

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ**

Направление: 09.03.02 Информационные системы и технологии

Профиль: Информационные системы в образовании

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка компьютерной игры «Ковбой» на движке Unreal Engine

Студент 4 курса

Группы 09-862

«__» _____ 2022 г. _____ (Н.А. Насридинов)

Научный руководитель:

канд. пед. наук, доцент

«__» _____ 2022 г. _____ (Ч.Б. Миннегалиева)

Заведующий кафедрой

канд. физ.-мат. наук, доцент

«__» _____ 2022 г. _____ (Ф.М. Гафаров)

Казань – 2022

Содержание

Введение	3
Глава 1. Основные принципы и методы разработки игр.	5
1.1. Развитие компьютерных игр	5
1.2. Программное обеспечение для разработки игры	7
Глава 2. Основы Unreal Engine	11
2.1. Архитектура игрового движка UE	12
2.1.1. Геометрии, Landscape. Foliage.....	13
2.1.2. Освещения и материалы.....	16
2.1.3. Основы физики и коллизии.....	22
2.1.4. Particle System	25
2.2. Реализация функционала на Blueprint и на C++.....	27
2.2.1. Основы Blueprint.....	28
2.2.2. Создание анимации с использованием Timeline	30
2.2.3. Основные классы и функции C++	31
2.2.4. Кодирование логики на C++	33
Глава 3. Программная реализация.....	37
3.1. Графическое оформление сцены.....	37
3.2. Применение анимации к персонажам.....	38
3.3. Система вооружения.....	45
3.4. Создание искусственного интеллекта бота NPC.....	48
3.5. Боевая система.	52
3.6. Cinematic.....	55
Заключение.....	61
Список литературы.....	62

Введение

Технологии были важным фактором роста нашей цивилизации, поскольку технологический прорыв стал ключевым моментом в развитии человечества. И игровая индустрия не является исключением, история видеоигр тесно связана с эволюцией компьютерных систем, уже с 1990-х годов видеоигры вошли в новое измерение, потому что во второй половине десятилетия графика стала трехмерной. Теперь игроки могли двигаться в трех направлениях вместо двух. Эти игровые миры выглядят и ощущаются более реалистично, и предлагают более широкие возможности. С каждым годом технологии стремительно развиваются. Новые технологии графического рендеринга сделали видеомиры более реалистичными. За прошедшее десятилетие видеоигры превратились в крупную отрасль с миллиардным оборотом. Прибыль от них выше, чем у кино и музыкальной индустрии.

Компьютерные игры постепенно меняют повседневный мир во многих положительных аспектах. Играя в игры, мы забываем об усталости и начинаем получать удовольствие. Кроме того, они улучшают скорость работы мозга и навыки решения проблем.

Данная видео игра представляет Shooter от третьего лица, в художественном стиле Low poly, который перенесет игрока далеко на Дикий запад в городок под названием Striver city, погружая игрока в атмосферу «Мир дикого запада». У игрока есть возможность взаимодействовать с реалистичными неуправляемыми персонажами(NPC).

Актуальность компьютерных игр: Сегодня новой и популярной формой развлечения и важным применением мощных технологий являются компьютерные игры, которые становятся все более популярными среди людей всех возрастов. А также благодаря развитию технологий, графика стала более реалистичной, что заинтересует большое количество людей в

мир компьютерных игр, позволяя игроку глубоко погрузиться в увлекательные миры, наполненные удивительными приключениями и ощущениями.

Объект исследования: Разработка компьютерной игры.

Предмет исследования: Инструменты и программное оборудование для разработки игры-шутера от третьего лица.

Цель исследования: Разработка компьютерной игры «Ковбой» на движке Unreal Engine.

Для достижения цели необходимо было решить следующие задачи:

1. Изучение основных аспектов разработки компьютерных игр.
2. Определить жанр, дизайн игры, движок для разработки игры.
3. Изучить основные функциональные возможности среды Unreal Engine.
4. Подготовить все необходимые материалы для разработки игры.
5. Разработать и оптимизировать игру.

Глава 1. Основные принципы и методы разработки игр

Разработка игры это нечто больше чем просто программирование, это искусство, в котором много нюансов и тонкостей. Разработка компьютерных игр включает в себя создание и подбор музыки, графики, системы управления, искусственный интеллект (ИИ), человеческий фактор и ряд других процессов.

1.1. Развитие компьютерных игр

Первые компьютерные игры появились в шестидесятых и семидесятых годах 20 века, начиная с Pong в семидесятых, видеоигры эволюционировали от простых цифровых игр с парой элементов управления до более сложных объектов с множеством персонажей, сюжетных линий, механик и так далее. В этой многомилиардной индустрии разработчики и издатели игр ежедневно внедряют инновации, чтобы поддерживать интерес сообщества. По мере развития технологий и увеличения количества более мощных компьютеров и консолей видеоигры все больше приближались к реалистичной графике, а также потребители были заинтересованы в оплате игровых услуг, поэтому можно сказать, что видеоигры стали частью культуры народа, и польза от них только увеличивается с каждым годом.

Компьютерные игры - это программы, которые позволяют игроку взаимодействовать с виртуальной игровой средой для развлечения и получения удовольствия. Существует множество жанров компьютерных игр, начиная от традиционных карточных и, заканчивая более сложными, такими как шутер и приключенческие игры. Компьютерные игры служат не только для развлечения, но и развивают когнитивных функций, такие как зрительно-мониторная координация, критическое мышление, многозадачность и т.д.

Каждая видеоигра имеет свой собственный стиль, стратегию и фантазию, что делает её, уникальной и интересной. Как известно, видеоигры можно разделить на разные типы и жанры. Само слово жанр определяется как категория художественного, музыкального или литературного произведения, характеризующееся определенным стилем, формой или

содержанием. Таким образом, жанр в компьютерных играх, как правило, описывает стиль игрового процесса, а популярность определенных жанров в этой области помогает разработчикам определить направление своего следующего проекта. Современные видеоигры не сосредоточены на каком-то определенном жанре, и в результате в одной игре может быть несколько жанров, причем одни считаются определяющими жанрами, а другие – поджанрами. Таким образом, определяющим жанром является основной геймплей игры, а другие жанры, присутствующие в меньшей степени, являются поджанрами.

На данном этапе развития игровой индустрии выделяют следующие современные жанры компьютерных игр:

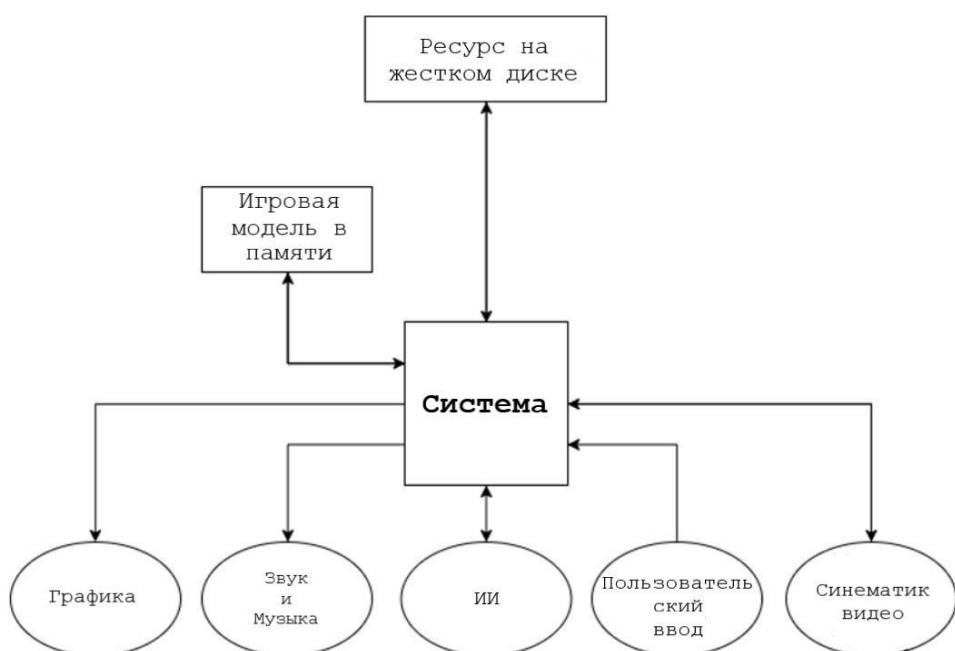
- Экшн. Этот жанр является одним из самых ранних жанров в индустрии видеоигр, экшн игры требуют от игрока быстрой реакции и точного расчета времени.
- Приключенческие игры. Игры в жанре приключения, как правило, это медленные и интеллектуальные игры в которых игрок должен исследовать мир.
- Ролевые игры(RPG). Игры этого жанра в основном являются иммерсивными играми, использующими обширное и детализированное построение мира, в которых игроки развивают своих персонажей, например, Ведьмак или Red Dead Redemption 2.
- Стратегические игры. Игры жанра стратегии в первую очередь предоставляют игрокам полный контроль над игровым миром или цивилизацией, от его зарождения и развития до окончательного формирования. В таких играх игроки с помощью тщательно продуманной тактики преодолевают пошаговые испытания или как игры в реальном времени, соревнуются друг с другом или с искусственным интеллектом, захватывая различные участки игрового мира.

- Шутер. Один из самых популярных жанров видеоигр, этот жанр прошел долгий путь, усовершенствовался и сформировал свои собственные, отдельные миры благодаря новым уровням сложности и повествования, таким образом, сегодня существует два направления жанра шутера: шутер от первого лица и от третьего лица. К примеру, такие игры, как Call Of Duty или Battlefield, широко известны как игры-шутеры.

