой и про. Базовая версия бесплатная и открывает доступ к базовым функциям плагина. Улучшенный функционал (как, например, возможность импорта карт высот, мешей и ландшафтов, размещение геймплейных объектов) становится возможен при покупке про версии.

На официальном сайте, размещающим пользовательские решения для движка, Unreal Engine Marketplace, пользователь Rockam предлагает плагин [20], основанный на использовании функций шума, позволяющий процедурно генерировать ландшафт на плоскости, шаре или кубе. После генерации ландшафта можно также процедурно расположить на нем указанные меши. В случае, если не предполагается необходимость изменять ландшафт во время игры, его можно сделать статическим мешем для повышения производительности. Однако, данное решение является платным. Кроме того, такой функционал движка, как Level Of Details, позволяющий

загружать контент постепенно в зависимости от того, что находится в поле зрения игрока, не поддерживается сгенерированными мешами.

Аналогичный функционал предлагает плагин, размещенный Isara Tech на том же сайте [19]. Ландшафт также генерируется на основе функций шума, с использованием функций шума можно также расположить меши окружения и разместить акторов, тем или иным образом влияющих на геймплей. Как и предыдущее решение, данный плагин является платным.

На Marketplace можно найти еще несколько решений, предлагающих аналогичный функционал, однако, все они являются платными.

В открытом доступе находится плагин cashgenUE [18], реализованный пользователем midgen, позволяющий процедурно генерировать ландшафт с использованием функций шума, поддерживающий Level of Details и имеющий возможность сгенерировать карту глубин для построения водоемов. Для расположения ключевых для игры объектов при использовании этого плагина нужно было бы модернизировать его или добавить соответствующий функционал.

Глава 2. Процедурная генерация ландшафта

В соответствии с поставленной задачей полученный ландшафт должен позволить размещать на себе дополнительные объекты, необходимые для дальнейшего игрового процесса, при этом продолжая выглядеть натурально, было решено рассматривать возможность процедурной генерации ландшафта, который не содержал бы в своей структуре навесов, пещер или объемных арок. Их отсутствие может быть исправлено путем добавления соответствующих статических или процедурных с помощью отдельных алгоритмов процедурной генерации после размещения ключевых для геймплея объектов. Подобный подход сводит задачу к тому, чтобы сгенерировать карту высот.

2.1 Генерация карты высот

При построении ландшафта можно руководствоваться тем, что такие объекты, как горные хребты или береговые линии в природе обладают фрактальными свойствами, то есть та или иная часть объекта может в какой-то мере походить на другую. Природные объекты не похожи на идеальные фракталы, однако, факт наличия фрактальных свойств избавляет нас от попыток создания уникальной на 100% структуру, которая отличалась бы от любого другого места сгенерированного ландшафта.

В таком случае можно использовать следующие способы генерации ландшафта [5]:

1. Схемы разделения (subdivision schemes)
2. Faulting
3. Шум

Алгоритмы разделения подразумевают под собой наличие базовой сетки. По данной сетке итеративно строится мозаичная модель, которая с каждой итерацией меняет свою форму. Количество итераций и плавность результата при использовании данных алгоритмов будет зависеть от размера базовой сетки. Эти алгоритмы зачастую используются для сглаживания 3D-моделей (Рис. 1), но их также можно применить для процедурной генерации.

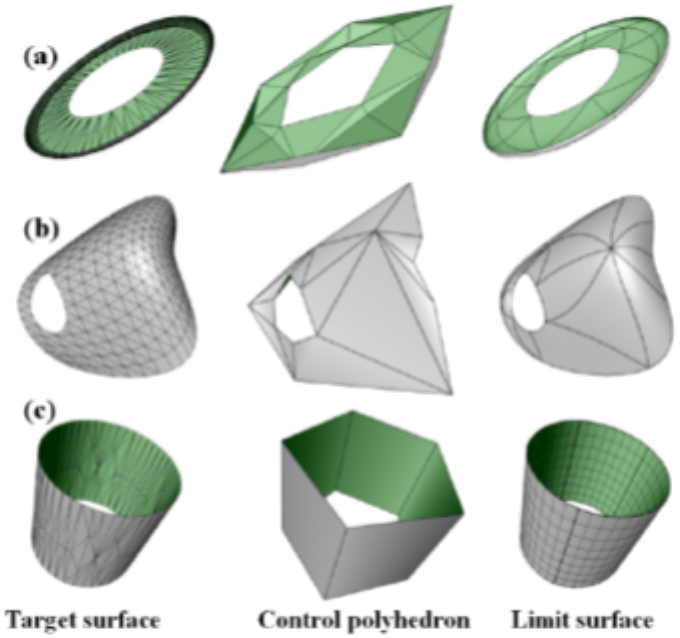

Рис. 1 Пример использования subdivision алгоритмов для сглаживания поверхности [6]

Алгоритмами разделения являются triangle, square и diamond-square алгоритмы, наиболее популярным из которых является последний.

Fault алгоритм заключаются в том, что плоскость разделяется на две части случайной прямой. В общем случае, значения высот для точек, оказавшихся с одной стороны от прямой, понижаются, а значения с другой стороны повышаются. Результат алгоритма зависит от количества итераций, которое возможно задать вручную.

Модификация алгоритма возможна путем выбора функции, определяющей понижение или повышение значения высоты точки [15] или построением зависимости высоты в точке от ее удаленности от построенной прямой. Другая возможная модификация — использование кривых или окружностей вместо прямых.

Функции шума генерируют случайные числа и располагают их нужным образом. Их использование не ограничивается процедурной генерацией, но они популярны в сфере компьютерной графики в связи с тем, что с помощью шума можно получить те или иные текстуры.

Самый простой пример шума - белый шум, являющийся набором случайных чисел и переведенный в черно-белое изображение. Очевидно, для генерации ландшафта белый шум не подходит. Для этой цели чаще всего используют градиентный или клеточный шумы.

При генерации градиентного шума создается решетка случайных градиентов, к которым для получения значения функции шума в данной точке применяется функция интерполяции. Наиболее популярными алгоритмами градиентного шума являются шум Перлина и симплекс шум.

Клеточный шум основывается на составлении поля расстояний до ближайших из множества опорных точек [11]. Примером такого шума является шум Вороного, основанный на диаграммах Вороного.

В качестве алгоритмов для реализации были выбраны следующие методы:

- Алгоритм Diamond-Square
- Fault алгоритм
- Шум Перлина

2.2 Unreal Engine Landscape API и Procedural Mesh Component

Прежде чем перейти к описанию реализованных алгоритмов, необходимо упомянуть возможности использования сгенерированной карты высот в Unreal Engine 4 (далее UE4).

UE4 обладает возможность загрузки собственных карт высот в движок для последующей работы с ними (Рис. 2) [12]. Карта высот должна представлять собой 16-битное черно-белое изображение формата .png или .raw, где белый цвет характеризует наиболее высокую точку ландшафта, черный — самую низкую, а оттенки серого будут переводится в значения между ними. Heightmap Resolution определяется размером загружаемого изображения. Material позволяет предварительно выбрать текстуру, накладываемую на ландшафт при его создании. Location, Rotation и Scale позволяют определять положение создаваемого ландшафта в системе координат уровня и его размер (с помощью z-координаты Scale можно изменить конечную высоту и глубину ландшафта). Section Size определяет, из скольких квадратных полигонов будет состоять одна часть ландшафта, число частей задается ниже.

