Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК
Зав. кафедрой

_____ /И.В. Нечта/

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

по направлению 09.03.01 «Информатика и вычислительная техника»
профиль Программное обеспечение средств вычислительной техники и автоматизированных систем

Разработка модуля генерации ландшафта для Unity3d

Студент _____/ Харченко Евгений Викторович /

Институт информатики и вычислительной техники

Группа - ИП-012

Руководитель _____/ Павлова Ульяна Владимировна /

Новосибирск 2024 г.

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

СТУДЕНТА Харченко Е.В. ГРУППЫ ИП-012

УТВЕРЖДАЮ
«_____» ___
зав. кафедрой ПМиК

_____/ _____/

Новосибирск 2024 г.

1. Тема выпускной квалификационной работы бакалавра

Разработка модуля генерации ландшафта для Unity3d

утверждена приказом СибГУТИ от «26» января 2024 г. № 4/113о-24

2.Срок сдачи студентом законченной работы «20» июня 2024 г.

3.Исходные данные к работе

1 Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. Классика Computers Science [Текст] / Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. — 4-е изд.. — СПб: Питер, 2012 — 784 с.

2 Unity Documentation / [Электронный ресурс] // Unity - Scripting API : [сайт]. — URL: https://docs.unity3d.com/ScriptReference/index.html

4.Содержание пояснительной записки (перечень подлежащих разработке вопросов)	**Сроки выполнения по разделам**
Постановка задачи	26.01.24-22.02.24
Теоретическая часть	23.02.24-19.03.24
Определение средств для разработки	20.03.24-31.03.24
Реализация алгоритма	01.04.24-30.04.24
Тестирование	02.05.24-19.05.24
Заключение	20.05.24-05.06.24
Список литературы	05.06.24-20.06.24

Консультанты по ВКР

(фамилия И.О., должность, ученая степень, ученое звание)

Задание выдано:
« 26 » января 2024 г.

_____ / Павлова У.В. /
подпись *(ФИО руководителя)*

Задание принял к исполнению
« 26 » января 2024 г.

_____ / Харченко Е.В. /
подпись *(ФИО студента)*

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

ОТЗЫВ

о работе обучающегося Харченко Евгения Викторовича
(фио обучающегося)
в период подготовки выпускной квалификационной работы по теме
«Разработка модуля генерации ландшафта для Unity3d»
(название темы вкр)
направление подготовки 09.03.01 Информатика и вычислительная техника
направленность (профиль): «Программное обеспечение средств вычислительной техники и автоматизированных систем»

текст отзыва – Общие выводы и заключение по результатам работы обучающегося в период подготовки ВКР, о степени выполнения поставленной цели и решения задач ВКР, о рекомендации ВКР к защите

Характеристика сформированности компетенций обучающегося	(высокий/средний/низкий)
Уровень освоения универсальных компетенций	
Уровень освоения общепрофессиональных компетенций	
Уровень освоения профессиональных компетенций	

Работа имеет практическую ценность		Тема предложена предприятием	
Работа имеет научно-исследовательский характер		Тема предложена студентом	
Рекомендую работу к внедрению		Тема предложена кафедрой	
Работа внедрена		Тема является фундаментальной	
Рекомендую работу к опубликованию		Рекомендую студента в магистратуру	
Результат опубликован		Рекомендую студента в аспирантуру	

Руководитель выпускной квалификационной работы

(должность, уч. степень, подпись, фамилия имя отчество полностью)

_____ _____ 2024г.
(подпись руководителя ВКР)

С отзывом ознакомлен _____/ _____ _____ 2024г.
(подпись) (ФИО обучающегося)

Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

направление подготовки 09.03.01 Информатика и вычислительная техника
направленность (профиль): «Программное обеспечение средств вычислительной техники и автоматизированных систем»

АННОТАЦИЯ

Выпускной квалификационной работы Харченко Е.В.
(Фамилия,И.О.)

по теме « Разработка модуля генерации ландшафта для Unity3d »

Объём работы - 44 страницы, на которых размещены 29 рисунков и 0 таблиц. При написании работы использовалось 5 источников.

Ключевые слова: среда разработки Unity3d, язык программирования C#, разработка игр, процедурная генерация, генерация ландшафта, инструменты разработки.

Работа выполнена ПМиК
(название предприятия, подразделения)
Руководитель ст. преподаватель Павлова Ульяна Владимировна
(должность, уч.степень, звание, Фамилия Имя Отчество)

Цель работы: разработка модуля генерации ландшафта для Unity3d, ускоряющего создание уникальных ландшафтов.

Решаемые задачи: 1) Изучение алгоритмов процедурной генерации; 2) Разработка приложения используя Unity API.

Основные результаты: разработан модуль генерации ландшафта для Unity3d, позволяющий быстро и просто создавать сложные ландшафты с помощью карт высот и процедурной генерации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ 5
1.1 Среда разработки Unity 5
1.2 Язык программирования C# 6
1.3 Редактор кода Visual Studio Code 8
ГЛАВА 2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 10
2.1 Теоретические основы генерации ландшафта 10
2.2 Объект Terrain 14
2.3 Класс EditorWindow 16
ГЛАВА 3. ПРАКТИЧЕСКАЯ ЧАСТЬ 18
3.1 Описание программной реализации 18
3.2 Пример использования программы 38
ЗАКЛЮЧЕНИЕ 43
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 44

ВВЕДЕНИЕ

Современная игровая индустрия представляет собой динамичную и быстроразвивающуюся сферу развлечений, которая в настоящее время находится в фазе активного роста. Этот рост обусловлен не только технологическими инновациями, но и изменяющимися предпочтениями и ожиданиями игроков, а также расширением доступности игровых платформ. Появление мобильных устройств, планшетов, виртуальной и дополненной реальности, а также консолей нового поколения создает новые возможности для разработчиков игр и расширяет аудиторию, привлекая к игровому опыту все больше людей. Разнообразие игровых жанров также позволяет удовлетворить различные интересы и предпочтения игроков, что способствует дальнейшему развитию и росту индустрии.

Основываясь на стремлении к получению более качественного и насыщенного игрового опыта, требовательность игроков в современной игровой индустрии соответственно продолжает расти. Этот рост требовательности обусловлен такими факторами, как красивая графика, захватывающая атмосфера, увлекательный игровой процесс и уникальные механики. В такой ситуации, стремясь предложить игрокам что-то новое и уникальное, игровые разработчики вынуждены постоянно совершенствовать свои продукты. Одним из ключевых факторов успеха современных игр является их способность привлекать и удерживать внимание игроков. В условиях огромного выбора игрового контента конкуренция за внимание становится все более острой, что вынуждает разработчиков и издателей игр стремиться к созданию уникальных и высококачественных продуктов.

В играх имеется множество различных объектов и аспектов для применения новаторских идей, одним из них является ландшафт. Ландшафт в игровом процессе играет значительную роль, определяя атмосферу, окружение и визуальное впечатление от игры. Он является неотъемлемой частью игрового мира, создавая пространство, в котором развивается сюжет и происходят действия игрока. В современных играх ландшафт становится все более важным элементом, который не только служит декорацией, но и активно взаимодействует с игровым процессом, влияя на игровой процесс и взаимодействие игрока с окружающим миром.

Атмосферные и разнообразные игровые миры, в которых действует игрок, часто являются ключевым фактором в создании эмоциональной привязки и погружения. Ландшафт может быть использован для передачи определенного настроения или эмоций. Правильно созданный ландшафт способен значительно усилить впечатление от игры и сделать её более запоминающейся и привлекательной для игроков. Разнообразные рельефы, такие как горы, реки, леса или пустыни, могут требовать от игрока применения различных стратегий и навыков для их преодоления, что обогащает игровой опыт и делает его более увлекательным.

Кроме того, ландшафт играет важную роль в создании нелинейности в игровом процессе. Разнообразные и интересные местности могут стимулировать игрока исследовать игровой мир, обнаруживать новые места и секреты, что расширяет возможности выбора и взаимодействия игрока с игровым миром.

Модуль генерации ландшафта в Unity3D представляет собой инструмент, позволяющий разработчикам создавать уникальные игровые миры, основываясь на предоставленных изображениях высот, математических процедурах и задаваемых параметрах. В контексте современной игровой индустрии, где важным является создание оригинального и качественного контента, использование такого модуля становится необходимостью, особенно для инди-разработчиков и малых студий, стремящихся создать уникальные и привлекательные игровые миры.

Таким образом, выбор темы о модуле генерации ландшафта в Unity3D обусловлен не только желанием исследовать новые технологии и инструменты в игровой разработке, но и стремлением к созданию более качественного и привлекательного игрового контента. Модуль генерации ландшафта не только сокращает время, затрачиваемое на создание игровых миров, но и позволяет разработчикам создавать разнообразные и уникальные ландшафты, способные увлечь игроков и углубить их погружение в игровой мир.

ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ

Основной платформой для разработки стал игровой движок Unity3d, широко используемый для создания как 2D, так и 3D игр. В качестве языка программирования был выбран C#, благодаря его тесной интеграции с Unity и богатым возможностям для разработки. Для написания кода использовался редактор Visual Studio Code, который предоставляет удобную и настраиваемую среду разработки с поддержкой множества расширений и инструментов для повышения производительности.

1.1 Среда разработки Unity

Unity — это кроссплатформенная среда разработки игр и интерактивных приложений, созданная компанией Unity Technologies. Первоначально выпущенная в 2005 году, Unity стала одной из самых популярных платформ для создания 2D и 3D контента, благодаря своей мощной функциональности, гибкости и доступности для разработчиков всех уровней. [1]

Unity3d предоставляет обширные возможности для создания интерактивного 3D-контента и рендеринга. Платформа включает в себя физический движок, который поддерживает высокоточные физические симуляции, включая физику твердых тел, мягких тел, жидкостей и столкновений. Это позволяет создавать реалистичные и динамичные окружения. Кроме того, Unity поддерживает продвинутые технологии освещения, такие как глобальное освещение, тени и отражения, что способствует созданию визуально впечатляющих сцен. Широкие возможности работы с шейдерами и материалами позволяют разработчикам создавать детализированные и сложные визуальные эффекты. Одним из ключевых преимуществ Unity является его кроссплатформенность. Платформа поддерживает разработку под множество операционных систем и устройств, включая Windows, macOS, Linux, iOS, Android, WebGL, а также основные игровые консоли, такие как PlayStation, Xbox и Nintendo Switch. Это позволяет разработчикам создавать один проект и компилировать его для различных платформ без необходимости внесения значительных изменений в код. Unity также активно поддерживает разработку VR и AR приложений. Платформа интегрируется с популярными SDK и платформами для виртуальной и дополненной реальности, такими как Oculus, HTC Vive, ARKit и ARCore, что позволяет разрабатывать передовые VR/AR решения. Дополнительно, Unity поддерживает создание и воспроизведение 360-градусного видео для VR приложений. Сама разработка осуществляется на языке программирования C#, что обеспечивает мощный инструментарий для реализации логики игры, управления поведением объектов и взаимодействием с пользователем. Полная интеграция с Visual Studio позволяет использовать этот IDE для

написания, отладки и тестирования кода, что значительно ускоряет процесс разработки.

Платформа имеет свой магазин Unity Asset Store, который предоставляет доступ к огромной библиотеке готовых активов, включая модели, текстуры, анимации, звуки и скрипты, что позволяет значительно сократить время разработки.

Для совместной работы над проектами Unity предлагает Unity Collaborate, встроенную систему для синхронизации изменений и управления версионностью проекта. Платформа также поддерживает интеграцию с популярными системами контроля версий, такими как Git и Perforce, что облегчает управление проектами в команде.

Для анализа и оптимизации производительности Unity предоставляет мощный профайлер, который помогает выявлять узкие места и оптимизировать использование ресурсов. Широкий набор инструментов для отладки кода и логики игры также способствует созданию высококачественного и стабильного продукта.

Unity Editor, основной интерфейс платформы, отличается интуитивно понятным дизайном, позволяющим легко создавать и настраивать сцены. Дополнительную гибкость предоставляет возможность создания пользовательских инструментов и панелей в редакторе через Unity Editor Scripting. Создание пользовательских интерфейсов в Unity Editor осуществляется с помощью библиотеки EditorWindow. Этот инструмент позволяет разработчикам создавать собственные окна редактора, которые могут включать в себя пользовательские панели, кнопки, текстовые поля и другие элементы интерфейса. Это особенно полезно для создания специализированных инструментов и редакторов, которые облегчают процесс разработки.

Используя EditorWindow, разработчики могут расширять функциональность Unity Editor, создавая удобные и интуитивно понятные интерфейсы для управления сложными процессами. Например, можно создать окно для управления настройками генерации ландшафта, в котором пользователи смогут легко настраивать параметры генерации.

Таким образом, Unity3d предлагает разработчикам удобные и гибкие возможности для создания интерактивного 3D-контента, делая процесс разработки более эффективным и результативным.