В дополнение к этим основным жанрам, существует еще множество жанров, таких как спортивные, гоночные, игры-головоломки, файтинги и т.д. Когда цель проекта поставлена, и выбор жанра сделан, необходима, определить среду, в которой будем реализовать нашу идею.

1.2. Программное обеспечение для разработки игры

Когда мы думаем о правилах игры, сюжете, целях, задачах и о том, как игрок должен взаимодействовать с ними, мы думаем об игровом процессе. А игровой процесс — это, по сути, то, как игрок взаимодействует с игрой. Игровая механика помогает обеспечить игровой процесс, предоставляя конструкцию методов или правил, предназначенных для взаимодействия игрока. Диаграмма игровой механики представлена ниже.



Игровая система.

Как видно на схеме выше, игровая система берет на себя следующие задачи:

- Управляет загрузкой и сохранением файлов.
- Затем он позволяет запускать анимацию с использованием аудио и графических элементов.
- Во время использования игрового интерфейса пользовательский ввод обеспечивает гармоничную работу графической и звуковой части игры.
- Наконец, игровая система гарантирует, что графика, звук, видео, пользовательский ввод и искусственный интеллект работают в игре в гармонии.

Ресурсы.

Необходимые файлы для игры, расположенные на жестком диске.

1. Эти ресурсы содержат файлы конфигурации, описывающие настройки проекта, в котором будет работать игра.
2. Если игра запущена и в ней содержится видеофайл, он будет воспроизводиться по желанию пользователя.
3. Содержит файлы, предоставляющие пользователю информацию о шкале здоровья в игре.
4. Он содержит файлы конфигурации, файлы изображений, музыкальные и звуковые файлы, файлы 3D-моделей и видеофайлы, а также другие элементы, которые потребуются игре для отображения визуальных данных игроку.

Модель игры.

1. Игровая модель — это состояние игровой графики, музыки и текстовых файлов, загруженных в память с ресурса.
2. Файлы изображений хранятся в памяти или в текстурной памяти видеокарты.
3. Информация о разделе, счете и т. д. хранится в переменных и структурах данных в игровой модели.

Графика.

Как правило, это самая важная и сложная часть игры. Потому что невозможно иметь игру без визуальных эффектов.

1. Воспроизведение любого видео,
2. Отображение игрового интерфейса.
3. Просмотр и отображение результатов, справки и информации о конфигурации.
4. Отображение собственной графики и визуальных элементов игры.

Звук и музыка.

Музыка — это инструмент, который может контролировать эмоции и задавать тон с помощью атмосферных треков во время игры.

1. Музыка играет фоном.
2. Звуковые эффекты для столкновений и движений.

Искусственный интеллект (ИИ)

Чем реалистичнее они выглядят и чем умнее их поведение, тем интереснее будет игра. И это не обязательно должны быть враги. В игре они могут просто ходить вокруг, они могут даже напасть на игрока, если с ними плохо обращаются.

Пользовательский ввод.

Игра без взаимодействия с пользователем невозможна, иначе это было бы похоже на просмотр фильма. Взаимодействие с пользователем - самая важная часть игровой механики, которая служит связующим звеном между пользователем и игрой. Пользовательский ввод — это часть игровой механики, в которой мы используем аппаратное обеспечение и оборудование, позволяющие пользователю взаимодействовать с игрой.

Кинематографические ролики.

Такие ролики имеют решающее значение для мира видеоигр. Они используются для объяснения сюжета с участием игрока в решающие моменты, а также они позволяют привлечь новую аудиторию за счет

создания зрелищных трейлеров. Таким образом, они служат множеству целей, от продвижения сюжета игры до продвижения ее выпуска.

Программное обеспечение.

Самый первый и важный инструмент для разработки игры, которое можем иметь в своем наборе инструментов — это те, которые помогут нам создать план. Многие начинающие разработчики игр бросаются создавать игру, не планируя цели разработки игры, не зная, в каком направлении они хотят двигаться, и на какую аудиторию рассчитана их игра.

Разработчики и команды разработчиков сначала придумывают идеи, как сделать свою игру уникальной и интересной, включая общий смысл игры, как разворачивается история, структура уровней и объектов игры. Не все игры строятся на замысловатых историях, сложных сюжетах или сложных структурах уровней, но важно определить цель и стиль своей игры, чтобы помочь игрокам понять, почему они должны попробовать сыграть в нее.

Игры можно разрабатывать в чистом коде на различных языках программирования, но разработчику нужны дизайнеры, инструменты рендеринга, компиляторы, графические библиотеки и т.д. Чтобы упростить процесс создания игр, были изобретены игровые движки. Игровой движок - это комплекс программных инструментов для манипулирования графическими элементами, игровыми объектами, скриптами, звуковыми эффектами и так далее. Следовательно, игровые движки позволяют быстро реализовать основные игровые функции, включая физику, рендеринг, написание сценариев, обнаружение столкновений и многое другое, без необходимости привлечения других специалистов. Безусловно, наиболее распространенными платформами, используемыми профессиональными игровыми студиями, являются Unity и Unreal, и, что удивительно, обе платформы бесплатны для разработки.

Untiy является платформой для создания игр и включает в себя интегрированную среду разработки, которая позволяет командам

разработчиков создавать уникальные 2D или 3D игры, с широким увлекательным миром. Помимо 2D и 3D игр, разработчики могут создавать интерактивные симуляторы для видеоигр и анимацию для фильмов.

Программное обеспечение для разработки игр Unreal Engine 4 является одним из лидеров среди игровых движков для разработки игр и при этом стоит, обратить внимание, что Unreal создан для профессионалов и в руках опытных разработчиков этот движок способен показать всю свою мощь во всей красе. По сравнению с движком Unity, Unreal имеет большое преимущество в графической составляющей, поддерживаемой Nvidia, а также Unreal предоставляет более продвинутые инструменты для достижения желаемых результатов. Unity больше подходит для инди-студий, в то время как Unreal используется крупными ведущими студиями. Платформа UE использовалась для таких культовых игр, как Mass Effect от BioWare, которая обладает великолепным сюжетом, а также для множества других известных игр, таких как Fortnite, BioShock: Infinite и др. На данный момент Unreal Engine остается одним из самых востребованных движков, очень гибким и простым в использовании.

Глава 2. Основы Unreal Engine

Unreal Engine - это игровой движок, разработанный компанией Epic Games в 1998 году, и его структура, написанная на C++, делает его лучшим игровым движком для разработчиков. Помимо кода, Unreal Engine поддерживает язык Blueprint, написанный специально для него. Blueprint, в свою очередь, представляет собой систему визуального программирования, в которой логика строится путем последовательного соединения узлов. Платформа UE предназначена не только для создания игр, но также и для проектирования дизайна интерьера и кинематографического искусства.

Движок предоставляет набор инструментов мирового класса для разработки самых разнообразных продуктов.¹

2.1. Архитектура игрового движка UE

Движок Unreal Engine в свою очередь состоит из нескольких компонентов, которые в совокупности позволяют запустить игру. Его обширная система инструментария и редакторы позволяют организовывать и манипулировать объектами для создания геймплея игры. В число основных компонентов движка Unreal Engine входят звуковой модуль, физический модуль, графический модуль и система ввода.

Звуковой механизм используется для воспроизведения музыки и звуков в игре. Наличие этого модуля позволяет воспроизводить различные звуковые файлы для придания атмосферы и добавления реалистичности. Звуки в игре можно использовать по-разному, например, в фоновом режиме, или использоваться по мере необходимости и срабатывать в определенных случаях, которые называются звуковыми эффектами.

В игровой среде, чтобы объекты реагировали так, как они реагируют в реальной жизни, мы должны включить функцию физики в те объекты, которым нуждаются законам физики. Unreal использует программный модуль PhysX, разработанный NVIDIA, для выполнения расчетов реалистичных физических взаимодействий, столкновений и т.д.

Графический механизм движка позволяет реализовать реалистичные картины при создании игры. Благодаря его способности оптимизировать сцену и обрабатывать огромное количество вычислений в режиме реального времени для обеспечения освещения, разработчики могут создавать реалистичные объекты в игре.