После импорта выбранного изображения карты высот в UE4, движок генерирует ALandscape, объект, наследующий класс AActor, который определяет возможность объекта быть размещенным на уровне. Основанный на импортированной карте высот Landscape автоматически выстраивается так, чтобы ей удовлетворять. Landscape представляет собой меш из полигонов, количество которых равны Selection Size * Total Components. В зависимости от выбора этих параметров, ландшафт будет выглядеть более гладко или более угловато.

Удобство данного инструмента не подвергается сомнениям, однако, в версии движка 4.22, которая используется при написании данной работы, Landscape API нельзя было использовать в Runtime, то есть непосредственно при игровом процессе. Создание нового ALandscape или изменение уже созданного также невозможно во время игрового процесса в связи с

Рис. 2 Импорт карты высот в Unreal Engine 4

ограничениями игрового движка. Таким образом, подобный способ генерации подойдет в качестве облегчения работы дизайнерам уровней при разработке. Он позволит создать базу для дальнейшей работы над ландшафтом и уровнем.

В случае, когда необходимо сгенерировать ландшафт в начале игрового процесса для дальнейшей игры на нем или когда необходимо изменять его во время игры, можно использовать Procedural Mesh Component [14]. Этот компонент позволяет создавать пользовательские геометрические объекты, созданные с помощью треугольных мешей, объединенных в полигоны. Для создания Procedural Mesh Component необходимо задать несколько массивов:

- массив с координатами высот квадратов, определяющих полигоны. Для записи координат используется встроенный в UE4 тип FVector;
- массив целых чисел, определяющих треугольники, из которых состоят полигоны;
- массив FVector, определяющий направление нормали каждого полигона (опционально);
- массив, задающий UV-развертку — соответствие между координатами объекта и координатами текстуры, накладываемой на объект (опционально);
- массив, задающий цвет каждой высоты каждого полигона (опционально);
- массив, определяющий касательные к нормалям полигонов (опционально).

С помощью данного компонента можно генерировать меш желаемой формы. В данной работе полученный меш используется в качестве ландшафта. Размер карты будет определяться количеством и размером полигонов.

Использование процедурного меша предполагает под собой его возможное последующее изменение. Это позволяет работать над его формой в течение всего игрового процесса. В случае, когда наличие такого

функционала не подразумевается, для улучшения производительности можно создать на основе процедурного меша статический, то есть такой, геометрический вид которого нельзя будет изменить.

В случае использования процедурного меша необходимо рассматривать ландшафт, как некоторую математическую модель, чтобы на основе базовой сетки полигонов суметь построить ландшафт, основанный на карте высот.

Ландшафт может быть описан функцией $f: \mathfrak{R}^2 \to \mathfrak{R}$, $z = f(x, y)$, где x, y — координаты плоскости, а z — соответствующее значение высоты [9]. Данное определение предполагает под собой отсутствие таких структур, как пещеры и навесы, что удовлетворяет требованиям к результату. Таким образом, достаточно будет получить значение высоты для вершины полигона и записать его в координату z соответствующего элемента массива высот полигонов.

2.3 Реализация алгоритмов процедурной генерации

Итак, для реализации было выбрано 3 алгоритма: алгоритм Diamond-Square, Fault алгоритм и шум Перлина.

Каждый алгоритм был реализован таким образом, чтобы его можно было использовать как для генерации двумерной текстуры для последующего импорта в движок, так и для последующего построения ландшафта на основе Procedural Mesh Component. Для сохранения текстуры в изображение формата .png использовалась размещенная в свободном доступе библиотека stb_write_image [21].

2.3.1 Diamond-Square

Алгоритм Diamond-Square [4] — алгоритм разделения, строящийся на сетке размера $2^n + 1$. Он является двумерной реализацией алгоритма midpoint displacement, в каждой итерации которого высчитывается значение высоты средней между данными точками. Выбор этого алгоритма обоснован интересными результатами при простой реализации. Кроме того, если задать достаточно большие значения высот угловым точком, возможно получить возвышения на краях полученной карты, тем самым визуально создавая ее границы.

В качестве входных параметров используются следующие:

- int size;
- float height.

В случае генерации изображения в качестве size передается его размер. В случае использования Procedural Mesh Component число полигонов по вертикали или по горизонтали. В случае, если $size \neq 2^n + 1$, размер пересчитывается таким образом, чтобы удовлетворять этому условию.

height определяет максимальное значение, которое могут принять угловые точки базовой сетки. Это значение также используется для генерации случайного числа, которое прибавляется к среднему значению угловых точек на каждом шаге.

Работа алгоритма начинается с 2D-сетки. Значения в четырех угловых точках сетки могут быть заданы вручную или случайным образом. Алгоритм заключается в итерационном исполнении шагов diamond и square:

- шаг diamond заключается в том, что для каждого «ромба» сетки находится срединная точка. В качестве значения высоты в этой точке

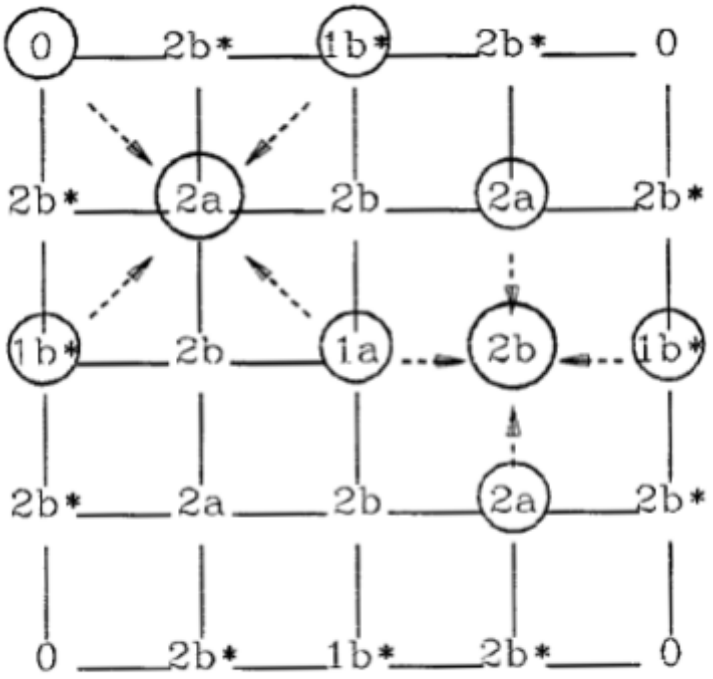

Рис. 3 [4]

устанавливается среднее от окружающих ее угловых точек + случайное число;

- шаг square заключается в том, что срединная точка находится для каждого квадрата сетки. Значение высоты в найденной точке определяется аналогично и равно среднему от окружающих ее точек + случайное число.

Таким образом, порядок нахождения точек на Рис. 3 следующий: угловые точки 0, 1a (срединная точка на первом шаге diamond), точки 1b* (срединные точки на первом шаге square), точки 2a, точки 2b* и т.д. Как можно заметить, на шаге square существуют точки, находящиеся на границах сетки так, что одна из окружающих их точек выходит за сетку. В этом случае, например, можно искать среднее значение только от трех окружающих ее точек или использовать одну из этих точек дважды.

Алгоритм возвращает двумерный массив float, содержащий в себе значения высот.

2.3.2 Fault алгоритм

Для fault алгоритма был реализован его стандартный вариант с модификацией функции, определяющей то, насколько повышается или опускается данная точка.