1.2 Язык программирования C#

C# (C-sharp) — это объектно-ориентированный язык программирования, разработанный корпорацией Microsoft как часть своей платформы .NET. Первоначально представленный в 2000 году, C# был создан под руководством Андерса Хейлсберга, который также участвовал в разработке языков Turbo Pascal и Delphi. C# был задуман как простой, современный, безопасный и производительный язык для разработки

разнообразных приложений, включая веб-приложения, настольные приложения и серверные решения.

C# широко используется в различных сферах разработки программного обеспечения. Он особенно популярен в корпоративной среде для создания бизнес-приложений, а также для разработки игр и мобильных приложений благодаря своей универсальности и мощным возможностям. Будучи основным языком для платформы .NET, C# предоставляет разработчикам доступ к широкому спектру библиотек и инструментов, что ускоряет процесс создания надежных и производительных приложений.

В контексте Unity, C# играет ключевую роль в разработке игр и интерактивных приложений. Unity использует C# для написания скриптов, которые определяют поведение объектов в игре, обработку ввода пользователя, управление логикой игры и взаимодействие с различными системами Unity. С помощью C# разработчики могут создавать сложные алгоритмы и управлять игровыми процессами, что делает его неотъемлемой частью экосистемы Unity.

Unity предоставляет интегрированную среду разработки (IDE), такую как Visual Studio, для написания и отладки C# кода. Это позволяет разработчикам использовать все преимущества мощных инструментов отладки и автозаполнения, что значительно упрощает процесс разработки и помогает избежать ошибок.

Возможности и преимущества C#:

- Объектно-ориентированное программирование (ООП). C# поддерживает все основные принципы ООП, такие как инкапсуляция, наследование и полиморфизм. Это позволяет создавать модульный и легко поддерживаемый код, что особенно важно при разработке больших и сложных проектов.
- Типобезопасность. C# является типобезопасным языком, что означает, что большинство ошибок типа обнаруживаются на этапе компиляции, а не во время выполнения. Это помогает предотвратить многие распространенные ошибки и делает программы более надежными и стабильными.
- Современные языковые конструкции. C# включает в себя множество современных языковых конструкций, таких как лямбда-выражения, анонимные методы, асинхронное программирование (async/await), линейный синтаксис LINQ и другие. Эти возможности позволяют писать более чистый и лаконичный код, а также повышают производительность разработки.
- Широкая стандартная библиотека. .NET предоставляет обширную библиотеку классов (BCL), которая включает в себя множество готовых компонентов для выполнения различных задач, таких как работа с файлами, сетью, базами данных и многим другим. Это ускоряет процесс разработки, позволяя использовать готовые решения для распространенных задач.

- Интеграция с .NET платформой. Использование C# в контексте .NET платформы обеспечивает доступ к мощным инструментам и технологиям, таким как ASP.NET для веб-разработки, ADO.NET для работы с базами данных, Windows Forms и WPF для создания настольных приложений, и многие другие. Это делает C# универсальным инструментом для создания широкого спектра приложений.
- Поддержка асинхронного программирования. C# включает встроенную поддержку асинхронного программирования через ключевые слова async и await, что позволяет легко писать асинхронные методы и обрабатывать асинхронные операции. Это особенно важно для разработки игр и приложений, где необходимо выполнять длительные операции без блокировки основного потока.
- Активное сообщество и поддержка. C# имеет большое и активное сообщество разработчиков, множество онлайн-ресурсов, форумов и учебных материалов. Это облегчает обучение языку и помогает быстро находить решения для возникающих проблем. Регулярные обновления языка и активное развитие платформы .NET также способствуют его актуальности и улучшению. [2]

1.3 Редактор кода Visual Studio Code

Visual Studio Code (VS Code) — это бесплатный, кроссплатформенный редактор исходного кода, разработанный компанией Microsoft. Впервые выпущенный в апреле 2015 года, VS Code быстро завоевал популярность среди разработчиков благодаря своей гибкости, высокой производительности и широким возможностям настройки. Редактор поддерживает множество языков программирования и предоставляет мощные инструменты для разработки, отладки и управления проектами.

VS Code отличается небольшим размером и высокой скоростью работы, что делает его удобным для использования на компьютерах с различной производительностью. Он быстро запускается и работает плавно, даже с большими проектами. Поддержка большого количества языков программирования и встроенная подсветка синтаксиса облегчают написание и чтение кода. Также доступна функция автодополнения, которая ускоряет процесс кодирования и снижает вероятность ошибок. Visual Studio Code является отличным выбором для разработки в Unity3d благодаря своей гибкости и широким возможностям настройки. Хотя Unity предоставляет встроенную поддержку Visual Studio, VS Code также можно легко интегрировать с Unity для разработки на C#.

Для работы с C# в VS Code используется расширение C# for Visual Studio Code, разработанное компанией Microsoft. Это расширение предоставляет поддержку IntelliSense, отладки, рефакторинга и других функций, необходимых для эффективной работы с C# в Unity. VS Code

может быть настроен в качестве внешнего редактора скриптов в Unity. Это позволяет открывать и редактировать C# скрипты напрямую из Unity, используя возможности VS Code для автодополнения и отладки. Unity автоматически синхронизирует изменения в скриптах, что делает процесс разработки более гладким и эффективным. Visual Studio Code был выбран в качестве среды разработки для модуля генерации ландшафта в Unity3d благодаря своим многочисленным преимуществам. Легкость и производительность, мощные инструменты редактирования и отладки, возможность интеграции с Unity и поддержка систем контроля версий делают VS Code идеальным инструментом для разработки игр и интерактивных приложений.

ГЛАВА 2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Теоретические основы генерации ландшафта

Генерация ландшафта — это процесс создания цифровых моделей природных или искусственных ландшафтов с использованием алгоритмов и программных средств. Целью генерации ландшафта является создание реалистичных и детализированных окружающих сред, которые могут быть использованы в видеоиграх, симуляторах, анимационных фильмах и других интерактивных приложениях. Этот процесс включает в себя моделирование различных географических и природных элементов, таких как горы, холмы, реки, леса и пустыни. Генерация ландшафта может быть осуществлена с использованием различных методов, включая фрактальную геометрию и шумовые функции, что позволяет создавать сложные и правдоподобные виртуальные окружения.

В данном проекте для генерации ландшафта я выбрал использование шума Перлина. Выбор обосновывается несколькими причинами: во-первых, шум Перлина создает плавные и естественные изменения высот, что позволяет моделировать реалистичные ландшафты с органичными формами, такими как холмы, горы и долины, избегая при этом резких и неестественных переходов, характерных для традиционных случайных шумов; во-вторых, его алгоритм легко настраивается и масштабируется, что дает разработчикам гибкость в создании различных типов ландшафтов.

Изначально шум Перлина был разработан Кеном Перлином, американским ученым и профессором информатики, в 1983 году. Перлин создал этот алгоритм в ответ на необходимость генерировать естественные текстуры и визуальные эффекты для компьютерной графики, особенно для создания плавных и органичных форм в анимации и играх. Шум Перлина можно рассматривать как метод генерации псевдослучайных значений, которые распределены таким образом, чтобы образовывать плавные и непрерывные переходы между точками. Основной идеей алгоритма является использование интерполяции между случайными градиентами, чтобы создать эффект плавного изменения значений.

Основные принципы шума Перлина:

1. Генерация решетки градиентов
    - Пространство, в котором генерируется шум, разбивается на решетку, где каждая точка пересечения (узел) имеет случайный градиентный вектор. Эти векторы указывают направление и величину изменений значений шума.

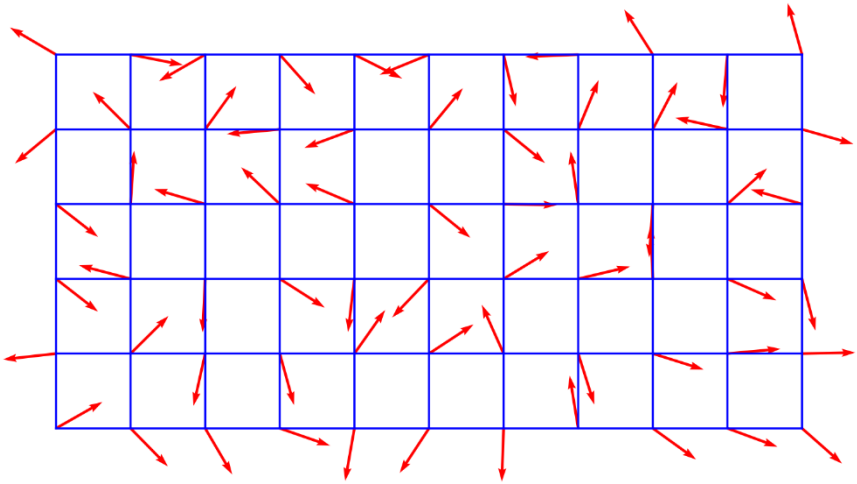

Рисунок 2.1 – Решетка градиентных векторов.

2. Вычисление влияния градиентов
    - Для каждой точки внутри ячейки решетки вычисляются векторные расстояния до всех узлов ячейки. Затем производится скалярное произведение этих векторов расстояний с соответствующими градиентными векторами. Это позволяет определить вклад каждого узла в итоговое значение шума для данной точки.

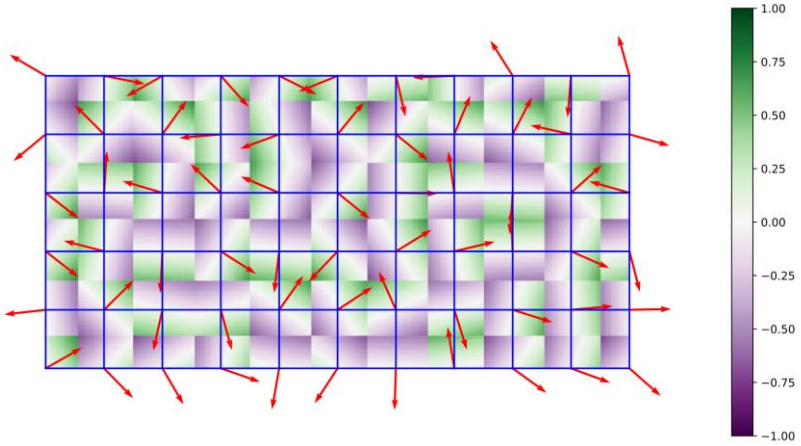

Рисунок 2.2 – Скалярное произведение векторов.

3. Интерполяция значений
    - Итоговое значение шума для точки внутри ячейки получается путем интерполяции (обычно с использованием функции сглаживания, такой как кубическая или квадратичная интерполяция) между вкладом всех узлов ячейки. Эта интерполяция создает плавные переходы между значениями, избегая резких изменений.

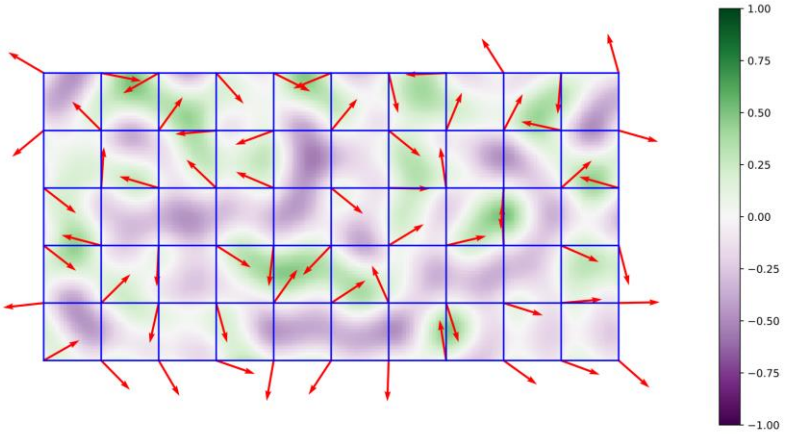

Рисунок 2.3 – Конечный результат

Встроенные инструменты и библиотеки Unity3D поддерживают использование шума Перлина. Одной из них является Mathf.PerlinNoise(), она предоставляет простой способ генерации двумерного шума Перлина.

```
float Mathf.PerlinNoise(float x, float y);
```

Параметры:

- x (float): Координата по оси X.
- y (float): Координата по оси Y.

Возвращаемое значение:

- float: Значение шума Перлина в диапазоне от 0.0 до 1.0 для заданных координат.

Листинг 2.1 – Классический пример использования функции

```
float CalculateHeight(int x, int y)
{
    float xCoord = (float)x / width * scale;
    float yCoord = (float)y / height * scale;

    return Mathf.PerlinNoise(xCoord, yCoord);
}
```

Данная функция преобразует координаты x и y в координаты для функции с учетом масштаба и возвращает значение шума Перлина для заданных координат. [3]

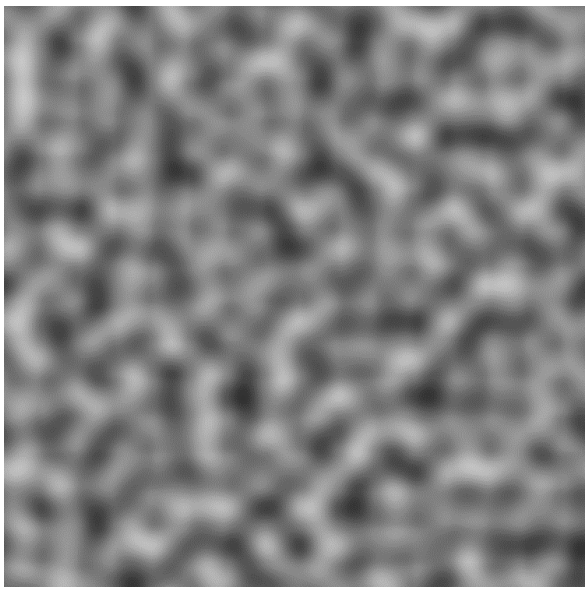

Рисунок 2.4 – Пример работы функции.