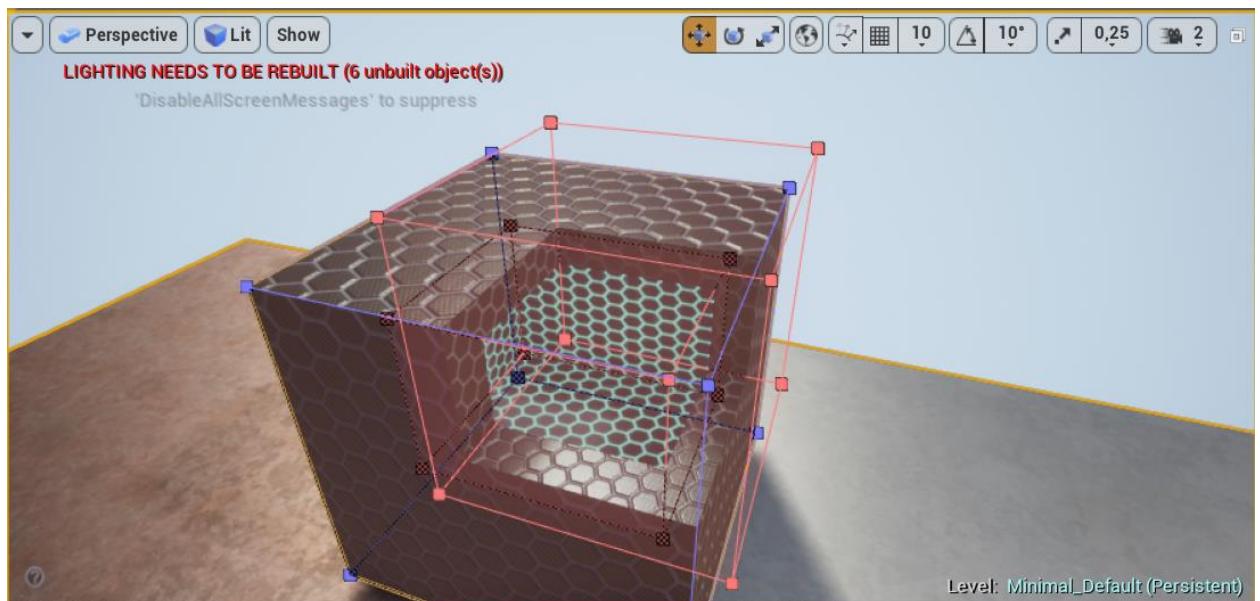
¹ Русскоязычное сообщество Unreal Engine 4 [Электронный ресурс] режим доступа: <https://uengine.ru/> (дата обращения: 12.12.2021).

Платформа UE состоит из системы вводов, которая позволяет преобразовать нажатия клавиш и кнопок игрока в конкретные действия, выполняемые игровым персонажем. Данная система ввода может настраиваться с помощью структуры игрового процесса.

2.1.1. Геометрии, Landscape, Foliage

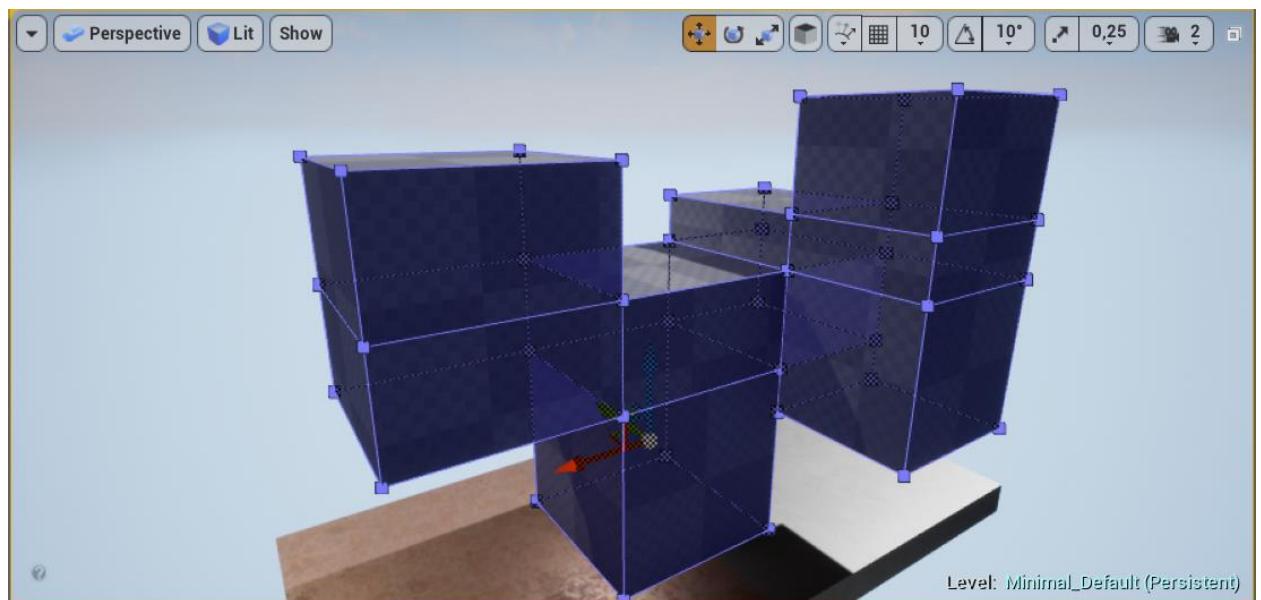
Геометрия является одной из важных составляющих частей при создании трехмерной компьютерной игры. Для того чтобы игра работала должным образом, необходимо максимально оптимизировать ее геометрию. Геометрия, также называемая полигональной сеткой, является набором вершин, ребер и граней, которые образуют форму трехмерного объекта. В движке геометрические объекты создаются и редактируются с помощью Geometry Editing.

Геометрия имеет additive(сложение) и subtractive (вычитание) типы. Это означает, что при наложении одного геометрического объекта на другой можно производить сложение или вычитание.² В примере ниже аддитивный тип обозначен, синим цветом, а субтрактивный - красным. Это очень подходит для создания дверей окон и т.д.



² Документация по Unreal Engine 4 | Actor Static Mesh [Электронный ресурс] режим доступа: <http://scriptix.ru/en-US/Engine/Actors/StaticMeshActor> (дата обращения: 14.02.2022).

Иногда приходится создавать более сложные конструкции, формы, и в этом случае используется параметр extrude, который позволяет создавать новые геометрии путем перемещения части грани.



Помимо вышеперечисленных, существует множество других настроек геометрии, которые позволяют создавать еще более сложные геометрические фигуры, например, сжимать геометрию в круги и т.д.

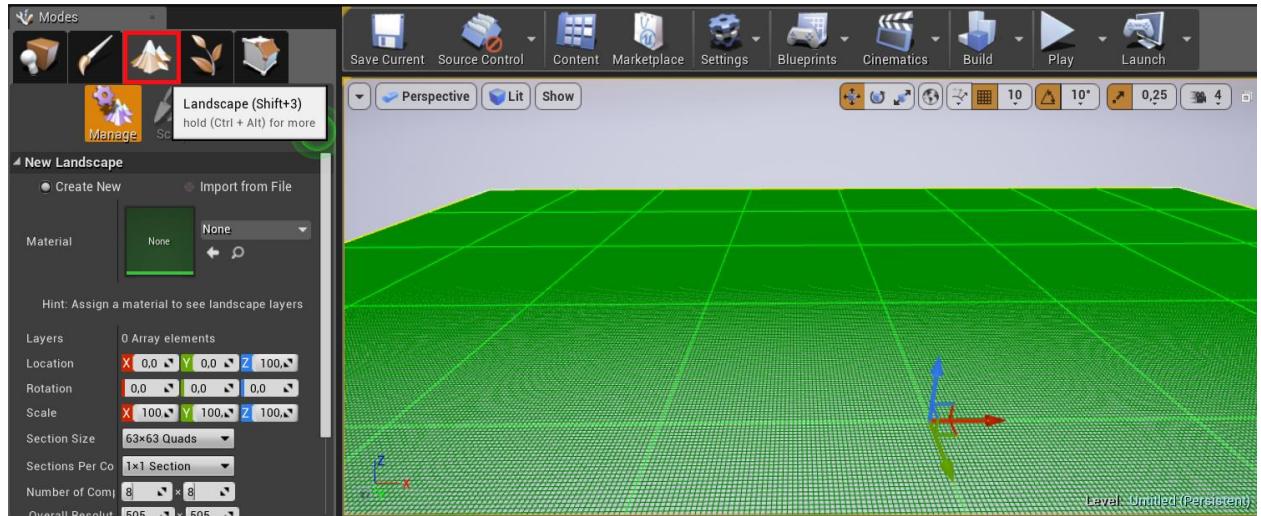
Инструмент Landscape

Среда UE обеспечивает возможность создания огромных ландшафтных миров с помощью набора мощных инструментов для редактирования пейзажей и ландшафтов. Инструмент "Landscape" позволяет создавать захватывающие открытые ландшафты, которые имеют оптимизацию и могут поддерживать приемлемую для игры частоту кадров.

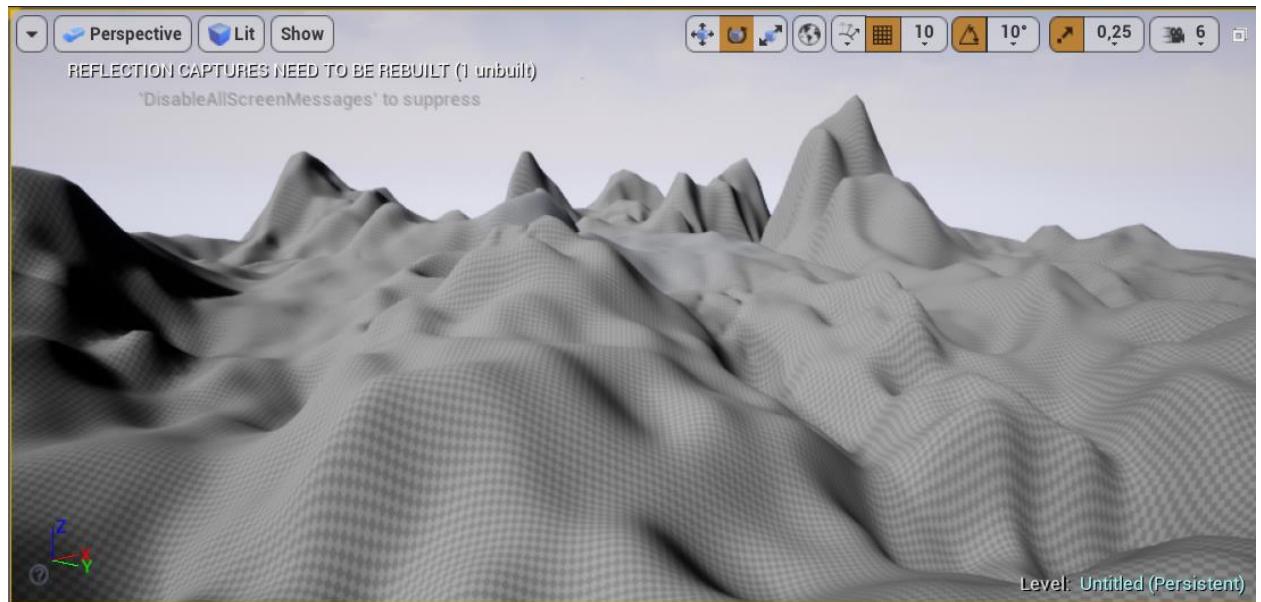
Ландшафты можно создавать с нуля с помощью встроенных в движок инструментов. Кроме того, можно импортировать карты ландшафта, созданные с помощью внешних инструментов.