В качестве входных параметров используются следующие:

- int size;
- float height;
- int iterations.

Как и в алгоритме Diamond-Square параметры size и height определяют размер карты и максимальную высоту ландшафта (как в случае генерации изображения, так и в случае использования Procedural Mesh Component).

Параметр iterations определяет количество итераций, которое должно произойти до прекращения выполнения алгоритма. Чем больше итераций, тем лучше результат (Рис. 4).

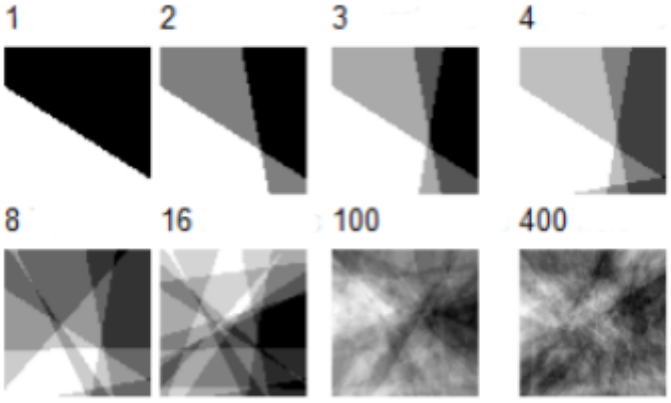

Рис. 4 Зависимость результата от количества итераций [15]

Каждая итерация начинается с получения прямой, которая разделяет плоскость ландшафта. Прямая определяется уравнением $a * x + b * y + c = 0$. Для получения коэффициентов прямой случайным образом выбираются две отличные друг от друга точки, лежащие внутри

данной плоскости или на ее границах. Зная точки, лежащие на прямой, можно найти соответствующие коэффициенты a, b и c.

После нахождения прямой, разделяющей плоскость на две части, необходимо определить положение каждой точки плоскости относительно построенной прямой. Значение высоты в точке будет увеличиваться, если при подстановке ее в полученное уравнение прямой получается значение > 0, и уменьшаться в обратном случае.

Величина, на которую будет изменяться высота, реализована с помощью функции синуса и в текущей реализации равна $\Delta h = amplitude * |sin(a * x + b * y + c) / MaxCornerValue * 2 * \pi|$. Здесь $amplitude$ — множитель для контроля величины получаемого значения, который в общем случае уменьшается с каждой итерацией; x и y — значения координат рассматриваемой точки плоскости; $MaxCornerValue$ — наибольшее из значений угловых точек при данных коэффициентах прямой. Способ вычисления был выбран экспериментальным способом и зависит лишь от потребностей и предпочтений разработчика.

После выполнения заданного количества итераций алгоритм возвращает двумерный массив float, содержащий в себе значения высот.

2.3.3 Шум Перлина

Шум Перлина — градиентный шум, основанный на наборе случайных или псевдослучайных векторов-направлений градиентов, расположенных в точках плоскости, к которым применена функция интерполяции. Важное свойство шума Перлина — это его связность, благодаря которой результат выглядит гладко и не содержит в себе обрывов или ярко выраженных переходов.

Для того, чтобы не жертвовать скоростью при создании шума, автор алгоритма, К. Перлин предлагает использовать заранее сгенерированную таблицу градиентов, для псевдослучайного выбора градиента из этой таблицы [13]. Таким образом, для каждой точки будет постоянно выбираться один и тот же градиент. Реализация шума Перлина в этой работе использует тот же подход, в котором значение градиента выбирается в соответствии с заранее заданной таблицей перестановок.

В качестве интерполирующей функции используется функция линейной интерполяции. Это обусловлено тем, что при генерации шума Перлина в реальном времени использование кубической или косинусной интерполяции может повлиять на производительность процесса, несмотря на то что их функции позволяют получить более сглаженный результат.

В качестве функции, смещающей значение координаты для получение более гладкого результата, используется предложенная Кеном Перлином в [8] функция *smoother step*.

В качестве входных параметров используются следующие:

- int size;
- int octaves.

Как и в случае предыдущих алгоритмов параметр size определяет размер плоскости ландшафта. Параметр height, определяющий максимальное значение координаты z не передается в связи с тем, что функция шума генерирует значения от 0 до 1. В случае с генерацией изображения, это желаемый результат, в случае использования Procedural Mesh Component, достаточно умножить результат на необходимую величину.

Параметр octaves определяет количество октав, которые используются для получения более интересных результатов. Использование октав подразумевает под собой генерацию более детализированного шума меньшей

амплитуды и добавление его к уже сгенерированного шуму. С каждым разом амплитуда уменьшается в два раза, оттуда и название — октава.

Алгоритм возвращает двумерный массив float, содержащий в себе значения высот.

2.4 Расположение ключевых объектов

В случае, когда карта генерируется при запуске игры для дальнейшего игрового процесса на ней, не подразумевая при этом вмешательство дизайнера уровней и расстановку объектов на карте мануально, необходимо определить области, в которых можно расположить ключевые для геймплея объекты. Под ключевым объектом понимается статический (недвижимый) объект, который будет играть определенную роль в игровом процессе и который должен быть доступен для игрока или искусственного интеллекта. Так, в стратегиях распространены механики построения собственных поселений или крепостей, которые используются для найма персонажей, цель которых во время игры определена, например, изучением карты. В таком случае для корректного расположения объектов и обеспечения возможности свободного перемещения игрока и искусственного интеллекта среди этих объектов необходимо выделить такую область, где их можно будет разместить.

Процедурная генерация ландшафта вышеописанными способами не гарантирует постоянное наличие такой области. Было принято решение изменять сгенерированную вышеописанными методами карту высот таким образом, чтобы на ней присутствовали плоские области.

Это реализовано следующим образом:

1. Ищется среднее значение элементов карты высот, сгенерированный одним из реализованных способов;
2. Происходит подсчет элементов, значение которых оказалось ниже найденного среднего значения. В случае, если такие элементы в общей сложности занимают больше, чем половину карты, они не изменяются, а среднее значение заменяется меньшим. Это сделано для того, чтобы сгенерированный ландшафт не терял интересной структуры, выраженной наличием, например, гористой местности;
3. Шаг 2 повторяется с пониженным значением до того момента, пока количество элементов, меньших, чем заданное число, не станет меньше половины;
4. Элементы, оказавшиеся меньше найденного числа, принимают его значение.

Таким образом, можно выделить области, которые потенциально выделялись бы среди возвышений, и сделать их плоскими для дальнейшего расположения на них ключевых для геймплея объектов.

В зависимости от нужд разработчика изменяться может больше или меньше, чем половина карты, или изменить алгоритм таким образом, чтобы нужная площадь ограничивалась и сверху, и снизу. Также можно изменить алгоритм таким образом, что выпрямляться будут возвышения, а не низины.

Для оценки размеров получившейся области и предоставления возможности выбора области, в которой можно разместить объект, были произведены следующие действия:

1. С помощью алгоритма поиска в ширину на основе двумерного массива высот был создан двумерный массив таким образом, что все элементы, значения высот которых отличаются от значения высот полученных плоских областей, равны 0, а элементы, соответствующие элементам

плоских областей, равны целочисленному числу, представляющему собой номер области;

2. Для каждой полученной пронумерованной области оценен прямоугольник, в который можно вписать эту область. Наименьшая из сторон получившегося прямоугольника характеризует размер области, по которому можно определить, можно ли расположить внутри нее тот или иной ключевой объект.