Также для генерации ландшафта может использоваться карта высот. Генерация ландшафта по карте высот представляет собой процесс создания трехмерной поверхности на основе двумерного изображения, где значение каждого пикселя указывает высоту ландшафта в соответствующей точке. Этот метод широко используется в разработке игр и других графических приложениях благодаря своей простоте и эффективности.

Карта высот — это изображение в градациях серого, где яркость каждого пикселя указывает на высоту в данной точке:

- Черный цвет (значение 0) соответствует самой низкой точке.
- Белый цвет (значение 255) соответствует самой высокой точке.
- Оттенки серого представляют промежуточные высоты.

Использование карт высот позволяет разработчикам легко контролировать и модифицировать рельеф, а также импортировать данные из реальных топографических источников.

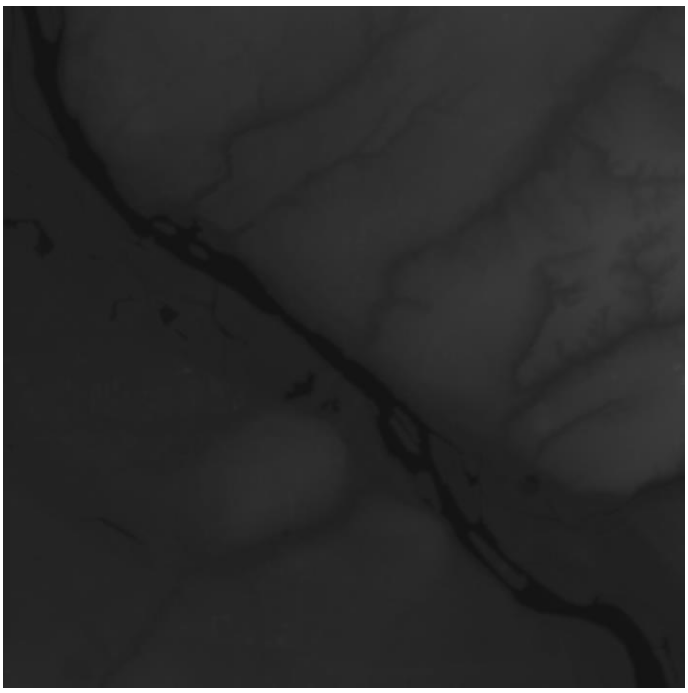

Рисунок 2.5 – Карта высот г. Новосибирск.

В Unity у объектов класса Color предусмотрен параметр grayscale, который возвращает оттенок серого в диапазоне от 0f до 1f, при котором 0f – это черный цвет, 1f – белый цвет.

2.2 Объект Terrain

Объект Terrain в Unity3D представляет собой специализированный компонент, используемый для создания и визуализации больших, детализированных ландшафтов в 3D-пространстве. Этот компонент предоставляет разработчикам инструменты для моделирования различных природных и искусственных форм рельефа, таких как горы, долины, равнины и водоемы.

Terrain состоит из нескольких классов, которые взаимодействуют друг с другом для создания и управления различными аспектами ландшафта. Основными классами являются Terrain, TerrainData, TerrainLayer и TreeInstance. Каждый из этих классов выполняет определенную роль в системе генерации и управления ландшафтами.

Класс Terrain является основным классом, который представляет объект ландшафта в сцене. Он отвечает за визуализацию и управление ландшафтом. Также он включает в себя не менее важный класс TerrainData. Класс TerrainData содержит все данные, необходимые для генерации ландшафта, включая высоты, текстуры, деревья и другие элементы. Именно через него применяются практически все физические изменения объекта.

Он включает в себя такие важные методы, как:

1. heightmapResolution - разрешение высотной карты, определяет количество треугольников на Terrain.
2. size - размеры ландшафта в мировых координатах.

3. GetHeights(), SetHeights() - получение и установка высот на ландшафте. Возвращает и получает массив размером [heightmapResolution]*[heightmapResolution], состоящий из значений от нуля до единицы. Высота в точке определяется умножением значения массива на значение size.y.
4. alphamapResolution: Разрешение альфа-карт, используемых для текстурирования.
5. terrainLayers: Массив объектов TerrainLayer, определяющих слои текстур.
6. treeInstances: Массив объектов TreeInstance, представляющих деревья на ландшафте.
7. GetAlphamaps(), SetAlphamaps(): Получение и установка данных альфа-карт.

TerrainLayer представляет собой компонент, используемый для управления текстурами, которые применяются к поверхности ландшафта (Terrain). Этот компонент позволяет настраивать внешний вид ландшафта, комбинируя различные текстуры и создавая реалистичные покрытия. Он включает в себя такие характеристики, как:

1. Diffuse Texture - основная текстура, отображающая цвет поверхности ландшафта.
2. Tile Size - размер текстуры в мировых координатах, определяющий, как часто текстура будет повторяться по поверхности.
3. Tile Offset - смещение текстуры, позволяющее изменять начальную точку повторения текстуры на поверхности ландшафта.

Методы и параметры:

1. metallic - уровень металличности текстуры, который влияет на отражение света.
2. smoothness - уровень гладкости текстуры, который влияет на отражение света.
3. diffuseRemapMin и diffuseRemapMax - позволяют изменять диапазон значений диффузной текстуры.
4. normalScale - масштаб нормалей, который определяет интенсивность эффекта нормальной карты. [4]

2.3 Класс EditorWindow

В среде Unity3d также существует средство разработки пользовательских интерфейсов. Предоставляет эти возможности класс EditorWindow. Он позволяет расширять функциональность Unity Editor, создавать собственные интерфейсы для специфических задач. Окна, создаваемые через EditorWindow, могут быть закреплены в Инспекторе или открываться как всплывающие окна. Эти окна могут содержать различные элементы интерфейса, такие как кнопки, поля ввода, слайдеры, выпадающие списки и другие GUI-элементы. [5]

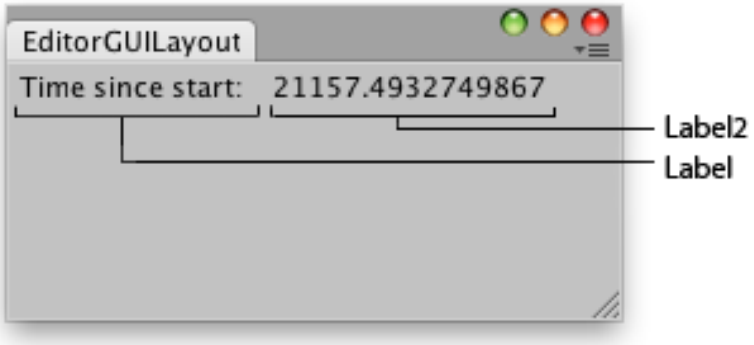

Рисунок 2.6 – LabelField.

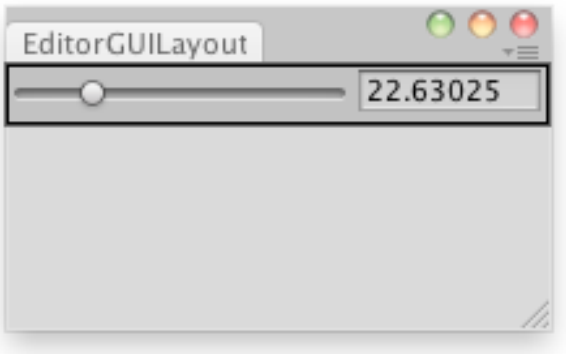

Рисунок 2.7 – Slider.

Основные методы для отображения интерфейса – это ShowWindow() и OnGUI:

ShowWindow() - используется для создания и отображения пользовательских окон, наследующих от класса EditorWindow. Этот метод обычно вызывается из статического метода, помеченного атрибутом MenuItem, что позволяет добавить новый пункт меню в Unity Editor. При вызове ShowWindow() создается экземпляр пользовательского окна и открывается в редакторе Unity, предоставляя доступ к созданным пользователем дополнительным инструментам и интерфейсам. Метод также может настраивать заголовок окна и его начальные параметры.

OnGUI() - является ключевым методом для рендеринга и обработки пользовательского интерфейса (GUI) в классах, наследующих от EditorWindow. Этот метод вызывается каждый раз, когда происходит обновление интерфейса, и отвечает за отображение всех элементов GUI, а также за обработку взаимодействий с пользователем.

Внутри метода OnGUI() используются различные функции из UnityEngine.GUI и UnityEditor.EditorGUILayout для создания и размещения элементов интерфейса, таких как кнопки, текстовые поля, метки, слайдеры и переключатели.

Листинг 2.2 – Пример создания пользовательского интерфейса с помощью класса EditorGUILayout.

```
void OnGUI()
{
    GUILayout.Label("Настройки", EditorStyles.boldLabel);
    string myString =
EditorGUILayout.TextField("Текстовое поле", myString);
    bool myBool = EditorGUILayout.Toggle("Переключатель",
myBool);
    float myFloat = EditorGUILayout.Slider("Слайдер",
myFloat, 0, 10);
}
```

Метод OnGUI() также отвечает за обработку событий ввода, таких как нажатия клавиш, клики мышью и изменение значений элементов интерфейса.

Листинг 2.3 – Обработка нажатия кнопки.

```
void OnGUI()
{
    if (GUILayout.Button("Нажми меня"))
    {
        Debug.Log("Кнопка нажата");
    }
}
```

OnGUI() позволяет настраивать внешний вид элементов интерфейса с помощью стилей и параметров оформления. Это может включать изменение шрифтов, цветов, размеров и других визуальных характеристик.

Листинг 2.4 – Создание и применение пользовательского стиля.

```
void OnGUI()
{
    GUIStyle style = new GUIStyle(GUI.skin.label);
    style.fontSize = 20;
    style.normal.textColor = Color.red;

    GUILayout.Label("Стилизация текста", style);
}
```

ГЛАВА 3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Описание программной реализации

Интерфейс модуля имеет компактный вид и деление на секции. Для самого модуля была создана отдельная опция в меню. Опция имеет название Tools.

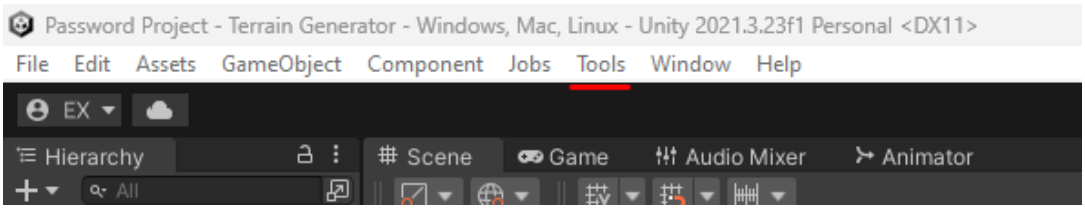

Рисунок 3.1 – Опция в меню: Tools.

Чтобы открыть модуль пользователь нажимает на опцию Tools и в выпадающем списке выбирает Terrain Generator.

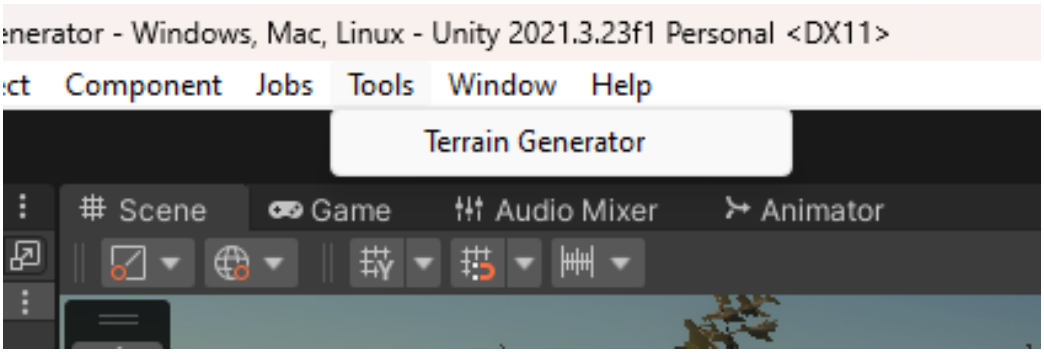

Рисунок 3.2 – Выбор Terrain Generator.