Ландшафт разделен на участки, и на вкладке section size увеличивается разрешение в участке, т.е. увеличивается детализация. Чем выше разрешение,

тем лучше будет качество ландшафта, но при этом может снизиться производительность. Обеспечение оптимального баланса между красотой и производительностью при создании ландшафта может стать непростой задачей. При создании ландшафта важно учитывать, как будут размещены ландшафты и какие размеры обеспечивают наилучшую производительность.



Существует целый ряд других инструментов для создания реалистичного ландшафта, таких как заглаживание, имитирующее землю после дождя, шум на поверхности, формирование горок, углубление и т.д.



Инструмент Foliage

Foliage (Листья) необходимы для заполнения кучи объектами какой-либо поверхности, к примеру, леса, заполненные травой, деревьями и камнями. Кроме того, это позволяет экономить память, что является

большим преимуществом, чем по одному наброску на уровень, таким образом, он группирует объекты в кластеры, и каждый кластер занимает одну прорисовку. Также в этом инструменте есть множество настроек свойств, таких как минимальное расстояние между объектами, плотность добавляемых сеток, размер кисти, радиус и т.д. Кроме того, каждый объект имеет свои собственные настройки свойств.



2.1.2. Освещения и материалы

Освещение является важнейшим инструментом для оптимизации игрового окружения. Освещение – это мощный инструмент в движке Unreal Engine, который в зависимости от того, как используется свет, может существенно повлиять на производительность игры. Освещение не будет работать, если не происходит рендеринг сцены. Помимо этого, система UE 4 позволяет разработчику использовать четыре различных типа света. У всех этих источников света уникальное поведение и способ использования, и для каждого типа света можно использовать все варианты подвижности:

1. Point light – работает как обычная лампочка, излучая свет во всех направлениях из одной точки.
2. Spot light – направленный свет, он светит как прожектор.

3. Directional light – по сути, это направленное освещение, он действует подобно солнцу, то есть излучает свет от какого-то источника света аналогично солнцу. Он также светит так, как будто охватывает весь игровой мир почти до бесконечности. Как только мы поставим его на уровень и куда бы мы его не перемещали, он будет освещать сцену везде равномерно.
4. Sky light – он также освещает сцену везде одинаково. Светит он из окружающей среды, т.е. принимает информацию, как она светит с некоторого расстояния от сферы.

Свет также бывает статическим (static), динамическим (movable) и стационарным(stationary):

Статичное освещение относится к тем объектам или свету, которые не двигаются с места, но как только в ходе игры мы передвинем объект, то в таком случае тени будут оставаться там, где они были изначально. В связи с этим тени должны быть рассчитаны только один раз во время построения, что гарантирует высокое качество и лучшую производительность.

Динамическое освещение, в отличие от статического, может быть использовано для объектов или света, которые могут перемещаться. Чем больше динамический источник освещает объектов, тем больше будет нагрузка, поскольку в данном типе освещения требуется вычислять тени в каждом фрейме.

Стационарное – освещение это нечто среднее между динамическим и статическим освещением, мы только можем менять свойства света в процессе игры, а объекты, которые он освещает, не должны двигаться, то есть не позволять игроку их менять.

Статический свет полностью запекается в lightmaps и следовательно не имеет потерь в производительности, но и не может меняться в игре. Статический свет может формировать тени для неподвижных объектов, но не для перемещающихся объектов. Это связано с тем, что статический свет не позволяет создавать тени для движущихся объектов. Статический свет

рассчитывается до включения игры, поскольку он не может быть изменен после рендеринга сцены. Если расположение объектов, которые он освещает, или сам статический свет изменится, то сцену придется рендерить заново. Статическое освещение имеет самое быстрое время рендеринга по сравнению с другими вариантами освещения, но при больших масштабах карты оно занимает очень много времени. У статического освещения есть две специальные опции, которые помогают создавать более эффективные тени. Это радиус источника света и разрешение карты света, и эти опции также можно использовать для подвижного и неподвижного освещения. Радиус источника света помогает создавать более мягкие тени. Если освещение имеет нулевой радиус, то будет наблюдаться жесткая тень от объектов, поскольку свет исходит из точки, иными словами, нет других линий, соответственно, тень будет четко разделена на те места, где есть свет и где его нет. Кроме того, свет имеет свойство отражаться от поверхности, что позволяет исключить полную затенённость за объектами. Происходит это благодаря опции *Indirect Light Intensity*, если значение равно нулю, то за кубом будет темная-темная тень, иными словами, свет не будет отражаться (Рисунок 1).

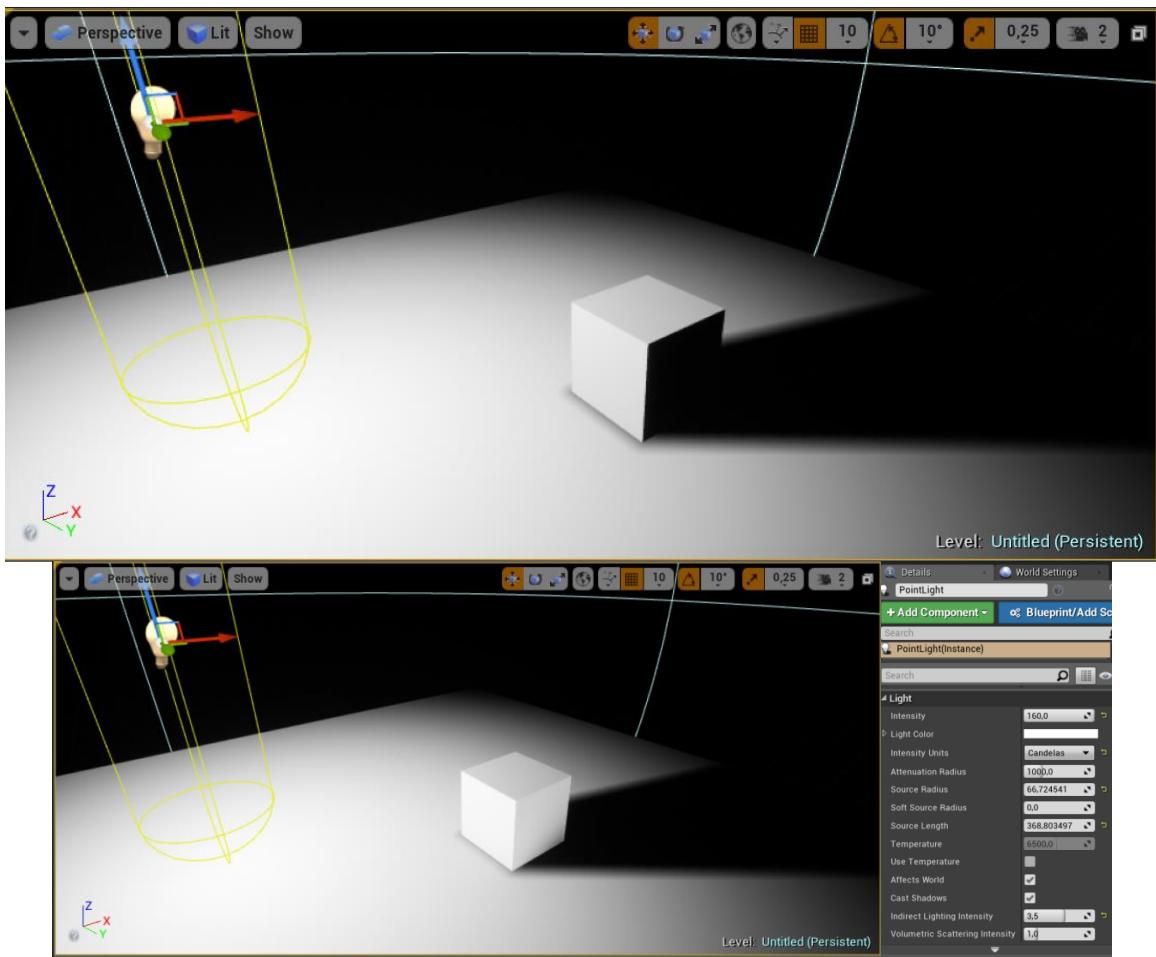


Рисунок 1 — Прямое освещения и Непрямое освещения.