После получения списка областей с их размерами, можно передать методу минимальный размер области, при котором объект можно будет разместить внутри нее, составить список из подходящих областей, а затем, в зависимости от нужд разработчика, разместить объект в каждой подходящей области или в случайной области.

В случае, если подходящей области найдено не было, ее можно создать в случайном месте карты тем же способом, с помощью которого создавались плоские области, т.е. путем нахождения средней высоты и приравнивания к этому значения всех высот в определенном радиусе.

Глава 3. Проведение эксперимента

В связи с необходимостью написания реализации алгоритмов для их использования при работе с Procedural Mesh Component и для генерации карты высот в виде 2D текстуры, эксперименты по сравнению реализованных алгоритмов проводились дважды.

3.1 Генерация 2D текстур и использование Landscape API

Для проверки реализованных алгоритмов при генерации 2D текстур производились следующие действия:

1. Написанный на языке C++ алгоритм использовался для генерации 2D изображения. Засекалось время работы каждого алгоритма;
2. Полученная 2D текстура загружалась в Unreal Engine 4 с помощью соответствующего встроенного в движок функционала. Полученный ландшафт оценивался с помощью встроенных средств на построение навигации для ИИ и на внешний вид. Во время второго шага время не засекалось в связи с тем, что получение ландшафта с помощью карты высот любого вида будет занимать примерно одинаковое время, которое зависит в основном от размера переданного изображения и, соответственно, размера генерируемой карты.

Ниже представлены результаты эксперимента в виде изображений текстуры и снимка экрана с сформированным на ее основе ландшафтом.

Изображения генерировались в разрешении 513x513 пикселей. Выбор такого размера в частности обусловлен наличие у алгоритма Diamond-Square условия, обязывающего базовую сетку иметь размер $2^n + 1$.

Можно привести любой заданный размер к данному путем уменьшения или увеличения заданного значения, а затем нормализации полученного

результата к нужному, однако ради чистоты эксперимента было принято решение исключить лишние операции, связанные с этим.

К ландшафту был применен материал, меняющий накладываемую текстуру на меш текстуру в зависимости от формы меша. Материал был создан с помощью текстур и функций, которые можно найти среди Starter Content — файлов, созданных Epic Games, которые предлагается использовать при создании нового проекта.

3.1.1 Diamond-Square

Рис. 5 2D текстура, полученная с помощью алгоритма Diamond-Square

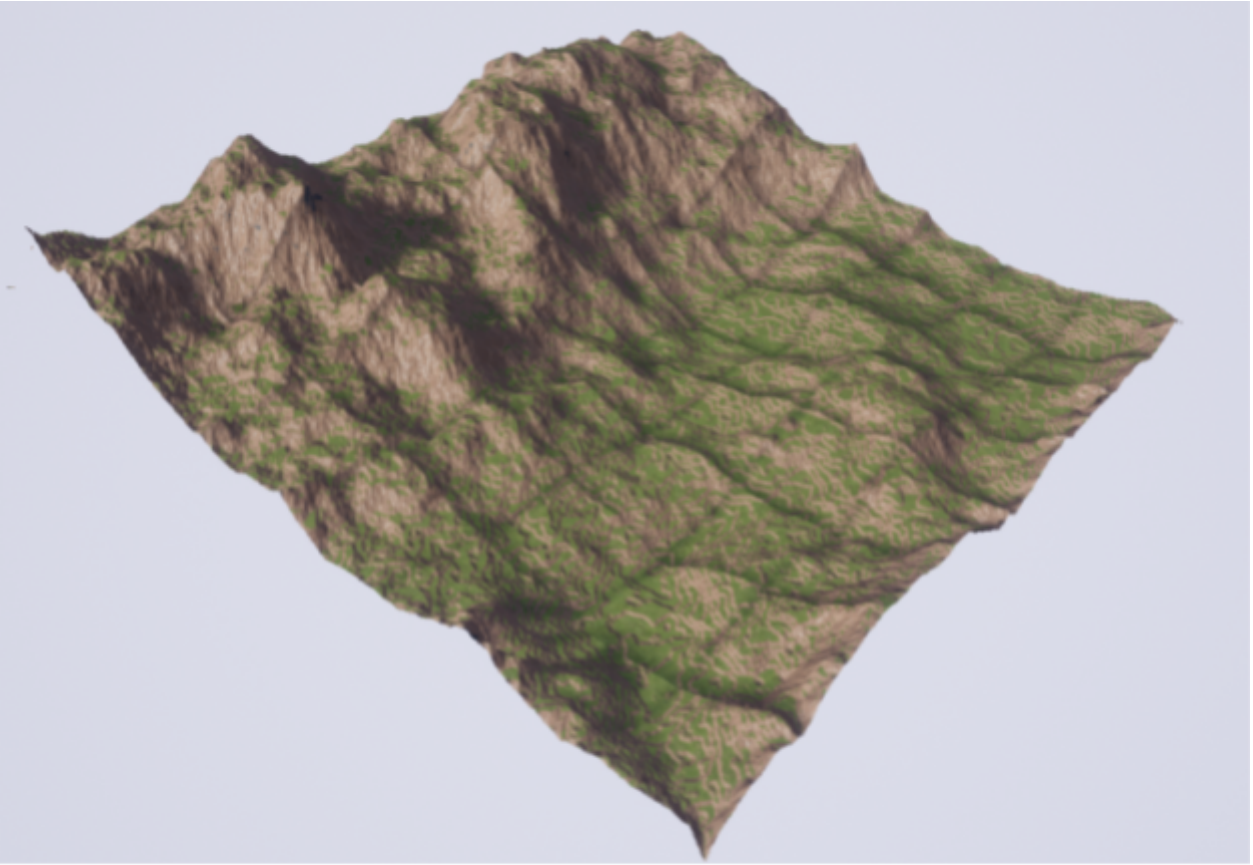

Рис. 6 Ландшафт, полученный на основе 2D текстуры

Среднее время генерации текстуры: 2.7 секунды.

Полученный в результате использования алгоритма Diamond-Square ландшафт удовлетворяет требованию, согласно которому он должен выглядеть реалистично. На Рис. 6 можно увидеть, что благодаря данному алгоритму можно успешно сгенерировать структуру, которая служила бы границей карты — в данном случае для этого служат горы.

Можно заметить, что при построении ландшафта с помощью данного алгоритма он содержит в себе множество небольших холмов и возвышений и, например, равнинные области являются не совсем однородными. Для того, чтобы сделать равнины более пологими и горы более однородными, можно значения высот возвести в квадрат. Такой подход возможен при генерации значений высот в пределах от 0 до 1, что и делается в данном случае. Для сглаживания самого ландшафта нужно будет увеличить количество полигонов.

3.1.2 Fault алгоритм

Рис. 7 2D текстура, полученная с помощью Fault алгоритма при количестве итераций = 150

Среднее время генерации текстуры (количество итераций = 150): 2.7 секунды.

Количество итераций было выбрано вручную так, чтобы время генерации текстуры примерно совпадало со времени генерации при использовании алгоритма Diamond-Square для более честного сравнения алгоритмов. Для понимания разницы между меньшим и большим количеством итераций, однако, было также произведено тестирования алгоритма при количестве итераций = 500 (см. Рис. 9, Рис. 10).