Листинг 3.1 – Программная реализация опции в меню и открытие окна

```
[MenuItem("Tools/Terrain Generator")]
 public static void ShowWindow()
 {
      GetWindow(typeof(TerrainGenerator));
 }
```

Модуль имеет 4 основных секции и панель с основной информацией о редактируемом объекте Terrain. По умолчанию все поля недоступны, пока пользователь не выберет редактируемый объект Terrain.

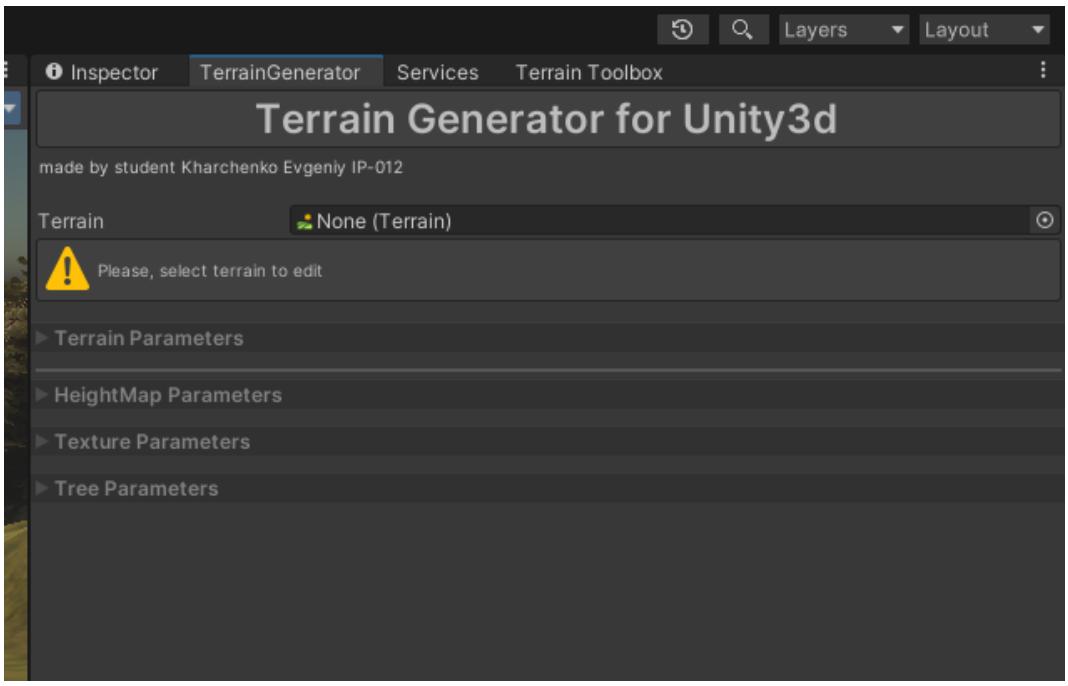

Рисунок 3.3 – Начальный интерфейс программы.

После выбора редактируемого объекта Terrain, пользователю становятся доступны поля Terrain Parameters, HeightMap Parameters, Texture Parameters и Tree Parameters. Также вместо предупреждения “Please, select terrain to edit” появляется панель с основной информацией об объекте.

Рисунок 3.4 – Интерфейс программы после выбора объекта Terrain.

Панель содержит базовую информацию о текущем проекте и объекте Terrain. Включает название проекта, имя разработчика, масштаб и разрешение высотной карты (heightmap) и альфа-карты (alphamap). Панель была реализована комбинацией стиля и сетки вертикального и горизонтального деления.

Листинг 3.2 – Реализация панели информации.

```
if(terrainObject)
        {
            EditorGUILayout.BeginVertical(EditorStyles.h
elpBox);
            EditorGUILayout.LabelField("Scale: " +
terrainObject.terrainData.size);
            EditorGUILayout.BeginHorizontal();
            EditorGUILayout.LabelField("Heightmap
Resolution: " +
terrainObject.terrainData.heightmapResolution);
            EditorGUILayout.LabelField("Alphamap
resolution: " +
terrainObject.terrainData.alphamapResolution);
            EditorGUILayout.EndHorizontal();
            EditorGUILayout.EndVertical();
            GUI.enabled = true;
        }
        else
        {
            EditorGUILayout.HelpBox("Please, select
terrain to edit", MessageType.Warning);
            GUI.enabled = false;
        }
```

Из листинга 3.2 видно, что элементы объявляются в ветвлении if(terrainObject). Переменная terrainObject является ссылкой на редактируемый объект Terrain. Ветвление здесь используется для того, чтобы в зависимости от того, выбрал ли пользователь, редактируемый Terrain, был отображались тот или иной интерфейс. Если пользователь выбрал редактируемый объект, ему отобразится панель с параметрами, иначе предупреждение, чтобы пользователь выбрал Terrain.

Листинг 3.3 – объявление переменной terrainObject и ее основное применение

```
public Terrain terrainObject;
…
EditorGUI.BeginChangeCheck();
        terrainObject =
EditorGUILayout.ObjectField("Terrain", terrainObject,
typeof(Terrain), true) as Terrain;
        //Параметры по умолчанию
        if(EditorGUI.EndChangeCheck())
        {
            if(terrainObject)
            {
                _terrainData = terrainObject.terrainData;
                _scaleTerrain.x = (int)_terrainData.size.x;
                _scaleTerrain.y = (int)_terrainData.size.y;
                _heightmapResolution =
(int)Math.Log(_terrainData.heightmapResolution, 2);
```

```
            }
        }
```

Разберем каждую секцию по отдельности:

1) Terrain Parameters.

Секция Terrain Parameters предназначена для настройки параметров объекта Terrain. Включает параметры масштаба и разрешения высотной карты, а также кнопки для применения изменений, масштаб и разрешение высотной карты применяются отдельно друг от друга. При этом, изменение разрешения карты высот приведет к сбросу карты высот Terrain'а, об этом для пользователя есть предупреждение.

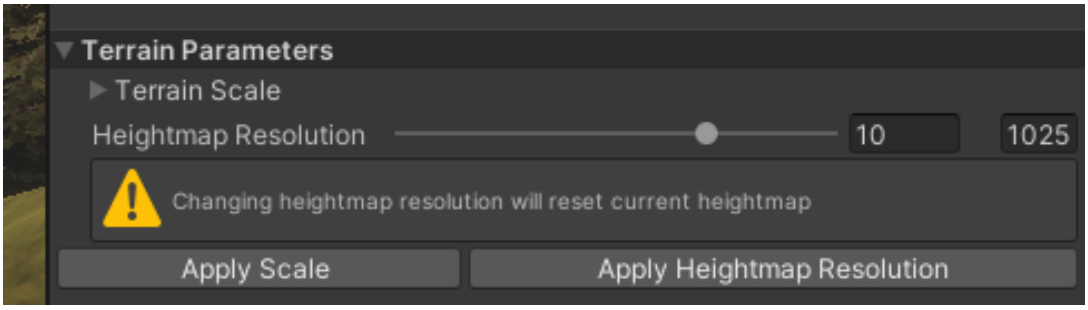

Рисунок 3.5 – Секция настройки параметров Terrain'а.

Вся секция отображается через элемент EditorGUILayout.Foldout. Это позволяет складывать всю секцию, если она не требует изменений.

Листинг 3.4 – Реализация секции Terrain Parameters.

```
EditorGUILayout.BeginVertical("PopupCurveSwatchBackground");
        terrainParameters =
EditorGUILayout.Foldout(terrainParameters, "Terrain
Parameters", foldoutStyle);
        EditorGUILayout.EndVertical();

        if(terrainParameters && terrainObject)
        {
            EditorGUI.indentLevel++;
            scaleTerrainFold =
EditorGUILayout.Foldout(scaleTerrainFold, "Terrain Scale");
            if(scaleTerrainFold)
            {
                _scaleTerrain =
EditorGUILayout.Vector2IntField("", _scaleTerrain);
                _scaleTerrain.x = (_scaleTerrain.x < 0) ? 1 :
_scaleTerrain.x;
                _scaleTerrain.y = (_scaleTerrain.y < 0) ? 1 :
_scaleTerrain.y;
            }

            EditorGUILayout.BeginHorizontal();
```

```
            _heightmapResolution =
EditorGUILayout.IntSlider("Heightmap Resolution",
_heightmapResolution, 5, 12);
            EditorGUILayout.LabelField("" +
((int)Mathf.Pow(2f, _heightmapResolution) + 1),
EditorStyles.numberField, GUILayout.ExpandWidth(true),
GUILayout.MaxWidth(50));
            EditorGUILayout.EndHorizontal();

            EditorGUILayout.HelpBox("Changing heightmap
resolution will reset current heightmap",
MessageType.Warning);
            EditorGUI.indentLevel--;

            EditorGUILayout.BeginHorizontal();
            if(GUILayout.Button("Apply Scale"))
            {
                ApplyTerrainScale();
                Debug.Log("Scale Applied!");
            }
            if(GUILayout.Button("Apply Heightmap
Resolution"))
            {
                ApplyHeightMapResolution();
                Debug.Log("Heightmap Resolution Applied!");
            }
            EditorGUILayout.EndHorizontal();
        }
```

Масштаб Terrain’а также имеет свой EditorGUILayout.Foldout. У значений масштаба имеются ограничения: как только становятся меньше нуля, они принимают значение единицы. Тем самым, значения _scaleTerrain.x и _scaleTerrain.y могут иметь минимальное значение равным единице.

Heightmap Resolution реализован через слайдер со значениями от 5 до 12. Фактически они представляют собой степень двойки. Слайдер сделан именно таким образом, потому что разрешение высотной карты может быть равным исключительно значениям по формуле 2^x+1. Слайдер в этом случае отвечает за значения X в формуле.

При нажатии на кнопки значения, задаваемые через GUI элементы, присваиваются параметрам TerrainData нашего объекта Terrain. Единственное отличие между ними в том, что при применении разрешения высотной карты, значение, переданное через слайдер, вычисляется по формуле. Также перед присваиванием, текущий размер ландшафта записывается в переменную. Это делается для того, чтобы после изменения разрешения высотной карты восстановить прежний размер Terrain’а.

2) HeightMap Parameters.

Секция HeightMap Parameters имеет самую сложную структуру интерфейса из всех секций. Эта секция предназначена для настройки параметров высотной карты объекта Terrain. Она включает управление слоями и их параметрами, такими как шум Перлина и импорт высотных карт, а также функции для генерации и сглаживания карты высот. Пользователь может добавлять сколько угодно слоев и удалять последний добавленный слой. Сглаживание не имеет какой-либо привязки к реальным единицам измерения, поэтому пользователь методом проб сам определяет какое значение использовать. Также предоставлена возможность генерации всех слоев по порядку сразу и каждого слоя по отдельности, или же вовсе сброса всей карты высот до нулевых значений.

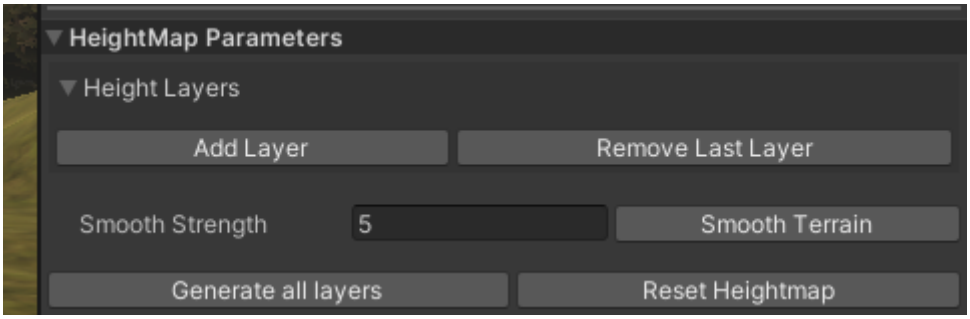

Рисунок 3.6 – Интерфейс секции HeightMap Parameters.

Каждый слой высотной карты имеет свои параметры, которые отображаются в отдельном EditorGUILayout.Foldout. В параметрах слоя можно включить или отключить слой, задать способ генерации (Perlin Noise или Height Map), а также настроить специфические параметры для каждого типа генерации.

Perlin Noise:

- Perlin Scale: Масштаб шума Перлина.
- Min Height: Минимальная высота.
- Max Height: Максимальная высота.
- Height Offset: Смещение по высоте.
- X Offset: Смещение по оси X.
- Z Offset: Смещение по оси Z.

Height Map:

- HeightMap: Текстура карты высот.
- Flip Horizontal Axis: Отражение по горизонтальной оси.
- Flip Vertical Axis: Отражение по вертикальной оси.
- Min Height: Минимальная высота.
- Max Height: Максимальная высота.