Параметр **Performance**

Чаще всего используется в динамических источниках света, так как сильно влияет на производительность, потому что из-за динамического типа освещения возникает значительные проблемы с производительностью. Таким образом, он отключает свет на определенном расстоянии от камеры или от игрока во время игры, чтобы лишний раз не проделывать вычисления света. Max draw distance — будет указываться максимальное расстояние до лампочки.

Max distance fade range — будет постепенно затухать лампочка, пока полностью не выключится, так же постепенно будет появляться.

При использовании подвижных типов источников света чаще всего требуется регулировка этого параметра с целью повышения производительности.

Lightmap Resolution

Разрешение Lightmap помогает создавать более детализированные тени на поверхности объектов. Если на вытянутом кубе получается плохая тень, то это означает, что мы увеличили его размер, а не разрешение. Существует множество способом повысить разрешения одним из них и наиболее простым является изменение в настройках объекта в разделе Light пункта overridden light map res здесь необходимо установить желаемое значение.

Существует три вида разрешения, которые определяют уровень производительности.

1. Синий цвет — низкое разрешение по сравнению с размером объекта.
2. Зеленый цвет — рекомендуемое / правильное разрешение.
3. Красный цвет — высокое разрешение.

Данный режим используется для определения тех объектов, которые имеют низкие, средние или высокие настройки карты освещенности (Рисунок 2).

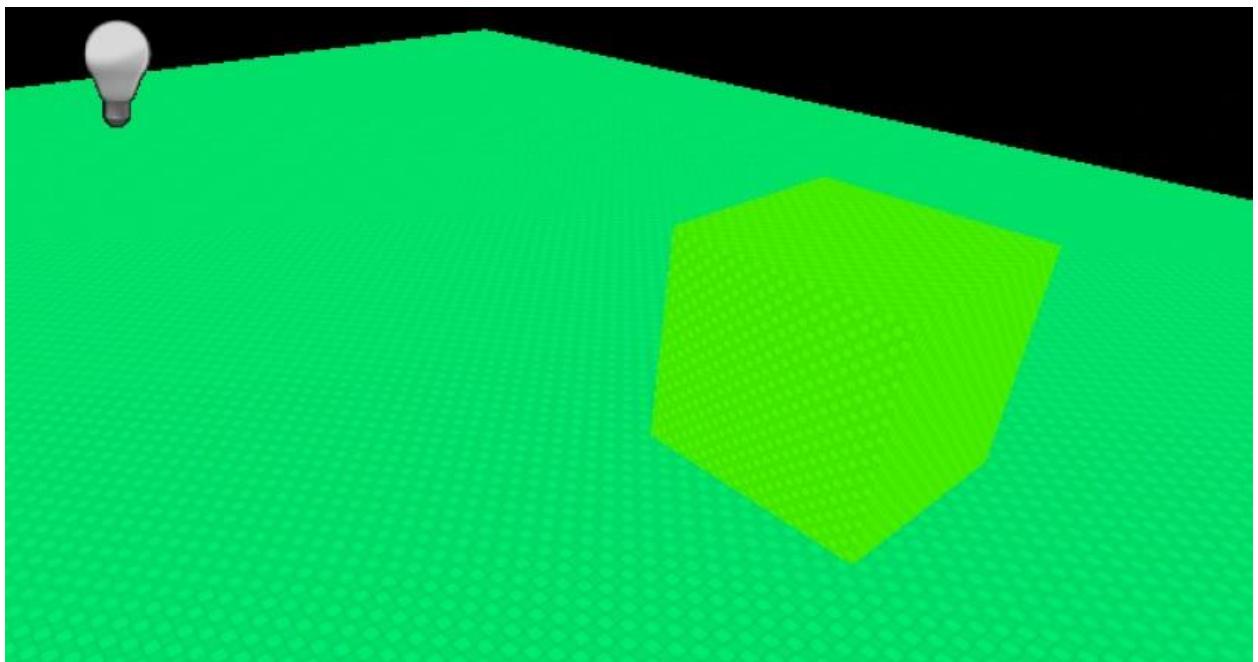


Рисунок 2 — Изменение разрешения карты освещения.

Также существует отличный инструмент, для расчета света называемый Lightmass он позволяет создавать освещение с использованием комплексных световых взаимодействий, к примеру, теневое затенение и

рассеянное обратно пропорциональное отражение. Таким образом, он используется для больших карт, для оптимизации. В значительной степени регулирует область, в которой свет будет просчитываться детально. При этом область, лишь в которой освещение будет лучше рассчитываться, может иметь различные геометрические формы.

Меш – это сетка полигонов, из которых состоит любой 3d-объект. Static mesh, в свою очередь, представляет собой объект, состоящий из полигонов. Причем у этих объектов отсутствует скелет. Статика подразумевает, что подобные объекты не могут иметь анимацию, однако их вполне можно перемещать, изменять их свойства.

Материалы и текстуры

Материалы представляет собой свойство, которое применяется к статическим сеткам для манипулирования внешним видом какого-либо объекта, а также позволяет настроить визуальный вид сцены. Материалы можно применять к различным элементам, например, частицам, элементам пользовательского интерфейса, а не только к статичным сеткам.

Материалы формируются путем добавления узла, это в основном расчет цветов, текстур, кроме того, один из способов создания спрайтов - это материалы, следовательно, материалы отвечают за то, как должна выглядеть поверхность. К примеру, ниже представлен светящийся материал, если его применить к мешу, то данный объект будет сияющим (Рисунок 3).

Текстура представляет собой изображение для материалов, с помощью которого можно покрывать поверхность объектов, на которые применяются материалы. Также текстуры могут быть использованы отдельно от материалов, к примеру, для создания эскиза HUD. С помощью текстур можно окрашивать объект или использовать их в различных вариантах наложения. Как правило, текстуры создаются в программах редакторов, но в некоторых случаях они могут быть сгенерированы и в UE.

Следовательно, материалы и текстуры в зависимости от их качества могут сильно повлиять на сцену уровня. С помощью текстур высокого

разрешения при определенных модификациях можно добиться более детализированного и реалистичного вида сцены.

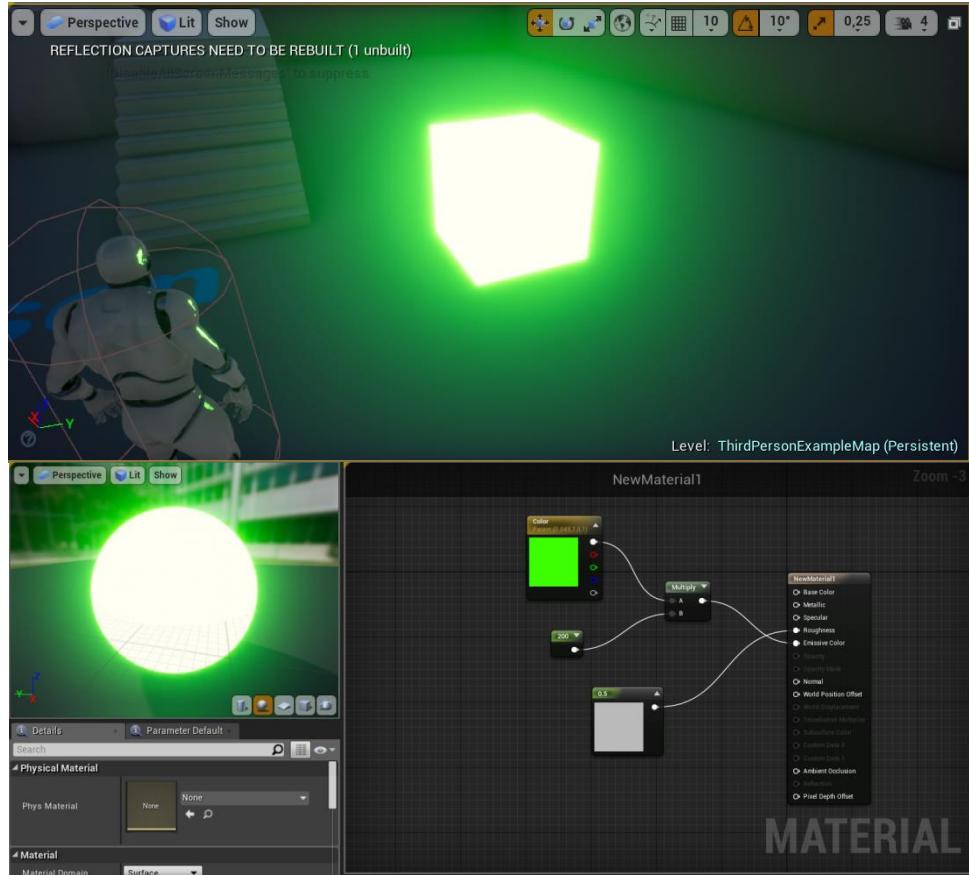


Рисунок 3 — Применение светящегося материала к кубу.

2.1.3. Основы физики и коллизии

Наличие физики в игре повышает уровень восприятия в каждой игре, поскольку будет присутствовать иллюзия того, что игроки взаимодействуют с разными объектами в рамках законов физики, как в реальной жизни.

Большинство игр на платформе Unreal Engine используют программное обеспечение PhysX, которое предоставляется в пакете с движком, что позволяет обойтись без необходимости разработки собственного программного кода для обработки сложных физических эффектов, присущих современным компьютерным играм. Следовательно, это программное обеспечение используется для вычисления физических явлений и столкновений во время игры.

Следовательно, PhysX дает возможность с помощью отличного программного обеспечения обрабатывать такие детальные визуальные

эффекты, как дым, взрыв, физика ткани и воды или листов бумаги, развивающихся на ветру. Система была разработана с целью, сделать физику действительно простой и отличной с точки зрения программного обеспечения.