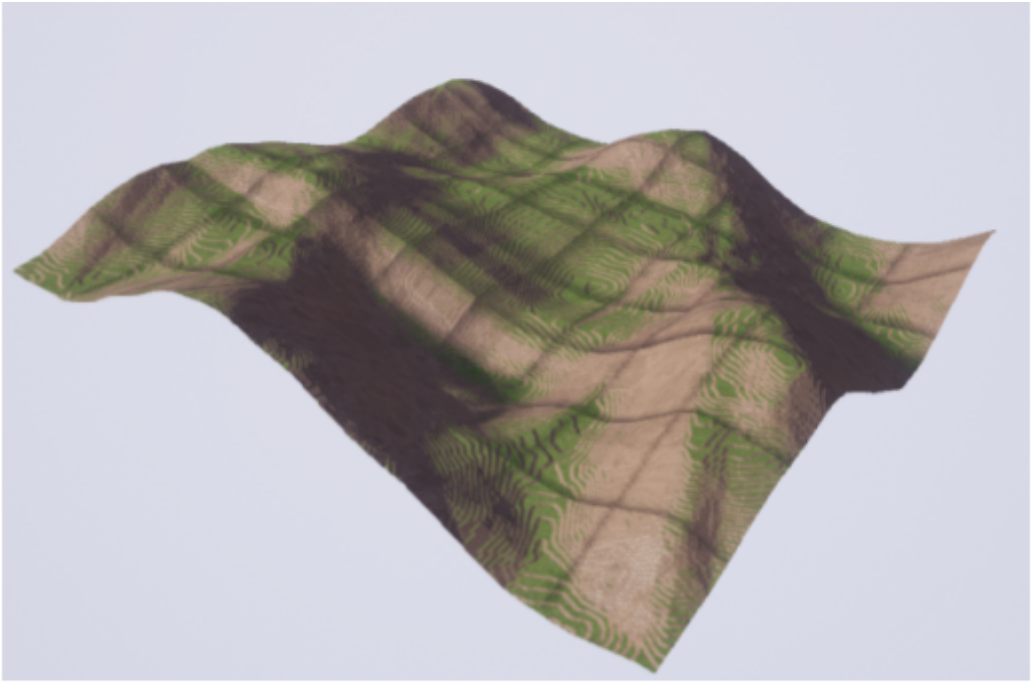

Рис. 8 Ландшафт, полученный на основе 2D текстуры

Рис. 9 2D текстура, полученная с помощью Fault алгоритма при количестве итераций = 500

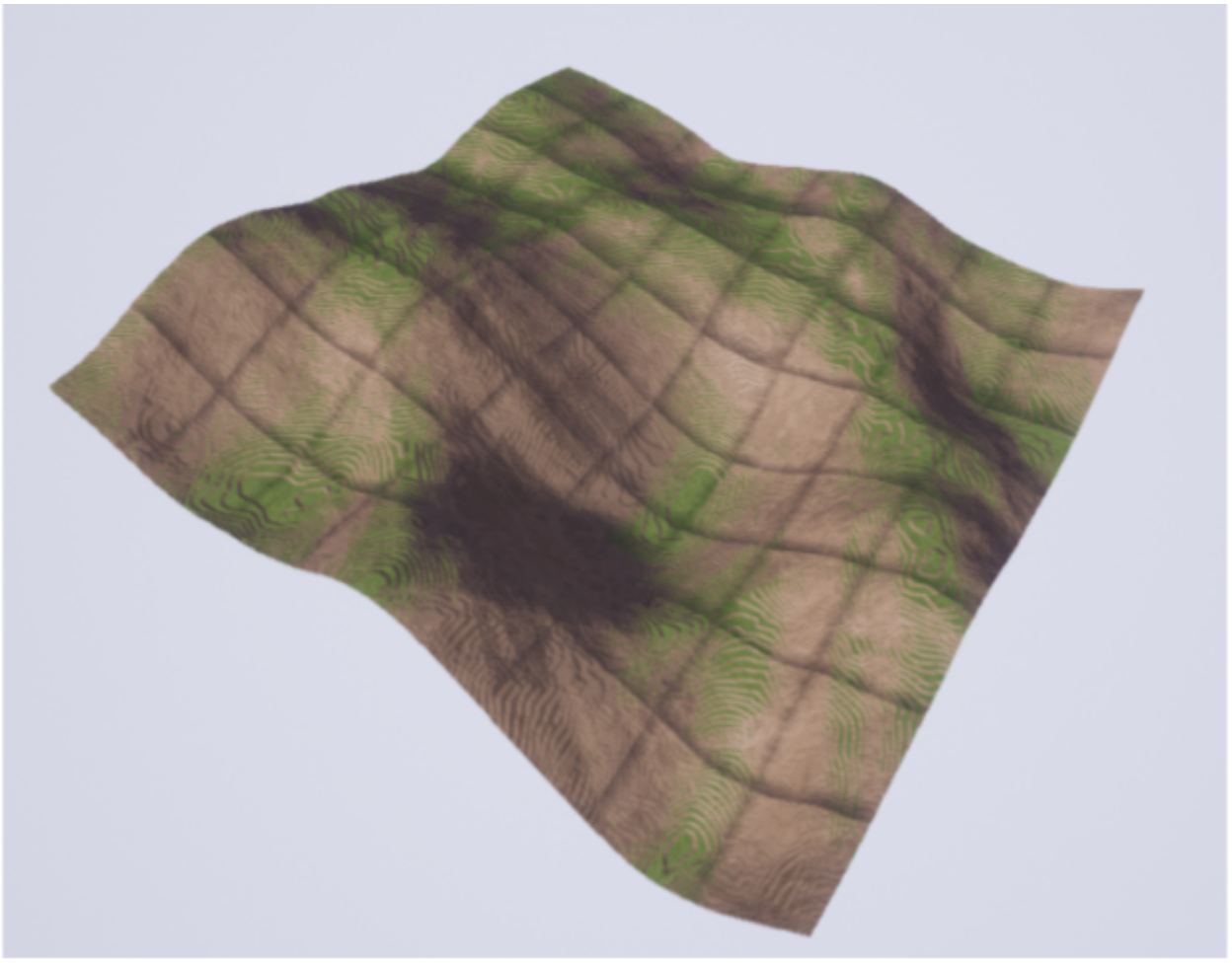

Рис. 10 Ландшафт, полученный на основе 2D текстуры

Среднее время генерации текстуры (количество итераций = 500): 9.2 секунды.

Очевидно, что качество результата повышается вместе с увеличением количества итераций. В основном это можно увидеть, посмотрев на сгенерированные при разном количестве итераций текстуры — так, при 150 итерациях на текстуре видны несглаженные полосы. При увеличении количества итераций карта высот будет включать в себя области с более низким значением высоты.

При сравнении результатов генерации с помощью алгоритмов Faulting и Diamond-Square видно, что Fault алгоритм предоставляет более реалистичные результаты в том смысле, что структура ландшафта менее однородна. В основном это обусловлено тем, что количество итераций в Diamond-Square алгоритме ограничено размерами базовой сетки. Однако, при

создании игры в определенном жанре, например, при создании стратегии, важно учитывать необходимость построения системы навигации для искусственного интеллекта. В этом случае для корректного построения такой системы будут в приоритете плавность и однородность полученного ландшафта.