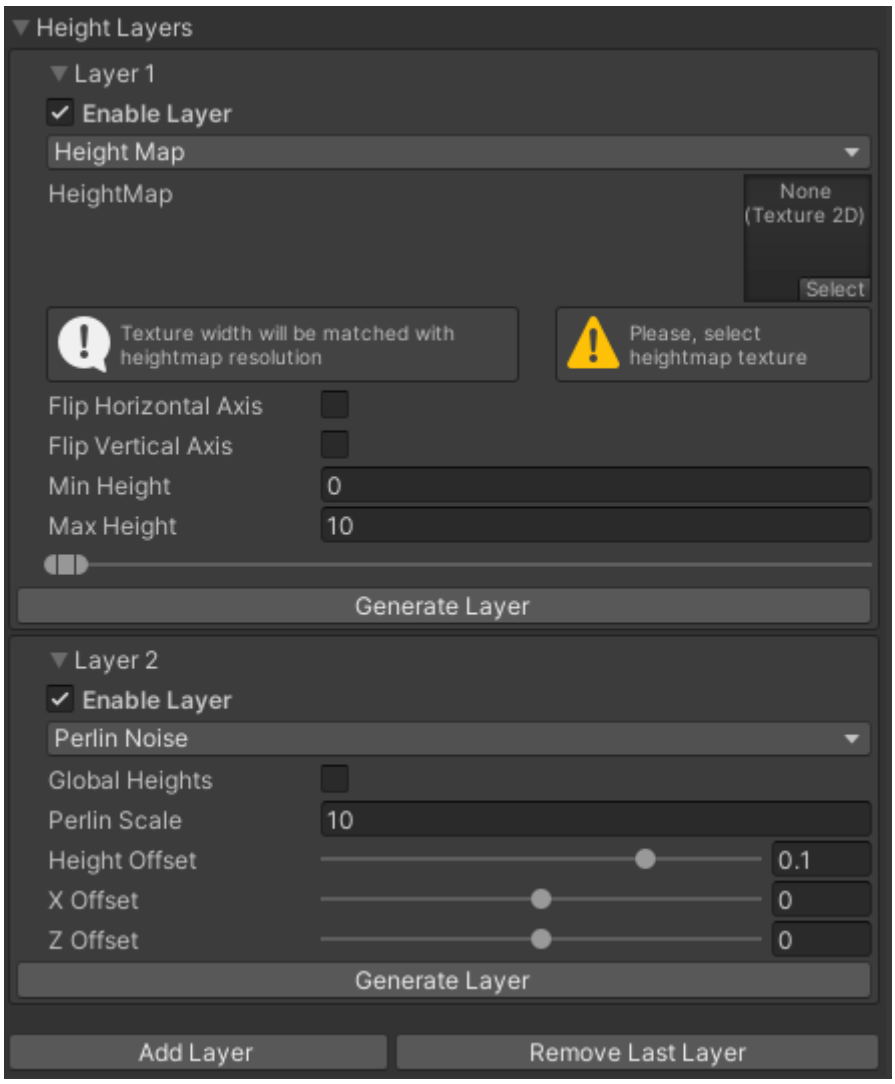

Рисунок 3.7 – Интерфейс двух способов генерации высот.

Слой Perlin Noise также имеет параметр Global Heights. Он отвечает за тип наложения шума Перлина. Если в поле стоит галочка, то значения массива высот будут приравниваться значениям шума Перлина с заданными параметрами без изменений. Если в поле стоит галочка, то значение шума Перлина будет изменять текущие значения массива высот.

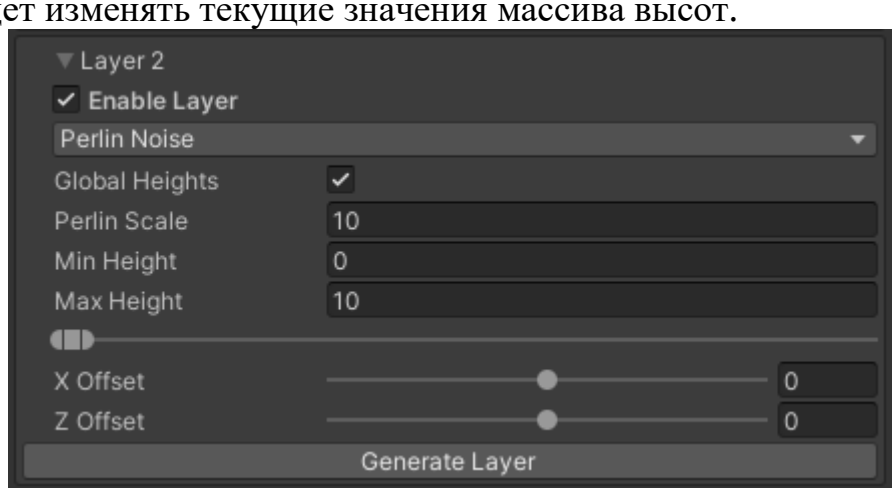

Рисунок 3.8 – Вариант слоя шума Перлина с параметром Global Heights.

Вывод слоев высот происходит через цикл. Каждый слой хранится в списке типа HeightMapLayer. HeightMapLayer – это созданный отдельный класс, который используется для создания и настройки слоев высотной карты объекта Terrain. Он включает в себя параметры для генерации шума Перлина и для использования текстур высотных карт. Класс также содержит методы для отображения GUI элементов и получения типа генерации.

Листинг 3.5 – Метод отображения GUI элементов для слоя высот.

```
public void DrawGUI()
    {
        typeIndex = EditorGUILayout.Popup(typeIndex,
options);
        switch(typeIndex)
        {
            case 0:
                isGlobal = EditorGUILayout.Toggle("Global
Heights", isGlobal);
                perlinScale =
EditorGUILayout.DelayedFloatField("Perlin Scale",
perlinScale);
                if(isGlobal)
                {
                    minHeight =
(int)EditorGUILayout.DelayedFloatField("Min Height",
minHeight);
                    maxHeight =
(int)EditorGUILayout.DelayedFloatField("Max Height",
maxHeight);
                    EditorGUILayout.MinMaxSlider(ref
minHeight, ref maxHeight, 0, maxY);
                }
                else
                {
                    heightOffset =
EditorGUILayout.Slider("Height Offset", heightOffset, -0.2f,
0.2f);
                }
                perlinOffsetX = EditorGUILayout.Slider("X
Offset", perlinOffsetX, -20, 20);
                perlinOffsetY = EditorGUILayout.Slider("Z
Offset", perlinOffsetY, -20, 20);
                break;
            case 1:
                heightMapTexture =
(Texture2D)EditorGUILayout.ObjectField("HeightMap",
heightMapTexture, typeof(Texture2D), false);
                EditorGUILayout.BeginHorizontal();
                EditorGUILayout.HelpBox("Texture width will
be matched with heightmap resolution", MessageType.Info);
                if(!heightMapTexture)
```

```
                {
                    EditorGUILayout.HelpBox("Please, select
heightmap texture", MessageType.Warning);
                }
                EditorGUILayout.EndHorizontal();
                flipZ = EditorGUILayout.Toggle("Flip
Horizontal Axis", flipZ);
                flipX = EditorGUILayout.Toggle("Flip Vertical
Axis", flipX);
                minHeight =
(int)EditorGUILayout.DelayedFloatField("Min Height",
minHeight);
                maxHeight =
(int)EditorGUILayout.DelayedFloatField("Max Height",
maxHeight);
                EditorGUILayout.MinMaxSlider(ref minHeight,
ref maxHeight, 0, maxY);
                break;
            default:
                break;
        }
    }
```

Метод DrawGUI() отображает элементы пользовательского интерфейса для настройки параметров слоя. В зависимости от выбранного типа генерации (typeIndex), отображаются различные элементы управления.

Листинг 3.6 – Вывод слоев высот через цикл.

```
for (int i = 0; i < heightMapLayers.Count; i++)
                {
                    heightMapLayers[i].perlinScale =
Mathf.Max(heightMapLayers[i].perlinScale, 0);
                    heightMapLayers[i].maxY =
(int)_terrainData.size.y;
                    heightMapLayers[i].minHeight =
Mathf.Clamp(heightMapLayers[i].minHeight, 0,
heightMapLayers[i].maxHeight);
                    heightMapLayers[i].maxHeight =
Mathf.Clamp(heightMapLayers[i].maxHeight, 0,
_scaleTerrain.y);

                    //Один слой
                    EditorGUILayout.BeginVertical(EditorStyle
s.helpBox);

                    heightMapLayers[i].heightMapLayerFold =
EditorGUILayout.Foldout(heightMapLayers[i].heightMapLayerFold
, $"Layer {i + 1}");
                    if(heightMapLayers[i].heightMapLayerFold)
                    {
```

```
                        heightMapLayers[i].enabled =
EditorGUILayout.BeginToggleGroup("Enable Layer",
heightMapLayers[i].enabled);
                        if(heightMapLayers[i].GetGenerationTy
pe() == "Height Map" && heightMapLayers[i].heightMapTexture
== null)
                        {
                            canGenerateAllLayers = false;
                        }
                        else
                        {
                            canGenerateAllLayers = true;
                        }
                        heightMapLayers[i].DrawGUI();

                        ...

                        EditorGUILayout.EndToggleGroup();
                    }
                    EditorGUILayout.EndVertical();
                }
```

Сгенерированные высоты применяются через встроенную функцию класса TerrainData.SetHeights. Основные манипуляции с массивом высот происходят в функции GenerateHeights(). В этой функции программа находит значение каждому элементу двумерного массива высот heights. В зависимости от выбранного способа генерации в слое высот, функция будет генерировать двумерный массив высот на основе шума Перлина или же карте высот. Также способ нахождения значения отличается в зависимости от переменной isGlobal - это та переменная, которая отвечает за вариант генерации шума Перлина.

Листинг 3.7 – Функция генерации высот.

```
float[,] GenerateHeights(int index)
    {
        float[,] heights = new
float[(int)_terrainData.heightmapResolution,
(int)_terrainData.heightmapResolution];
        float[,] currentHeights = _terrainData.GetHeights(0,
0, _terrainData.heightmapResolution,
_terrainData.heightmapResolution);

        switch(heightMapLayers[index].GetGenerationType())
        {
            case "Perlin Noise":
                for(int x = 0; x <
(int)_terrainData.heightmapResolution; x++)
                {
                    for(int z = 0; z <
(int)_terrainData.heightmapResolution; z++)
```

```
                    {
                        if(heightMapLayers[index].isGlobal)
                        {
                            heights[x, z] =
heightMapLayers[index].minHeight / _terrainData.size.y +
                                                CalculateHeight(x
, z, index) *
                                                (heightMapLayers[
index].maxHeight / _terrainData.size.y -
heightMapLayers[index].minHeight / _terrainData.size.y);
                        }
                        else
                        {
                            heights[x, z] = currentHeights[x,
z] + CalculateHeight(x, z, index) *
heightMapLayers[index].heightOffset;
                        }
                    }
                }
                break;
            case "Height Map":
                Texture2D resizedTexture =
ResizeTexture(heightMapLayers[index].heightMapTexture,
(int)_terrainData.heightmapResolution,
(int)_terrainData.heightmapResolution);
                Color[] pixels = resizedTexture.GetPixels();
                for(int x = 0; x <
(int)_terrainData.heightmapResolution; x++)
                {
                    for(int z = 0; z <
(int)_terrainData.heightmapResolution; z++)
                    {
                        int x1 =
(heightMapLayers[index].flipX) ?
(int)_terrainData.heightmapResolution - x - 1 : x;
                        int z1 =
(heightMapLayers[index].flipZ) ?
(int)_terrainData.heightmapResolution - z - 1 : z;

                        float heightValue = pixels[x1 + z1 *
(int)_terrainData.heightmapResolution].grayscale;
                        heights[x, z] =
heightMapLayers[index].minHeight / _terrainData.size.y +
                                            heightValue *
                                            (heightMapLayers[inde
x].maxHeight / _terrainData.size.y -
heightMapLayers[index].minHeight / _terrainData.size.y);
                    }
                }
                break;
        }

        return heights;
```

```
}
```

Важно отметить, что при генерации по карте высот, у используемого изображения меняется размер в соответствии с разрешением высотной карты. Также, после генерации зачастую встречаются артефакты рельефа – на ландшафте видны пиксели изображения.

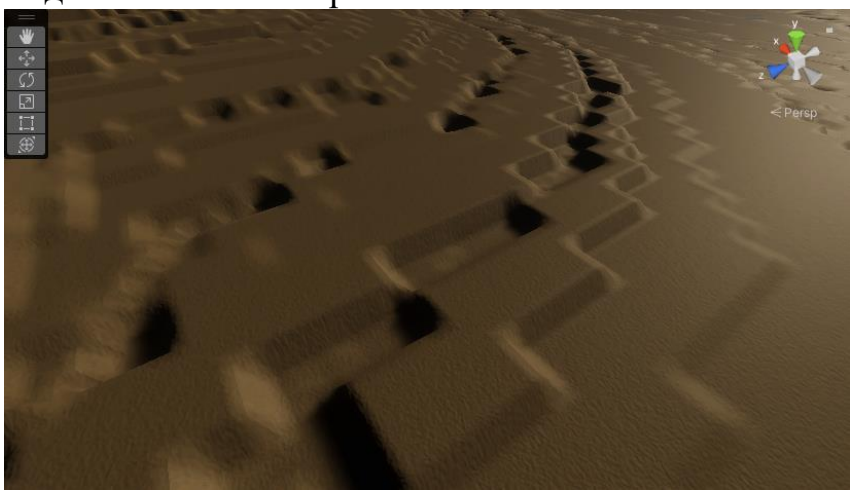

Рисунок 3.9 – Артефакты при генерации по карте высот.