Для того чтобы объекту было свойственно реагировать на физические явления, следует включить симуляцию физики в его свойствах. Тогда появится возможность наблюдать, как объект взаимодействует с окружающим пространством в соответствии с законами физики, а именно: он может вращаться при толчке, падать, реагировать на столкновения, а также можно регулировать массу объекта, гравитацию и другие параметры (Рисунок 4).

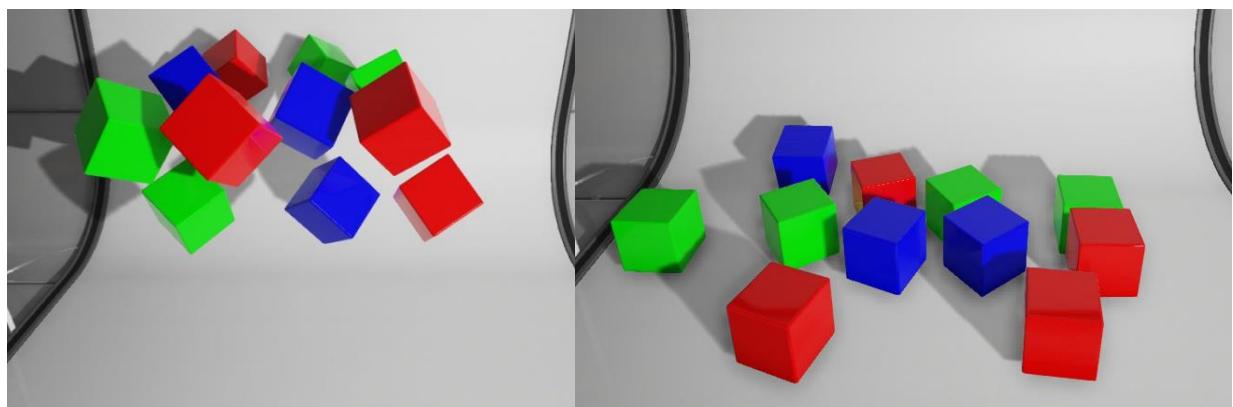


Рисунок 4 — Основы физики.

Коллизия

В играх довольно часто возникает необходимость обрабатывать столкновения между объектами. Ими могут быть NPC, пули из оружия или объекты, с которыми необходимо взаимодействовать и так далее. В параметрах объекта можно заметить collision presets данная функция позволяет определить, как обрабатывать столкновение, здесь представлено достаточно много вариантов обработки столкновения, например, блокировать все, блокировать только пешку и т.д. А также можно создать свой собственный метод обработки столкновений, выбрав тип объекта, зададим свою собственную коллизию по трем видам взаимодействия

1. Ignore – полностью игнорировать то есть мы даже не узнаем пересек границу столкновений или нет.
2. Overlap – объект будет игнорирован и при этом даст нам знать что объект пересек границу столкновений и он нам отправить событию по которой мы можем что ни будь запустить.
3. Block – блокировать.

Соответственно, существует несколько способов создания столкновений: выбрать предустановленное столкновение или же создать пользовательское столкновение. При создании пользовательского столкновения нам необходимо выбрать тип объекта и активировать необходимые блоки для трех типов взаимодействий. Типы объектов для обработки столкновений:

1. WorldStatic – те объекты которые не подвижны
2. WorldDynamic – те объекты которые могут двигаться в процессе игры.
3. Pawn – это какой-либо персонаж.
4. PhysicsBody – это объект с каким-то физическим материалом.
5. Vehical – это какой-либо техника.
6. Destructible – это разрушаемые объекты.

А также столкновение может быть сделано через сетку какого-либо объекта, но в таком случае оно перестает влиять на физические явления. Поскольку устанавливается комплексное коллизия, к большому сожалению, в Unreal если объект имеет комплексное столкновение, то на него перестают действовать законы физики, поэтому комплексное столкновение лучше применить к неподвижному объекту, при этом можно вручную через куб определить столкновение или же использовать автоматическое определение столкновения объектов (Рисунок 5).

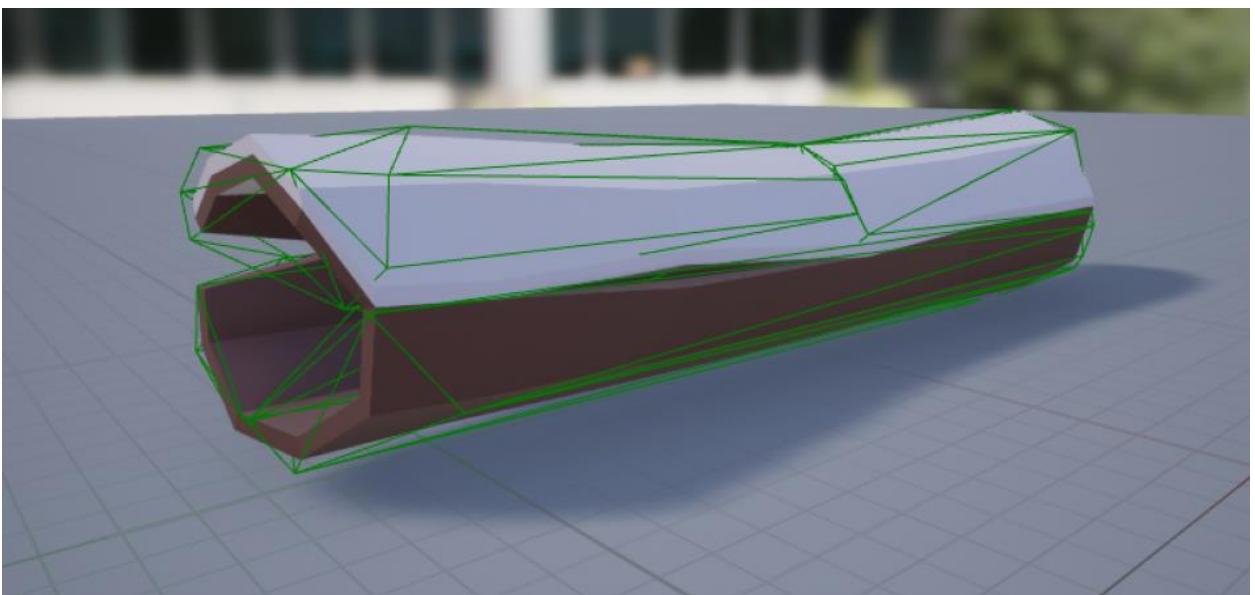
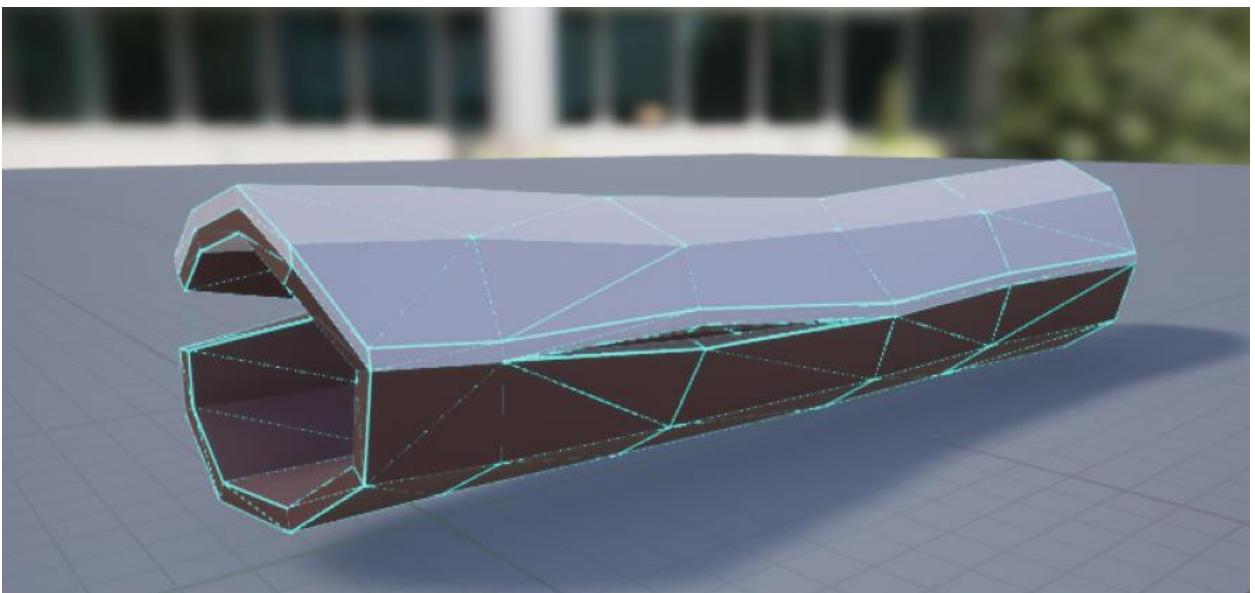


Рисунок 5 — Комплексная и автоматически сгенерированная коллизия.

2.1.4. Particle System

Под визуальными эффектами подразумеваются различные способы создания и манипулирования частицами для придания им большей зрелищности, эмоций или драматизма в играх. Кроме того, в процессе создания игр визуальные эффекты играют ключевую роль, поскольку они придают игре более профессиональный вид, а также делают игровой процесс более интересным и приятным и помогают игроку лучше погрузиться в атмосферу видеоигры.