3.1.3 Шум Перлина

Рис. 11 2D текстура, полученная с помощью шума Перлина при количестве октав = 8

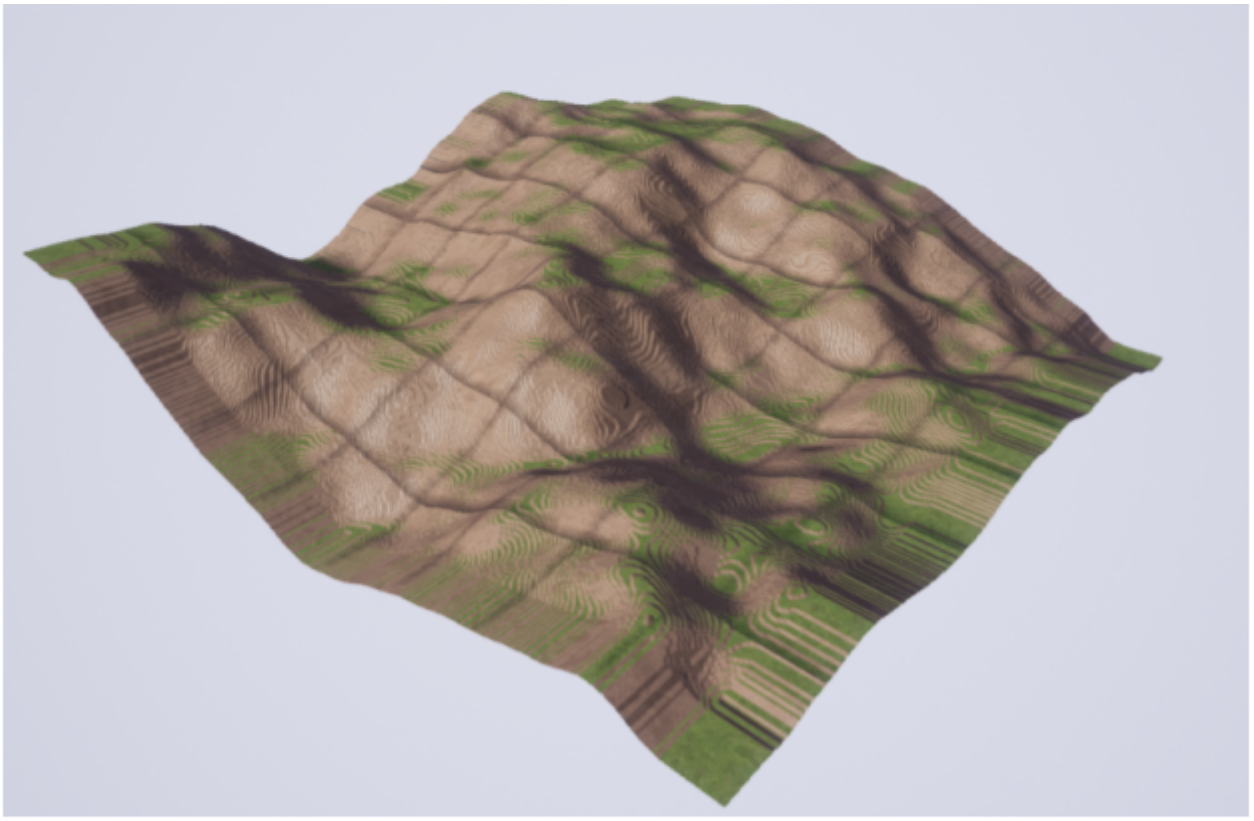

Рис. 12 Ландшафт, полученный на основе 2D текстуры

Среднее время генерации текстуры (при количестве октав = 4): 1.8 секунды. Среднее время генерации текстуры (при количестве октав = 8): 2.8 секунды.

Количество октав обусловлено тем, что при значении от 4 до 8, получаются хорошие результаты. Вынесены результаты генерации при количестве октав = 8, так как среднее время генерации текстуры практически совпадает со средним временем генерации при использовании других методов.

Видно, что полученный ландшафт похож по своей структуре на ландшафт, сгенерированный с помощью Fault алгоритма. Однако, шум Перлина позволяет получить результат, который гарантируют 500 итераций Fault алгоритма, за более короткое время.

3.2 Генерация ландшафта с применением Procedural Mesh Component

Для проверки реализованных алгоритмов при использовании Procedural Mesh Component производились следующие действия:

1. Написанный на языке C++ с использованием типов данных и функций, встроенных в Unreal Engine 4, алгоритм использовался для генерации 2D массива значений высот;
2. Полученный массив передавался в качестве массива высот для создания полигонов для процедурного меша.

Таким образом, время генерации карты высот не отличается от время генерации при использовании 2D текстуры. Однако, время работы каждого алгоритма засекалось для того, чтобы выяснить как будет отличаться время создания плоских областей для размещения на них ключевых для геймплея объектов.

Ниже представлены результаты эксперимента в виде снимков экрана с процедурном мешем, служащим в качестве ландшафта.

Аналогично с предыдущим экспериментом, генерировался меш с размерам 513x513 полигонов. Размер меша можно изменять с помощью изменения размера полигонов, в случае данного эксперимента он был равен 150 единицам.

Также, по аналогии с предыдущим экспериментов, к мешу применялся материал, меняющий вид накладываемой текстуры в зависимости от формы меша. Для визуального выделения точек, в которых можно разместить ключевые объекты, используется функция движка DrawDebugPoint(). Таким образом, на снимках экрана они будут представлены в виде красных точек.