Чтобы это исправить, мною была добавлена функция сглаживания поверхности. Как показала практика, такое решение является очень удобным, так как пользователь сам может задавать фактор сглаживания и регулировать силу сглаживания.

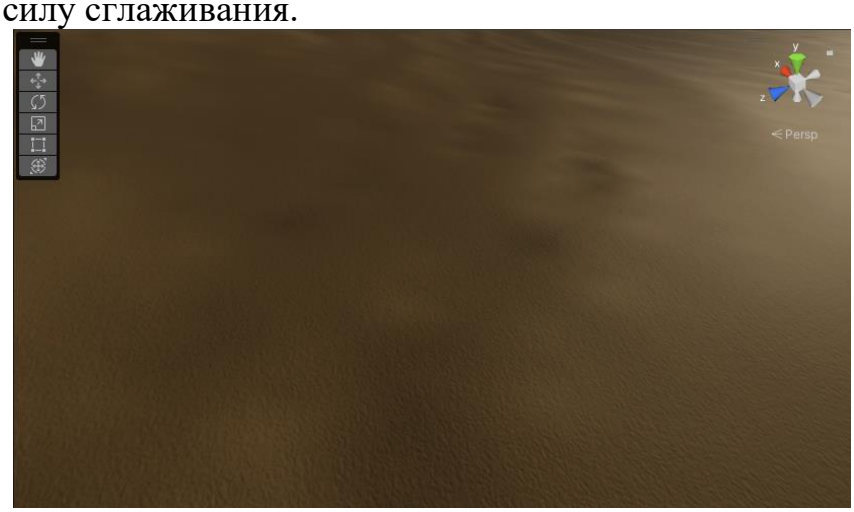

Рисунок 3.10 – Результат сглаживания той же поверхности, что и на рисунке 3.9.

3) Texture Parameters.

Секция Texture Parameters предназначена для настройки и применения текстур для объекта Terrain. Эта секция позволяет пользователю настраивать различные параметры текстур: размер и смещение плитки текстуры, диапазон высот, в которых будет применяться текстура, а также параметры шума Перлина. Структура секции состоит из слоев текстур, которые пользователь добавляет через инспектор объекта Terrain (Terrain

Layers). Каждый слой имеет свои параметры независимо от других, и накладываются текстуры также по отдельности. Пользователь может накладывать текстуру двумя способами: в диапазоне высот и на весь ландшафт. Выбор такого наложения определяется через параметр Range Texturing. Если пользователь выбирает наложение в диапазоне, то ему становится доступна функция плавного затухания текстуры, для создания равномерного перехода одной текстуры к другой. Также предоставлена возможность накладывания текстуры через шум Перлина.

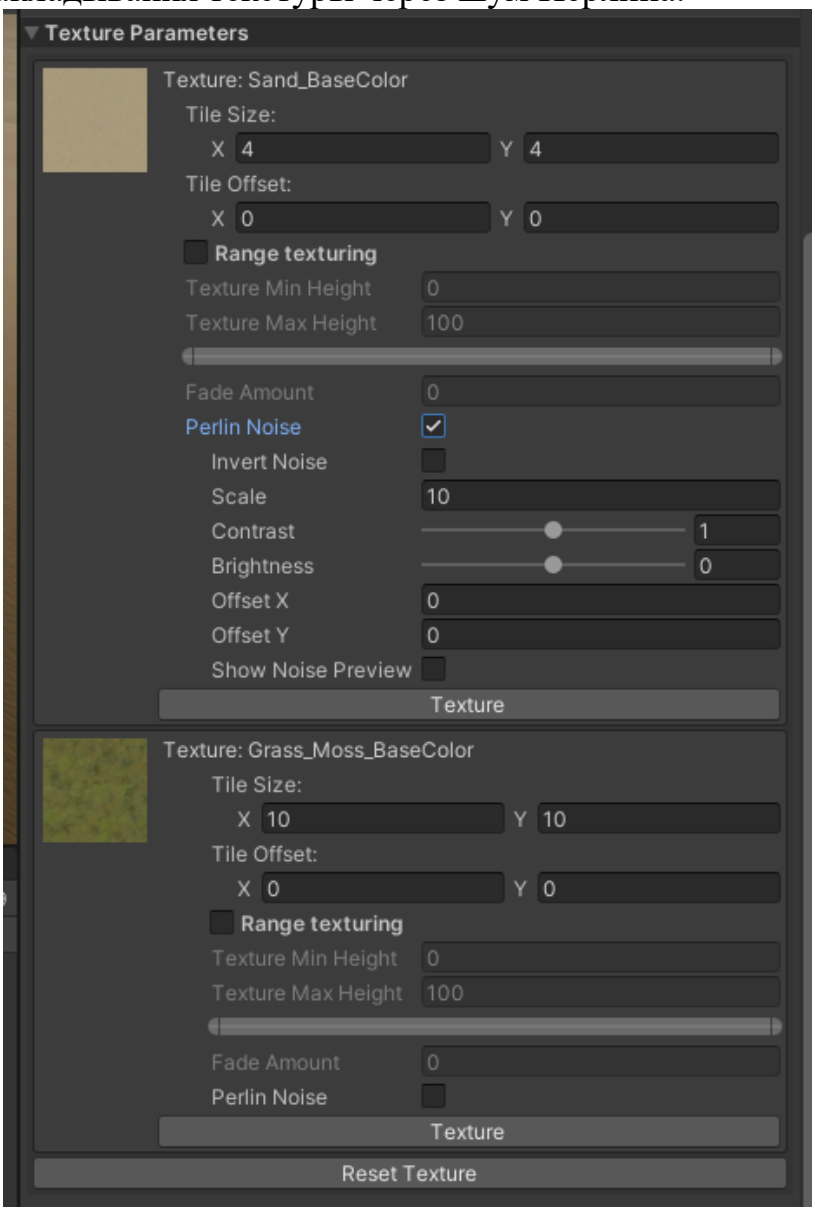

Рисунок 3.11 – Интерфейс секции настройки текстурирования

Для шума Перлина была создана структура PerlinNoiseSettings. Она содержит в себе различные параметры, изменяющие внешний вид шума:

- InvertNoise – флаг для инвертирования значений шума. Визуально, черное становится белым, и наоборот.
- Scale – масштаб шума Перлина. Этот параметр контролирует размер узора шума.

- Contrast – контраст шума. Контролирует разницу между яркими и темными областями в узоре шума. Высокие значения контраста увеличивают различие между пиками и впадинами шума, низкие значения делают картинку нейтрально серой.
- Brightness - яркость шума. Этот параметр регулирует общую яркость узора шума.
- Offset X и Offset Y – горизонтальное и вертикальное смещение шума. Служит в основном для создания разных паттернов узора.
- Show Noise Preview – флаг для включения или выключения предварительного просмотра шума. При включенном состоянии позволяет в реальном времени наблюдать за изображением шума Перлина, которое в последствии будет применено к ландшафту.

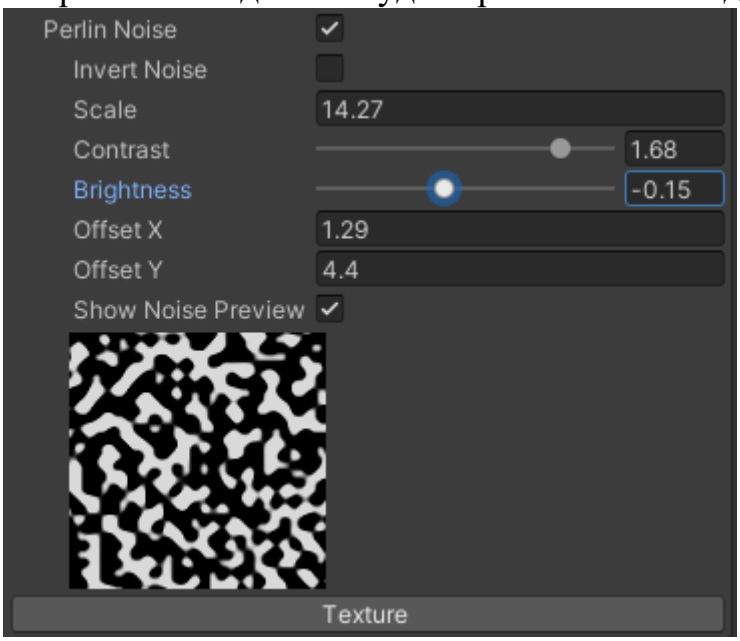

Рисунок 3.12 – предварительный просмотр шума.

Вывод слоев в секции Texture Parameters происходит аналогично, как и в предыдущей секции. В цикле происходит логика, заметно сложнее чем в Heightmap Parameters. Параллельно со слоями ландшафта TerrainLayer, существует массив типа структуры TextureLayer. Структура TextureLayer предназначена для хранения настроек текстурного слоя, который применяется к ландшафту.

Листинг 3.8 – Структура TextureLayer.

```
public struct TextureLayer
{
        public string textureName;
        public bool isRangeTexturing;
        public float minHeight;
        public float maxHeight;
        public bool isPerlin;
        public float fadeDistance;
        public PerlinNoiseSettings perlinNoiseSettings;
        ...
        public string GetTextureName()
```

```
        {
            return textureName;
        }

        public void SetTextureName(string textureName)
        {
            this.textureName = textureName;
        }
}
```

Каждый параметр этой структуры отвечает за определенные свойства текстурирования:

- textureName – Имя текстуры. Этот параметр используется для идентификации текстуры, которая будет применена к ландшафту.
- isRangeTexturing – Флаг, указывающий, будет ли применяться текстура в диапазоне высот. Если значение true, текстура будет применяться в диапазоне высот, заданных параметрами minHeight и maxHeight.
- minHeight – Минимальная высота, при которой будет применяться текстура. Этот параметр используется только в случае, если isRangeTexturing равно true.
- maxHeight – Максимальная высота, при которой будет применяться текстура. Этот параметр используется только в случае, если isRangeTexturing равно true.
- isPerlin – Флаг, указывающий, будет ли применяться шум Перлина для текстурирования. Если значение true, параметры шума Перлина, заданные в perlinNoiseSettings, будут применены к текстуре.
- fadeDistance – Расстояние затухания текстуры. Этот параметр определяет, насколько плавно текстура будет переходить в другие текстуры или в поверхность ландшафта.
- perlinNoiseSettings – Настройки шума Перлина. Этот параметр хранит настройки, которые будут применены к текстуре, если isPerlin равно true.

В программе массив типа этой структуры требуется для того, чтобы при изменении TerrainLayers не сбрасывались настройки каждого слоя. Синхронизация массива TextureLayers со всеми текстурами ландшафта происходит перед выводом всех элементов секции.

Листинг 3.9 – Функция синхронизации текстур.

```
void ApplyTextureLayers()
{
    TerrainLayer[] terrainLayers =
_terrainData.terrainLayers;
    if(textureLayers.Length == 0 && terrainLayers.Length
== 0)
    {
```

```
            return;
        }
        if(textureLayers.Length == 0)
        {
            textureLayers = new
    TextureLayer[terrainLayers.Length];
            Array.ForEach(textureLayers, layer =>
            {
                layer = new TextureLayer();
            });
        }
        if(textureLayers.Length < terrainLayers.Length)
        {
            Array.Resize(ref textureLayers,
    terrainLayers.Length);
        }
        for(int i = 0; i < terrainLayers.Length; i++)
        {
            if(!Array.Exists(textureLayers, layer =>
    layer.GetTextureName() ==
    terrainLayers[i].diffuseTexture.name))
            {
                textureLayers[i] = new
    TextureLayer(terrainLayers[i].diffuseTexture.name);
            }
        }
    }
```

Функция проходит по условиям, которые подготавливают массив к возможным изменениям, и далее вносит требуемые изменения, добавляя отсутствующие текстуры.

В цикле вывода секции, также происходит изменение параметров элементов массива TextureLayers и вызов функции ApplyTexture. В этой функции происходит накладывание текстуры на ландшафт с учетом всех параметров слоя. Функция проходит по каждому элементу сетки TerrainData.alphamapWidth * TerrainData.alphamapHeight и накладывает текстуру в соответствии с описанными ранее правилами и параметрами. Также есть важное свойство, что на ландшафт с размером, отличным от разрешения высотной карты, текстуры накладываются некорректно. Поэтому в самом начале мы изменяем размер ландшафта до размера высотной карты и работаем с ним.