Во многих видеоиграх визуальные эффекты действительно очень важны, поскольку позволяют добиться огромного улучшения. Они всегда

придавали особый интерес к любой игре с момента создания первых игр, с того времени они превратились уже в некоторую тенденцию в области разработки игр, почти все современные игры не обходятся без спецэффектов, к примеру, дым, туман, огонь из дула оружия, снег, дождь, огонь, жидкости и т.д. Ранее для создания таких эффектов требовались огромные усилия и наличие различных графических инструментов, однако в настоящее время движок Unreal Engine предоставляет интегрированную чрезвычайно мощную и надежную систему частиц под названием Particle System и эти частицы создаются в редакторе Cascade.³ Таким образом, цель визуальных эффектов в играх — это заинтересовать и увлечь игрока.

Система частиц как таковая состоит из эмиттеров, а эмиттеры, в свою очередь, представляют собой структуры, отвечающие за генерацию всех частиц и определяющие их поведение и свойства. Следовательно, они регулируют такие параметры, как продолжительность жизни частицы, положение и направление, в котором генерируется частица, скорость и движение, ее окраску и объем и т.д. Изменив эти параметры, мы тем самым устанавливаем определенное поведение наших частиц, и они будут действовать в соответствии с полученными данными. Например, ниже на рисунке представлено 5 эмиттеров и каждый из них имеет свои параметры, вышеупомянутые и такие, как столкновение, действие частиц в соответствии с мировыми координатами, применение кривых для мягкого преобразования величин направления, образование частиц, при смерти других частиц. А также было добавлено изменение цвета в течение жизни частицы, параметр задержки после разбросывания частиц во все стороны. Для более реалистичного эффекта салюта было необходимо сформировать хвостовую линию при выстреле с земли, кроме того для взрыва в воздухе

³ Туториал по Unreal Engine: C++ [Электронный ресурс] режим доступа: <https://habr.com/ru/post/348600/> (дата обращения: 03.02.2022).

использовалось два эмиттера, один для самого взрыва, а другой для частиц, которые падают и сталкиваются с землей (Рисунок 6).

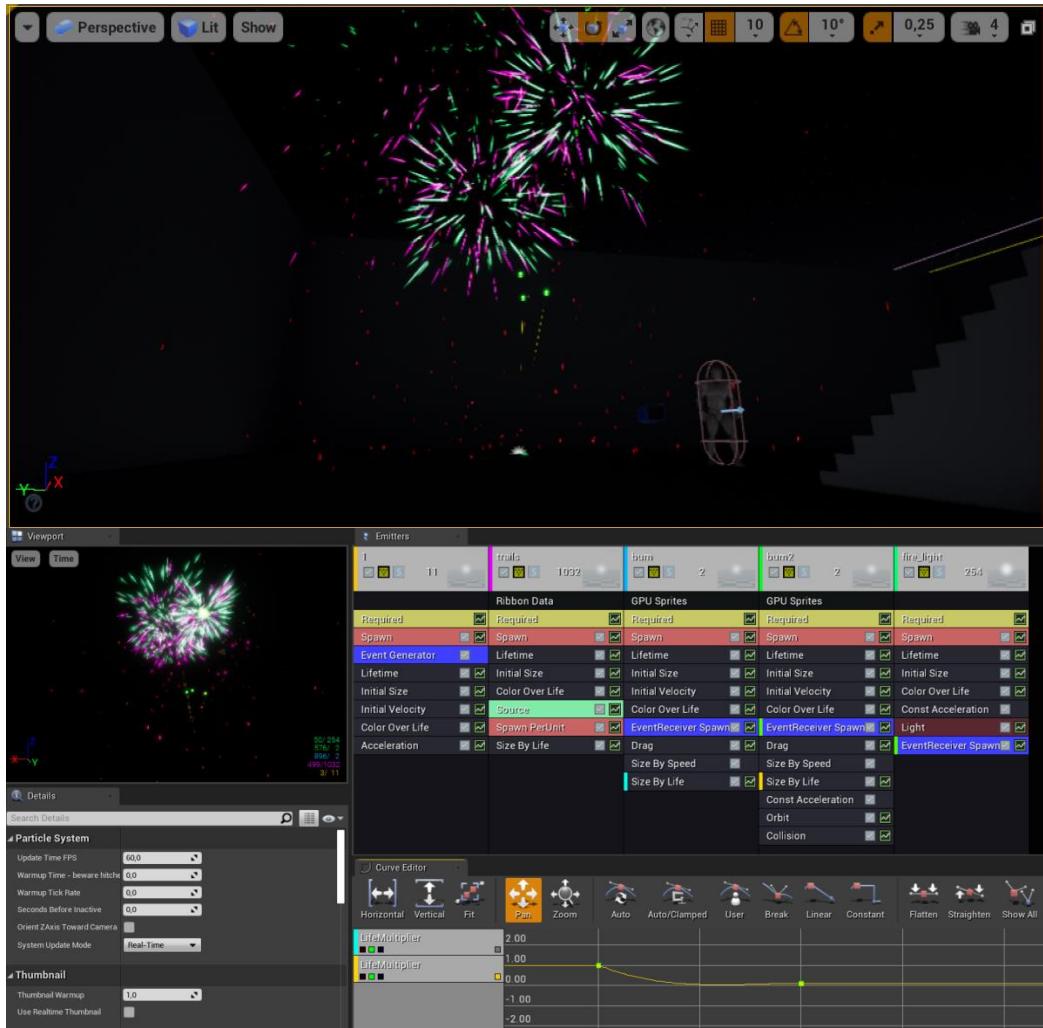


Рисунок 6 — Создание системы частиц на основе эмиттеров.

2.2. Реализация функционала на Blueprint и на C++

В данной исследовательской работе для разработки функционала были использованы Blueprint и C++. Система Blueprint представляет собой весьма гибкую и мощную систему, так как позволяет разработчикам использовать практически все функциональные возможности движка. При этом C++ имеет то отличие, что в нем есть возможность работать за пределами движка, то есть подключение плагинов и работать с сетевыми ресурсами. По сути, работать в Blueprint то же самое, что и в C++, однако реализация логики гораздо удобнее и быстрее в небольших проектах. Кроме того, они могут использоваться и совместно, то есть, к примеру, можно создать функцию или класс в C++ и вызвать его из Blueprint, как это делают в больших проектах.

Потому что обычно, когда они используются вместе, Blueprint используется для редактирования графических компонентов объектов, таких как материалы, текстуры, системы частиц и т.д. А на языке программирования C++ осуществляется функционал самой игры. То есть, из класса C++ можно предоставить нужные функции или переменные в Blueprints, после чего дизайнеры смогут их соответствующим образом модифицировать.

2.2.1. Основы Blueprint

Входным точкам в процессе игры является события event begin play, она позволяет запустить цепочку логику, которая идет за ней, при старте игры. С помощью blueprint можно создать и настраивать все что угодно в движке параметры света, управления, камера, создание персонажа и т.д.

Есть два способа создание логики:

1. Level blueprint. Применяется, когда необходимо управлять объектами, которые находятся на сцене, таким образом, он может взаимодействовать только с классами, которые имеются на сцене.
2. Blueprint Class. Служит для прописывания логики к объектам, например, открывание дверей, кнопок, стрельба и т.д. Таким образом, здесь прописывается логика игры.

По сути event представляет собой событию, по которой определяется начала конкретной логической последовательности, они имеют ноды красного цвета. Она весьма удобна, и применяется, если имеется определенная функция, которая должна быть выполнена только в определенный момент.

Есть также события, которые запускаются при помощи клавиатуры, они задействуют ту или иную функцию при нажатии кнопки, практически во всех играх основные логические потоки запускаются именно через них.

Функции

Как и в языке программирования, она используется для того, чтобы, если необходимо вызвать одну и ту же конструкцию несколько раз в разных

местах, то используемая конструкция помещается в отдельную функцию, и затем в любом месте ее можно вызвать. Кроме того, есть функции от самой архитектуры Unreal Engine, которые выполняют определенную функциональность.

Переменная / Вход и выход.

По сути, игра представляет собой программу с обработкой данных, и поэтому данные где-либо нужно хранить, данными могут быть здоровье персонажа, скорость и т.д. Имеются простые типы данных, такие как целое число, строка/текст, число с плавающей точкой, вектор, так же существует более сложные типы данных, такие как массив значений, структуры, объектный тип и другие.

Ноды имеют вход и выход, по которым они активируют других нод или передают и принимают некоторые данные, и для подключения их тип должен совпадать. То есть целочисленные данные могут подключаться только к целочисленному входу, и они окрашиваются в соответствии с типом данных. Однако есть исключения в некоторых случаях, когда используется преобразователь данных, например, строка в целое число или другие, то в таком случае подключения осуществима.

Контроль последовательности.

Существует несколько нод, которые контролируют ход выполнения, определенного нода. Они определяют порядок выполнения функций, иными словами они перенаправляют управления на множество других нод. Одним из таких блоков является разветвление, так как нода может выпускать только одну ветку выполнения, и она позволяет выполнять любое количество последовательностей по очереди, но из-за того, что она очень быстро выполняется разница не ощутима, к примеру, его можно использовать, когда необходимо запускать несколько функций одновременно по нажатию кнопки. Кроме того, имеются и другие блоки, в частности, branch, то же самое, что и if, switch, циклы и другие. Ниже на примере представлено простая логическая последовательность, в которой события запускается при

нажатии номера 0, и он запускает цикл, который сработает 10 раз и выводить каждый индекс на экран с преобразованием целочисленного типа данных на строку (Рисунок 7).