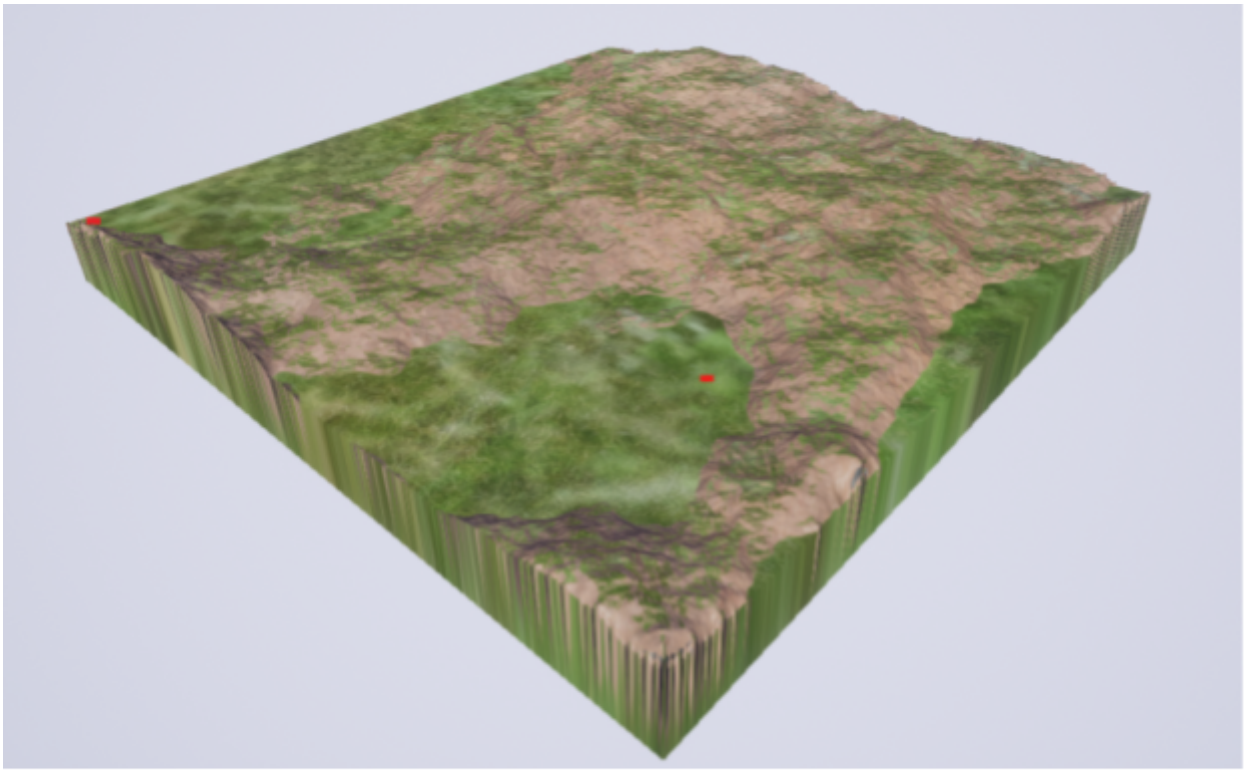

Рис. 13 Результат применения Diamond-Square алгоритма

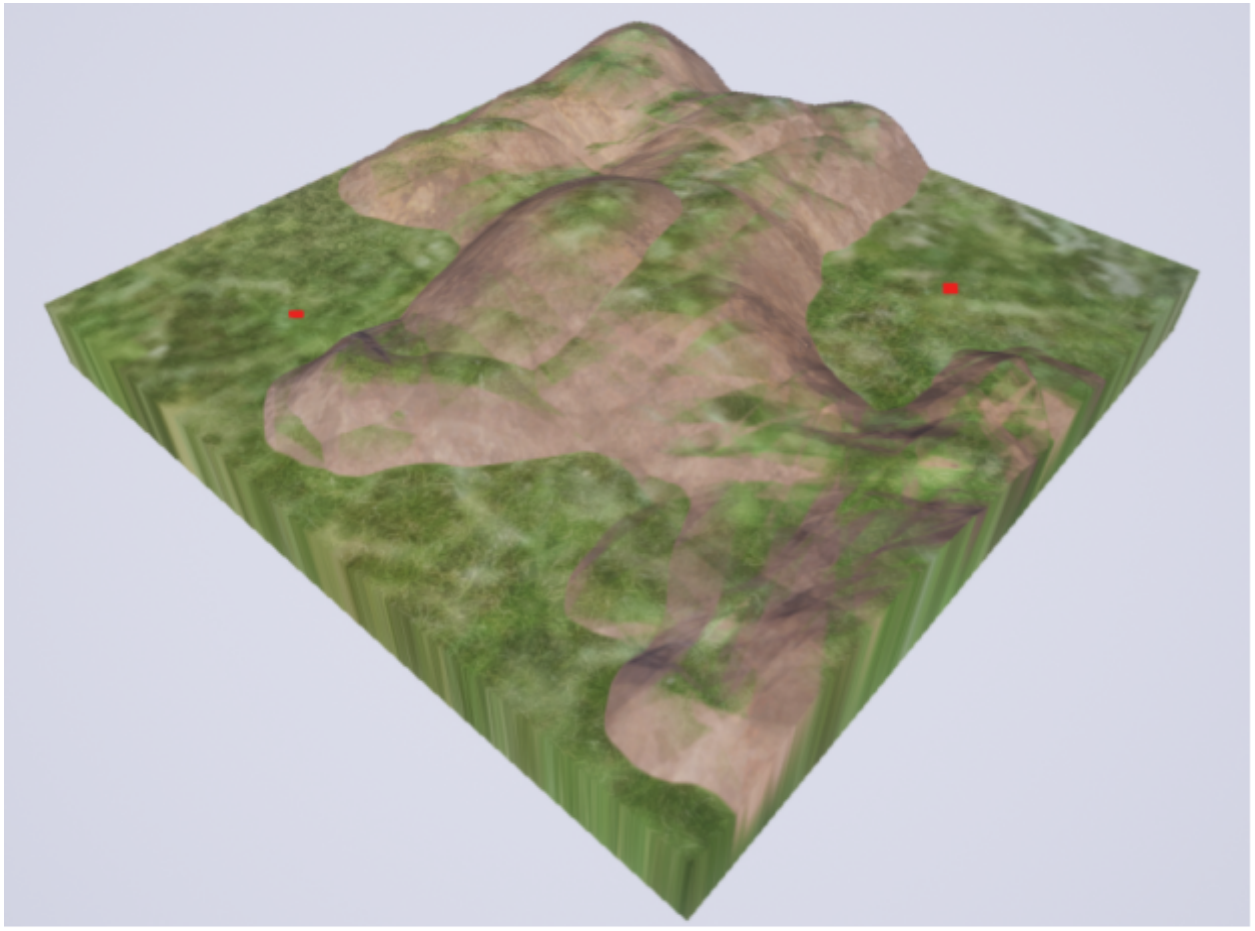

Рис. 14 Результат применения Fault алгоритма

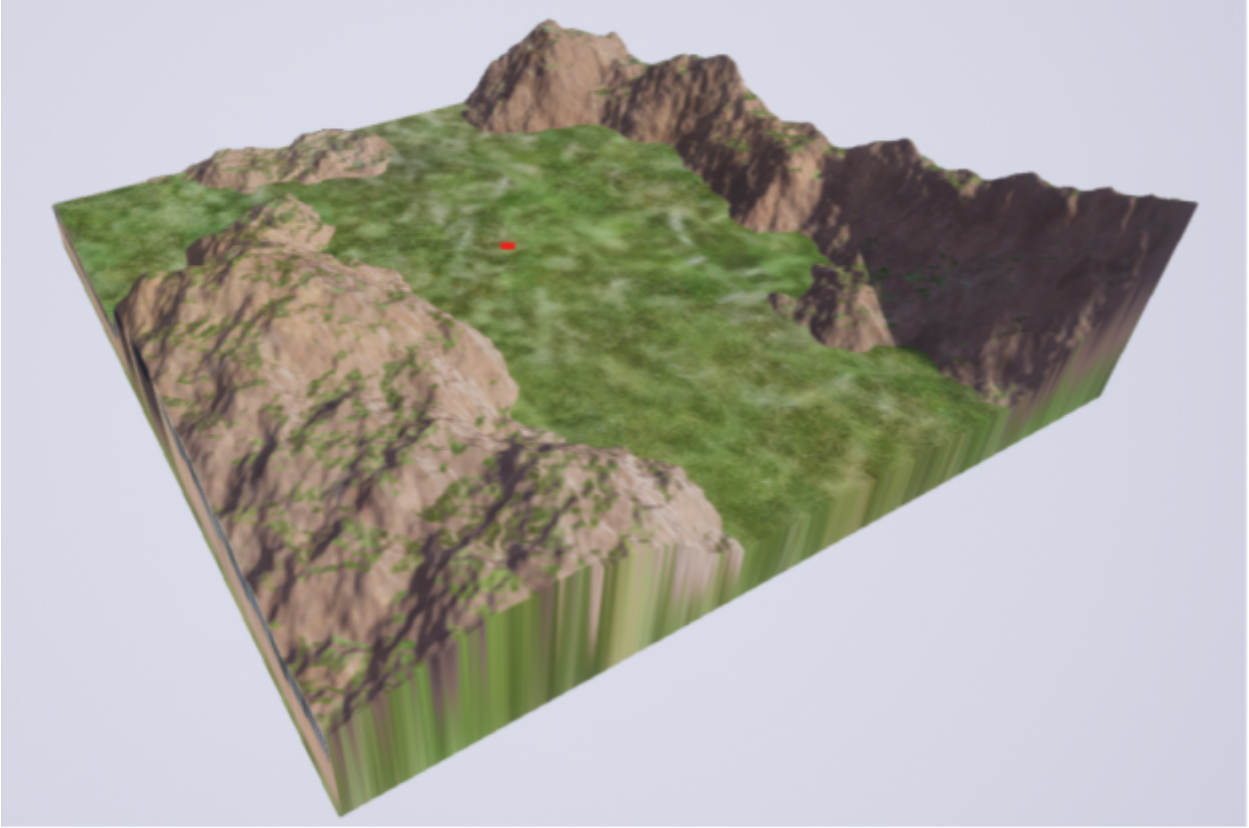

Рис. 15 Результат применения шума Перлина

В таблице ниже приведено среднее время генерации карты высот (учитывающее время, которое требуется алгоритму для нахождения и создания плоских областей). Оно заведомо ниже времени генерации 2D текстур, в частности в связи с хорошей оптимизацией движка.