Листинг 3.10 – Начало работы с модифицированным объектом Terrain.

```
void ApplyTexture(int index, TextureLayer layer)
{
    Undo.RegisterCompleteObjectUndo
(_terrainData.alphamapTextures, "Apply Texture");
    float[, ,] alphaMaps = _terrainData.GetAlphamaps(0, 0,
_terrainData.alphamapWidth, _terrainData.alphamapHeight);
```

```
    //Меняем размер террейна ддя корректного наложения
текстур
    Vector3 origScale = new Vector3(_terrainData.size.x,
_terrainData.size.y, _terrainData.size.x);
    Vector3 modifScale = new
Vector3(_terrainData.heightmapResolution,
_terrainData.size.y, _terrainData.heightmapResolution
);
    _terrainData.size = modifScale;

    for (int x = 0; x < _terrainData.alphamapWidth; x++)
    {
        for (int y = 0; y < _terrainData.alphamapHeight; y++)
        {
            int terrainX = Mathf.RoundToInt(x *
_terrainData.size.x / _terrainData.alphamapResolution);
            int terrainY = Mathf.RoundToInt(y *
_terrainData.size.z / _terrainData.alphamapResolution);
            float height = _terrainData.GetHeight(terrainY,
terrainX);

            float xCoord = x / _terrainData.size.x *
layer.perlinNoiseSettings.scale +
layer.perlinNoiseSettings.offsetX;
            float yCoord = y / _terrainData.size.z *
layer.perlinNoiseSettings.scale -
layer.perlinNoiseSettings.offsetY;
```

Всего существует 4 варианта накладывания текстуры, в зависимости от двух флагов: isRangeTexturing и isPerlin. В подготовительном этапе мы определяем параметры для будущих действий. После определения координат мы определяем значение шума Перлина на заданных координатах. Далее мы корректируем это значение в соответствии с заданными параметрами perlinNoiseSettings. Так, к примеру контрастность шума вычисляется по формуле:

$$0.5 - (0.5 - perlinValue) * contrast^4 \quad (3.1)$$

где *perlinValue* – значение шума Перлина в заданных координатах;
contrast – значение контраста в perlinNoiseSettings.

Текстурирование в диапазоне примечательно тем, что для эффекта затухания используется гиперболическая функция, выведенная мною, как и контраст шума, через графический калькулятор.

Листинг 3.11 – Определение параметров.

```
    //Основной шум
    float perlinValue = Mathf.PerlinNoise(xCoord, yCoord);
    //Контраст шума
    perlinValue = 0.5f - (0.5f - perlinValue) *
Mathf.Pow(layer.perlinNoiseSettings.contrast, 4);
    //Яркость шума
    perlinValue = Mathf.Clamp01(perlinValue) +
layer.perlinNoiseSettings.brightness;
```

```
    //Инвертировать шум
    perlinValue = layer.perlinNoiseSettings.invertNoise ? 1-
perlinValue : perlinValue;

    float distanceFromBorder = 0;
    float alpha = 0;
    if(layer.isRangeTexturing)
    {
        if(height   >=   layer.minHeight   &&   height   <=
layer.maxHeight)
        {
            //Выбираем границу
            distanceFromBorder                                  =
Mathf.Min(Mathf.Pow(layer.maxHeight     -     height,     1),
Mathf.Pow(height - layer.minHeight, 1));
            //Гиперболическая функция затухания
            alpha = 1 - 1 / (1 + Mathf.Pow(distanceFromBorder
/ layer.fadeDistance, 2));
        }
        perlinValue *= alpha;
        perlinValue = Mathf.Clamp01(perlinValue);
    }
    else
    {
        alpha = 1;
    }
```

Далее уже вычисляется значение элемента альфакарты. В зависимости от того, выбрано ли текстурирование с использованием шума Перлина или без. Единственное отличие двух веток заключается в том, какое значение вычитать или прибавлять.

Листинг 3.12 – Вычисление значения элемента альфакарты.

```
    if(layer.isPerlin)
    {
        for (int i = 0; i < _terrainData.alphamapLayers; i++)
        {
            alphaMaps[x, y, i] = (i == index) ? alphaMaps[x,
y, i] + perlinValue : alphaMaps[x, y, i] - perlinValue;
        }
    }
    else
    {
        for (int i = 0; i < _terrainData.alphamapLayers; i++)
        {
            alphaMaps[x, y, i] = (i == index) ? alphaMaps[x,
y, i] + alpha : alphaMaps[x, y, i] - alpha;
        }
    }
```

После прохождения всех итераций двойного цикла, мы получаем новую альфакарту всех текстур с измененными значениями. Последнее, что

останется сделать, это применить новую альфакарту к нашему объекту Terrain через специальную функцию TerrainData.SetAlphamaps, и вернуть прежний размер ландшафта.

Также у секции изменения параметров текстурирования есть кнопка, которая сбрасывает все текстуры. Сброс текстур заключается в том, чтобы задать максимальный вес одной текстуре и обнулить веса всех других текстур массива TerrainLayer. В моем случае при сбросе ландшафт покрывается самой первой текстурой в массиве TerrainLayer.

4) Tree Parameters

Секция Tree Parameters предназначена для настройки и применения объектов, в частности деревьев и кустов для объекта Terrain. Эта секция позволяет пользователю настраивать различные параметры деревьев: выбор генерируемых объектов, максимальный угол склона, на котором возможна генерация, диапазон высот, в который будут генерироваться объекты, диапазон между минимальным и максимальным размером объекта, количество генерируемых деревьев, а также генерация по шуму Перлина с использованием структуры, описанной в прошлой секции. Структура секции состоит из списка объектов TerrainData.treePrototypes, которые пользователь добавляет через инспектор объекта Terrain. В отличие от текстур, здесь все объекты списка treePrototypes имеют общие параметры, пользователь в свою очередь может с помощью флагов у каждого объекта выбирать, какие будут использованы при генерации.

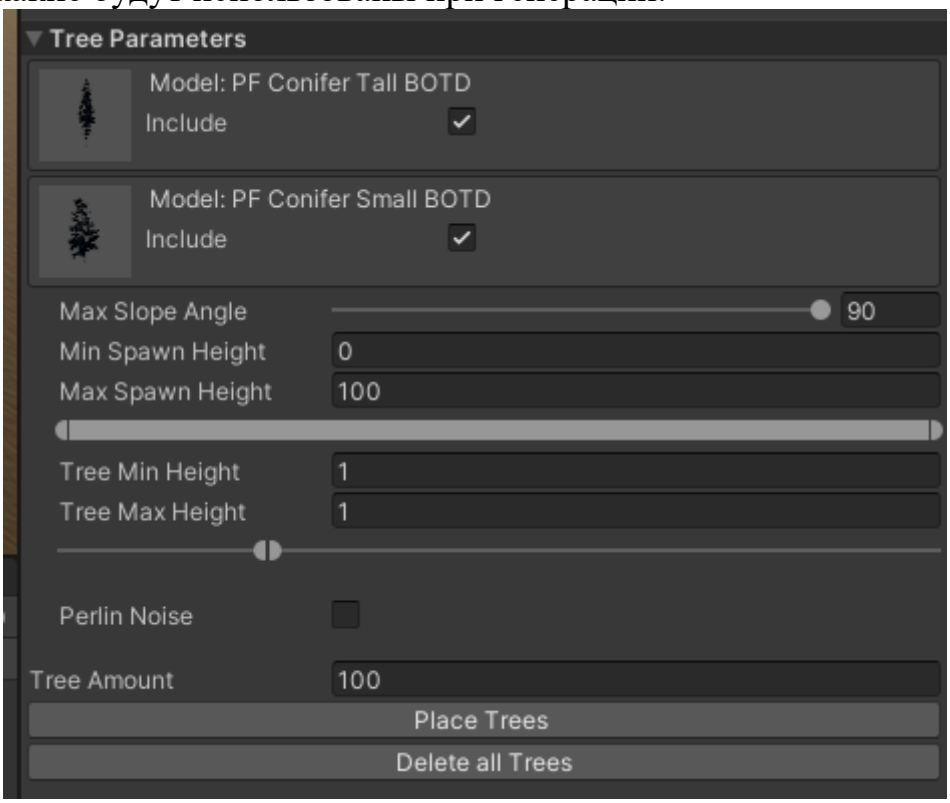

Рисунок 3.13 – Интерфейс секции Tree Parameters.

Из новых параметров здесь не появилось чего-то стоящего упоминания, поэтому перейдем к описанию кода.

Принцип работы очень схож с предыдущей секцией – здесь также используется цикл для вывода деревьев, также существует дополнительный массив enableTree, который хранит в себе значения флага Include, который в этой секции имеется у каждого объекта в массиве. Перед выводом интерфейса происходит проверка массива treePrototypes и массива enableTree на соответствие размеров. Если элементов в enableTree меньше, чем в treePrototypes, тогда добавляются новые элементы, если больше, то удаляются. В конечном итоге эти массивы должны быть одной длины.

Листинг 3.13 – Функция проверки массива enableTree.

```
bool[] ApplyTreePrototypes()
{
    TreePrototype[] treePrototypes =
_terrainData.treePrototypes;
    bool[] newEnableTree;
    if(enableTree == null)
    {
        newEnableTree = Enumerable.Repeat(true,
treePrototypes.Length).ToArray();
        return newEnableTree;
    }
    if(enableTree.Length != treePrototypes.Length)
    {
        newEnableTree = Enumerable.Repeat(true,
treePrototypes.Length).ToArray();
        int arrayLength = (enableTree.Length <
treePrototypes.Length) ? enableTree.Length :
treePrototypes.Length;
        for (int i = 0; i < arrayLength; i++)
        {
            newEnableTree[i] = enableTree[i];
        }
        return newEnableTree;
    }
    return enableTree;
}
```

При разработке этой секции я столкнулся с совершенно другим подходом – теперь требуется генерировать строго заданное число деревьев. Но очевидно, что при заданных диапазонах будут появляться ограничения, и соответственно не получиться просто работать через цикл от 1 до заданного количества деревьев, так как не всегда будет генерироваться нужное количество. Если же задавать выполнение генерации, пока количество сгенерированных объектов не станет равным заданному количеству, то могут возникнуть бесконечные циклы, например, если задать диапазон от 0 до 0. Чтобы избежать эти ситуации, я заключил генерации случайных значений в циклы попыток. Условно, если за 200 итераций

алгоритм не смог найти случайные значения, то он просто пропускает итерацию.

На этом принципе работает вся функция генерации объектов на поверхности ландшафта. В самом начале дается 200 попыток на генерацию координат, чтобы высота в данной точке попадала в диапазон высот. Далее, если угол наклона в этой точке меньше заданного и вероятность генерации в этой точке попадает в диапазон, то программа идет дальше. На это дается 100 попыток. После этого подпирается случайный индекс из массива выбранных нами объектов. Только после этого мы формируем новый TreeInstance и добавляем его на наш ландшафт.

Листинг 3.14 – Пример цикла попыток генерации случайных координат.

```
    //Попытка подбора высоты в нужном диапозоне допустимых
высот
    for(int _try2 = 0; _try2 < 200; _try2++)
    {
        if(minTreeSpawnLimit < y && y < maxTreeSpawnLimit)
        {
            break;
        }
        x = UnityEngine.Random.Range(0f, _terrainData.size.x);
        z = UnityEngine.Random.Range(0f, _terrainData.size.z);
        position = new Vector3(x, 0f, z);
        y = terrainObject.SampleHeight(position);
    }
```

3.2 Пример использования программы

Основное предназначение приложения – предоставить разработчикам быстрый и удобный способ создания ландшафта, без применения особых усилий. Вот один из примеров создания ландшафта через модуль генерации:

1. Задание размера ландшафта и разрешения высотной карты.
   Задаем размер ландшафта 500*100*500 метров. Также для детальной поверхности ставим разрешение 2049.

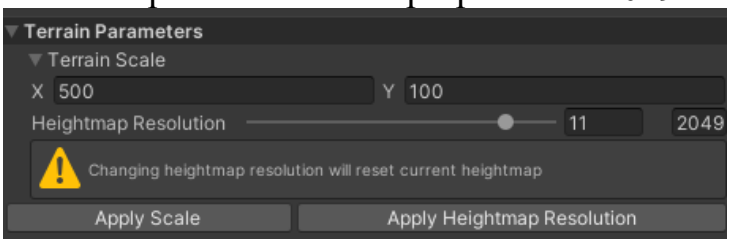

Рисунок 3.14 – Задание параметров.

   Нажимаем кнопки применения размера и разрешения и получаем следующий результат:

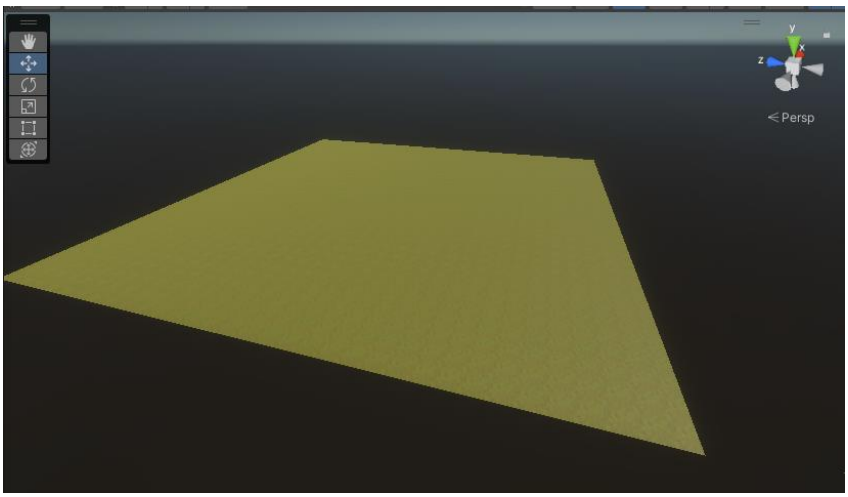

Рисунок 3.15 – Результат первого этапа.