Таблица 1. Результаты эксперимента по генерации ландшафта

Алгоритм	Время генерации
Diamond-Square	230 мс
Fault алгоритм	2200 мс
Шум Перлина	210 мс

3.3 Выводы

По результатам экспериментов можно выделить определенные плюсы и минусы реализованных алгоритмов.

Алгоритм Diamond-Square является хорошим решением для генерации граничных мешей. Он также обладает относительно быстрой скоростью. Задавая нужные значения высот для границ базовой сетки, можно продолжить генерировать ландшафт бесконечно. Игры, требующие наличия как можно более реалистичного ландшафта, могут воспользоваться этим алгоритмом. Однако, при необходимости генерации более однородной поверхности необходимо будет или увеличить количество полигонов, что повлияет на производительность, или воспользоваться другим алгоритмом.

Шум Перлина обладает высокой производительностью. Благодаря особенностям шума, ландшафт, сгенерированный с помощью шума Перлина, может быть бесконечно продолжен. Основную форму и структуру ландшафта можно менять с помощью октав (чем меньше их количество, тем плавнее и однороднее будет результат).

Fault алгоритм показал худшие результаты из реализованных алгоритмов. Несмотря на простую реализацию алгоритма, нельзя не замечать времени, которое требуется для генерации карты высот. Однако, это может быть недостатком функции, определяющей то, как понизится или повысится значение высоты в точке. В данной реализации для получения более интересных результатов для этого использовалась функция sin. При использовании более быстрой функции эта проблема может быть решена.

Таким образом, из реализованных алгоритмов лучшими вариантами для генерации ландшафта в реальном времени будут шум Перлина и алгоритм Diamond-Square. С помощью обоих методов ландшафт можно генерировать бесконечно, не затрачивая при этом на генерацию много

времени. При правильном задании размеров генерируемой структуры, генерация будет происходить, не мешая игровому процессу.

Для генерации карты высот в виде 2D текстуры подходит практически любой алгоритм. Критериями выбора алгоритма в этом случае будут размер текстуры и желаемая структура ландшафта.

Заключение

В рамках проведенной работы были выполнены следующие задачи:

- Путем обзора научной деятельности в сфере игровой индустрии показана актуальность работы;
- Проведен обзор существующих решений;
- Проведен обзор основных групп алгоритмов, применимых для решения данной задачи;
- Рассмотрены 2 способа решения задачи: генерация 2D текстуры и создание ландшафта на ее основе, генерация ландшафта с применением Procedural Mesh Component;
- Выбраны, реализованы и оценены 3 алгоритма.

При дальнейшей работе над алгоритмом планируется:

- Реализовать другие алгоритмы генерации карты высот, например, симплекс-шум (улучшенная версия шума Перлина) или шум Вороного;
- Процедурная генерация окружения — расположение определенных мешей, задающих местность, например, травы, деревьев, камней;
- Усложнение геометрии ландшафта, например, генерация пещер и каменных навесов;
- Добавление возможности генерации биомов — природно-климатический зон, которые будут определяться набором определенных параметров таких как высота, объекты, характеризующие эту зону, текстуры, накладывающиеся на ландшафт и т.п.

Список литературы

[1] Bernier, Y.W. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. // Game Developers Conference. - 2001.

[2] Bosman, F. G. There Is No Solution!: “Wicked Problems” in Digital Games. // Games and Culture. - 2019. - №5. - С. 543-559.

[3] Costello, B. M. The Rhythm of Game Interactions: Player Experience and Rhythm in Minecraft and Don’t Starve // Games and Culture. - 2018. - №8. - С. 807-824.

[4] Fournier A., Fussell D., Carpenter L. Computer rendering of stochastic models. // Commun. ACM 25. - 1982. - №6. - С. 371–384.

[5] Galin E., Guérin E., Peytavie A., Cordonnier G., Cani M.-P., Benes B., Gain J. A Review of Digital Terrain Modeling // Computer Graphics Forum. - 2019. - №2. - С. 553-577.

[6] Lavoue G., Dupont F., Baskurt A. Toward a near optimal quad/triangle subdivision surface fitting // Fifth International Conference on 3-D Digital Imaging and Modeling. - 2005. - С. 402-409.

[7] Orkin J. Three States and a Plan: The A.I. of F.E.A.R // Game Developers Conference. - 2006.

[8] Perlin K. Improving Noise // ACM Trans. Graph.. - 2002. - №3. - С. 681–682.

[9] Thalmann D., Musse S. R. Crowd Simulation. - Berlin, Heidelberg: Springer-Verlag, 2007. - 254 с.

[10] Wang, Y. - H., Wu, I. - C., Jiang, J. - Y. A portable AWT/Swing architecture for Java game development. // Softw. Pract. Exper.. - 2007. - №37. - С. 727-745.

[11] Клеточный шум // The Book of Shaders URL: https://thebookofshaders.com/12/?lan=ru (дата обращения: 23.04.2020).

[12] Creating and Using Custom Heightmaps and Layers // Unreal Engine 4 Documentation

URL: https://docs.unrealengine.com/en-US/Engine/Landscape/Custom/index.html (дата обращения: 23.04.2020).

[13] Noise Machine URL: http://noisemachine.com/talk1/ (дата обращения: 23.04.2020).

[14] Procedural Mesh // Unreal Engine 4 Documentation URL: https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/index.html (дата обращения: 23.04.2020).

[15] Terrain Tutorial. Two Possible Variations // lighthouse3d.com URL: http://www.lighthouse3d.com/opengl/terrain/index.php?faultvar (дата обращения: 23.04.2020).

[16] Официальный сайт Unreal Engine // URL: https://unrealengine.com (дата обращения: 23.04.2020).

[17] Официальный сайт Voxel Plugin URL: https://voxelplugin.com/ (дата обращения: 23.04.2020).

[18] CashgenUE by midgen // github URL: https://github.com/midgen/cashgenUE (дата обращения: 23.04.2020).

[19] Procedural Landscape Generator by Isara Tech // Unreal Marketplace URL: https://www.unrealengine.com/marketplace/en-US/product/procedural-landscape-generator (дата обращения: 23.04.2020).

[20] Procedural Terrain Generator by Rockam // Unreal Marketplace URL: https://www.unrealengine.com/marketplace/en-US/product/procedural-terrain-generator (дата обращения: 23.04.2020).

[21] stb_write_image // github URL: https://github.com/nothings/stb (дата обращения: 23.04.2020).