2. Настройка высотной карты.
   Для начала добавим новый слой высоты и выберем тип генерации по карте. Не меняя настроек по умолчанию, только выбрав изображение, нажимаем кнопку генерации слоя. Чтобы избавиться от пикселей, сгладим поверхность на 9. Далее добавляем новый слой и ставим тип на шум Перлина с параметрами Perlin Scale равным 13 и Height Offset равным 0.2.

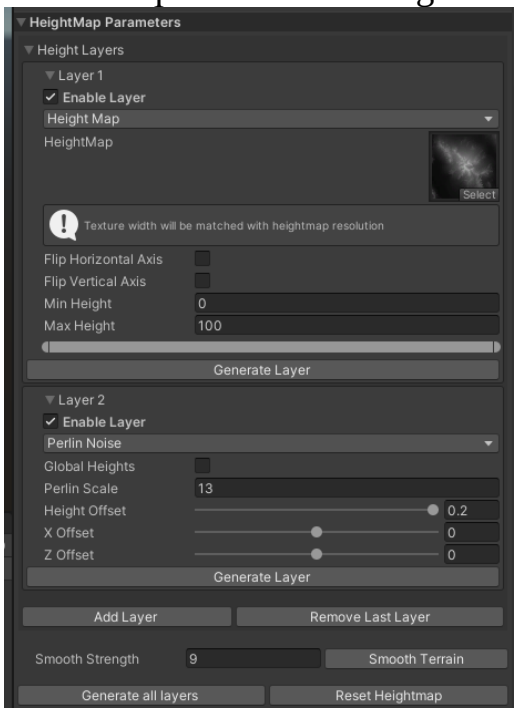

Рисунок 3.16 – Параметры высотной карты.

Нажимаем сгенерировать и получаем следующий результат:

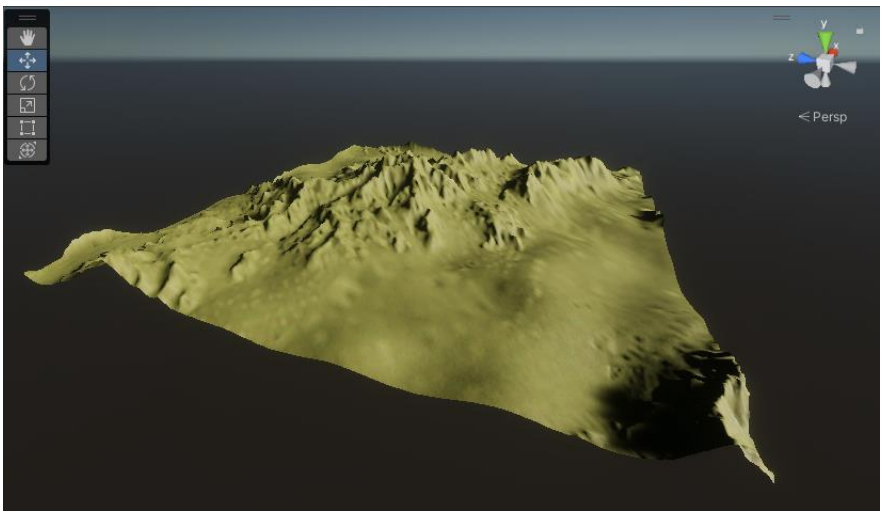

Рисунок 3.17 – Результат второго этапа.

3. Нанесение текстур.
   У текстуры щебени ставим флаг Range texturing, и задаем диапазон от 30 до 100. Ставим затухание на 10 и применяем текстуру. Далее у этой же текстуры ставим диапазон от 20 до 100, затухание на 5, отмечаем шум Перлина и ставим контраст на 1.2 и также нажимаем кнопку текстурирования. Теперь добавим немного динамики полям. Для текстуры сухой травы применим следующие параметры:

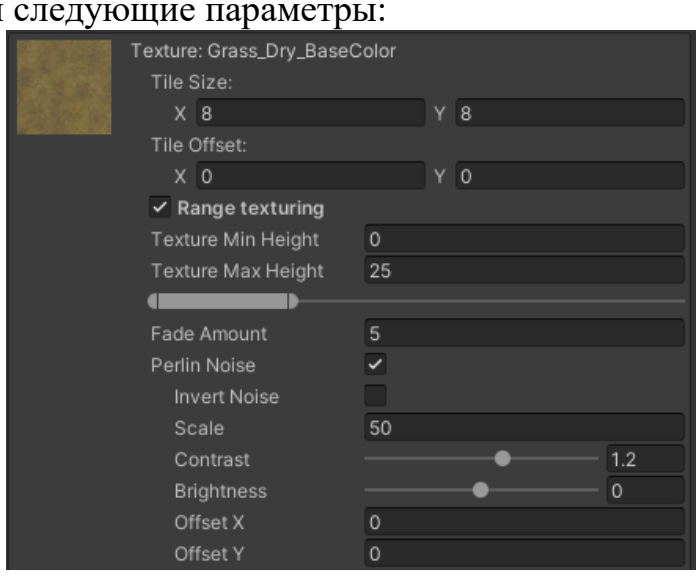

Рисунок 3.18 – Параметры текстуры.

После всех этих манипуляций мы получаем:

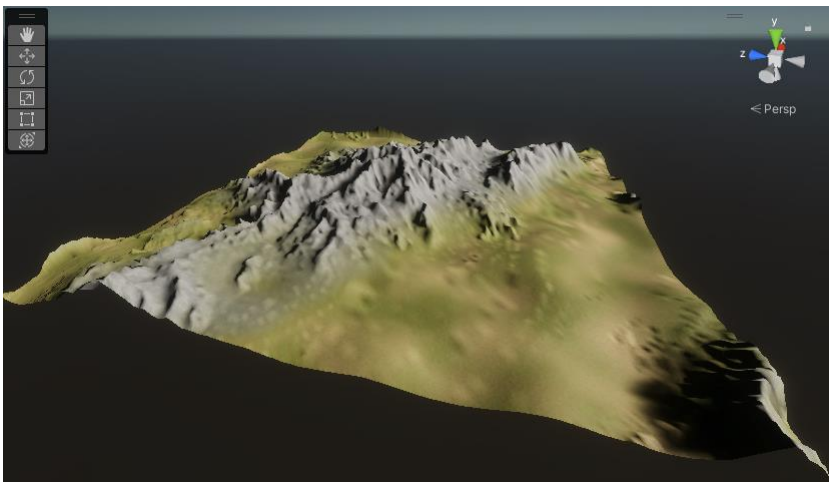

Рисунок 3.19 – Результат третьего этапа.

4. Генерация деревьев.

Для генерации деревьев выставим следующие параметры:

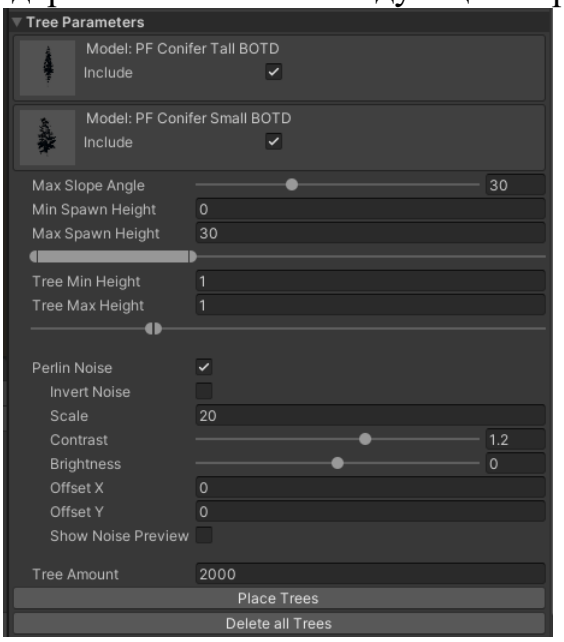

Рисунок 3.20 – Параметры генерации деревьев.

В результате чего мы получаем следующий ландшафт:

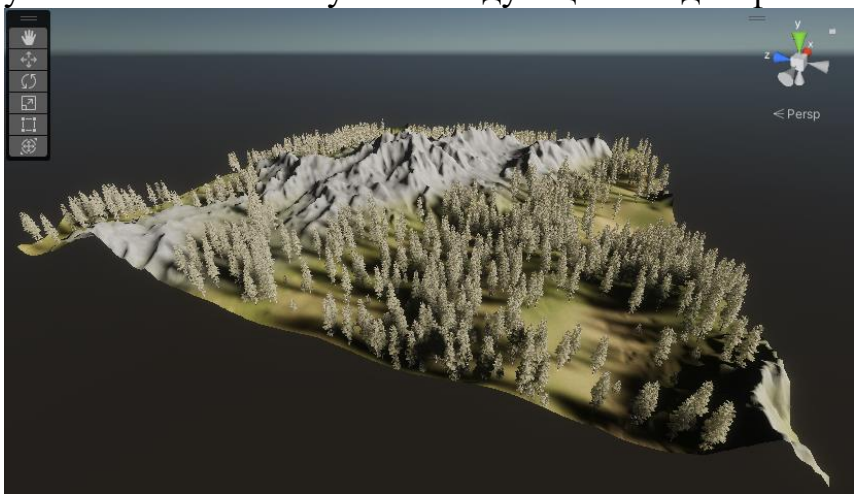

Рисунок 3.21 – Результат последнего этапа.

После этого можно добавить пост-эффекты, подобрать ракурс камеры и у нас получится пейзаж, на который без моей программы ушло бы как минимум в пять раз больше времени.

Рисунок 3.22 – Финальный кадр.

ЗАКЛЮЧЕНИЕ

Подведем итоги проделанной работы: в ходе выполнения выпускной квалификационной работы была поставлена и решена задача разработки модуля генерации ландшафта для Unity3d, способствующего ускорить и упростить процесс создания ландшафта. В рамках работы были достигнуты следующие результаты:

1. Изучение алгоритмов процедурной генерации.
   Были рассмотрены основные методы и алгоритмы процедурной генерации ландшафтов, включая использование шума Перлина и карт высот. Это позволило заложить теоретическую основу для разработки модуля.
2. Выбор средств разработки.
   Были определены и обоснованы основные средства разработки: игровой движок Unity3d, язык программирования C# и редактор кода Visual Studio Code. Эти инструменты были выбраны за их широкие возможности, удобство использования и поддержку современного функционала.
3. Разработка модуля генерации ландшафта.
   Были созданы и протестированы компоненты модуля, обеспечивающие генерацию ландшафта на основе карт высот и шума Перлина. В частности, реализованы классы и структуры для управления параметрами генерации, работы с текстурами и объектами.
4. Тестирование и оптимизация.
   Проведены испытания модуля на различных сценах и условиях. Выявлены и устранены основные ошибки и недостатки, проведена оптимизация кода для обеспечения высокой производительности и стабильности работы.
5. Документация и описание работы.
   Подготовлены детальные описания каждого этапа разработки, а также инструкции по использованию модуля. Это позволяет другим разработчикам легко интегрировать и использовать созданный модуль в своих проектах.

Созданный модуль генерации ландшафта для Unity3d доказал свою эффективность и применимость для разработки игр и интерактивных приложений. Он позволяет значительно ускорить процесс создания сложных и реалистичных ландшафтов, обеспечивая высокое качество и разнообразие генерируемых пространств. В будущем возможна дальнейшая доработка и расширение функционала модуля, включающее поддержку новых алгоритмов генерации и улучшение пользовательского интерфейса.

Работа над проектом позволила мне углубить знания в области процедурной генерации, освоить новые инструменты и методы разработки, а также получить ценный практический опыт, который будет полезен в дальнейшей профессиональной деятельности.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Unity Technologies, Платформа Unity для разработки в реальном времени [Электронный ресурс]. URL: https://unity.com/ru (дата обращения: 15.06.2024) [1]
2. Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. Классика Computers Science [Текст] / Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. — 4-е изд.. — СПб: Питер, 2012 — 784 с. [2]
3. Перлин К., Делая шум [Электронный ресурс]. URL: http://www.noisemachine.com/talk1/ (дата обращения 15.06.2024) [3]
4. Unity Documentation / [Электронный ресурс] // Unity - Scripting API : [сайт]. — URL: https://docs.unity3d.com/ScriptReference/index.html (дата обращения: 17.06.2024) [4]
5. Хокинг Дж., Unity в действии. Мультиплатформенная разработка на C#. - М.: Питер, 2018. [5]