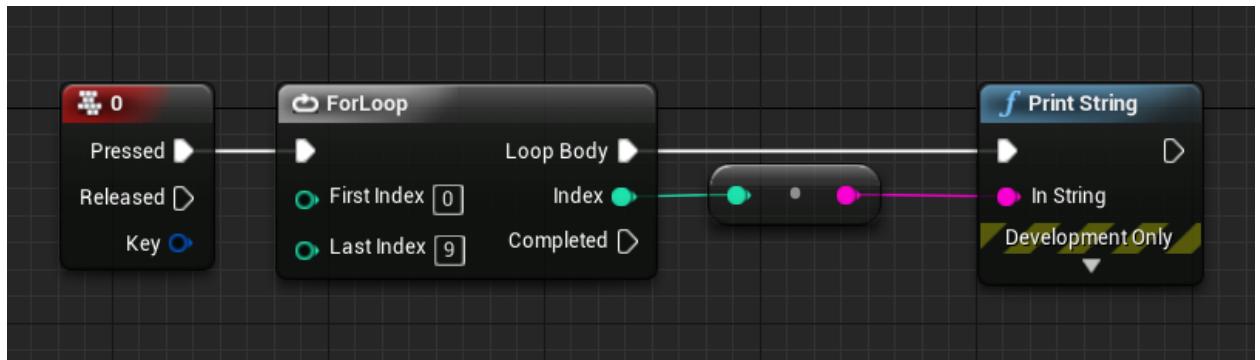


Рисунок 7 — Создание простой логической последовательности.

Классы

Классы создаются с помощью наследования от их родительских классов, наиболее базовым и часто используемым классом в UE является класс Actor. Поэтому все объекты, которые могут быть размещены в сцене, являются актерами, иными словами, актером может быть, например, персонаж, источник света и т. д. Таким образом, класс Actor является базовым классом для большинства объектов на сцене. Помимо этого, существует еще множество других классов для контроля над персонажем, создания персонажа и другие.

2.2.2. Создание анимации с использованием Timeline

Timeline ноды являются специальными нодами, которые позволяют быстро анимировать любые объекты на основе времени, их можно вызвать при определенных событиях, векторах или цветах. Для их редактирования достаточно дважды щелкнуть по ноду временной шкалы на вкладке Graph. Они специально разработаны для выполнения несложных задач, таких как открытие дверей, изменение освещения или выполнение других манипуляций с активными актерами в сцене.

Для создания анимации необходимо добавить переменные на временную шкалу, которые могут быть типа float или vector, затем следует добавить точки, которые будут интерполироваться по временной шкале до

указанного времени, кроме того, их можно зациклить, активировав опцию loop. Чтобы прикрепить анимацию к объекту, нужно использовать узлы масштабирования и перемещения в пространстве, указав им объект, который нужно анимировать (Рисунок 8).

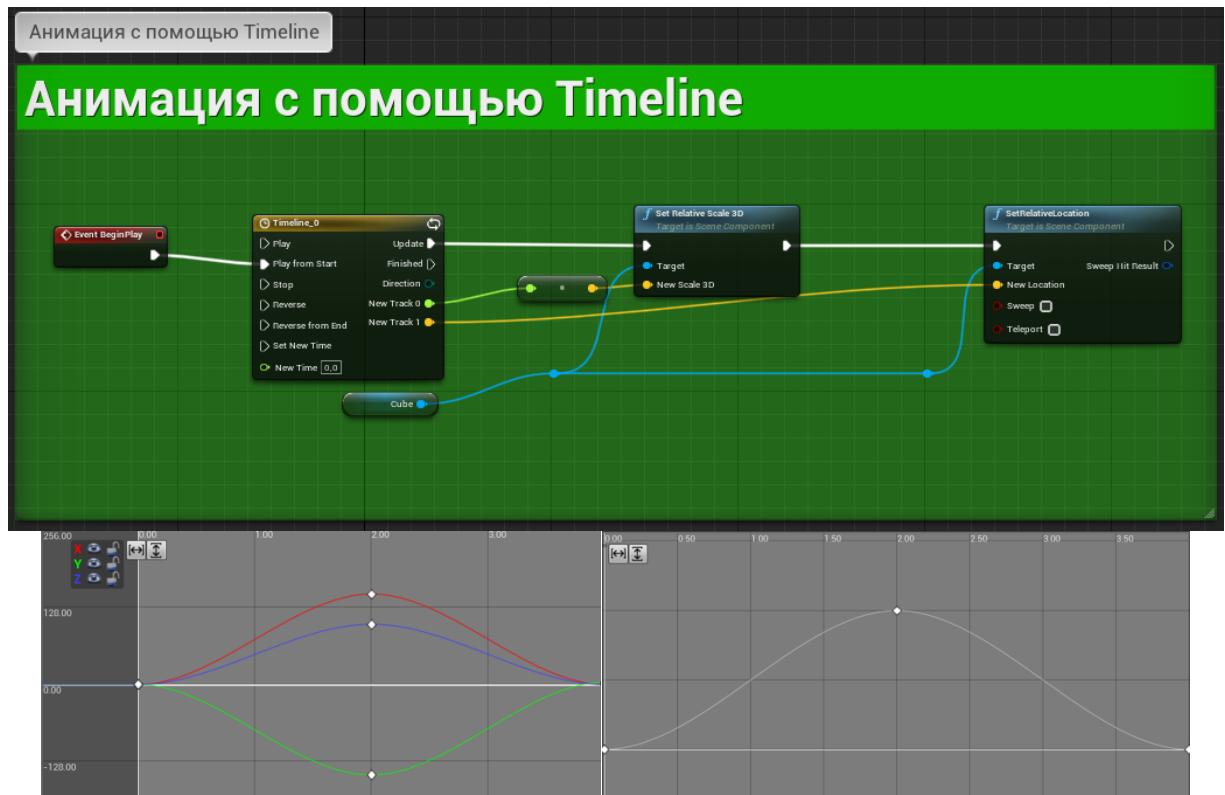


Рисунок 8 — Создание анимации с использованием Timeline.

2.2.3. Основные классы и функции C++

Для создания функциональной части и объектов для игрового процесса, можно использовать язык программирования C++. В данной работе было использована среда разработки Microsoft Visual Studio для написания и компиляции кода. вне зависимости от того, какой язык программирования используется для создания игры с помощью C++ или Blueprint, они взаимодействуют с одной и той же архитектурой движка, поэтому и в случае C++ базовыми классами является класс Actor и класс Object. Любой класс, основанный на Actor, может быть помещен или создан на уровне. Эти классы имеют визуальное представление и поддерживают сетевое взаимодействие. Классы на основе объектов обычно предназначены для хранения данных, и объем их памяти обычно меньше, чем у классов на основе актеров. Кроме

этого все объекты класса Actor помечаются приставкой A, то есть она используется для типов AActor, AController и т.д. Все объекты, которые создаются в памяти или компоненты, которые могут существовать, в объекте они помечаются приставкой U, иными словами она используется для объектов UObject, UActorComponent, USceneComponent. Это связано с тем, что технология Unreal Engine требует, чтобы все классы начинались с определённой буквы.

Таким образом, при создании класса используются два файла, заголовочный и исходный, в заголовочном файле объявляются все классы, переменные, функции и т.д., а в исходном файле выполняется реализация кода.

Макросы

Макрос UCLASS() — это специальный макрос, необходимый Unreal Engine, чтобы сообщить редактору о классе и включить его для сериализации, оптимизации и других функций, связанных с движком. Этот макрос связан с макросом GENERATED_BODY(), который включает в себя дополнительные функции и определения типов в теле класса.

Макрос UFUNCTION() — с помощью этого макроса можно отображать функции в графах Blueprint, он предоставляет разработчикам возможность вызывать или расширять функцию из ресурсов Blueprint без необходимости изменение кода C++.

Макрос UPROPERTY() — представляет собой макрос, используемый для того, чтобы сделать переменные, функции и т.д. доступными в редакторе.

Параметры UPROPERTY().

EditAnywhere — это свойство можно редактировать в Blueprint по умолчанию и экземплярах, размещенных в мире.

EditDefaultsOnly — это свойство можно редактировать только в Blueprint по умолчанию. При размещении экземпляров актера в мире данное свойство нельзя редактировать отдельно для каждого экземпляра.

`EditInstanceOnly` – это свойство можно изменить только для экземпляров, размещенных на уровне. Это свойство недоступно в Blueprint по умолчанию.

`VisibleAnywhere` – представляет собой свойство, которая имеет схожесть с `EditAnywhere`, но свойство нельзя редактировать. Он доступен только для чтения.

`VisibleDefaultsOnly` – данное свойство аналогично `EditDefaultsOnly`, однако его нельзя редактировать. Оно доступно только для чтения.

`VisibleInstanceOnly` – это свойство имеет такую же видимость, что и `EditInstanceOnly`, но его нельзя редактировать. Он доступен только для чтения.

Параметры UFUNCTION().

Exes – функцию можно запустить с внутри игровой консоли. Команды Exes работают только тогда, когда они объявлены в определенных классах.

`SealedEvent` – функция не может быть переопределена в подклассах. Ключевое слово `SealedEvent` следует применять только для событий. Для функций, не относящихся к событиям, необходимо объявить их как `static` или `final`, чтобы запечатать их.

`BlueprintAuthorityOnly` – Данная функция будет выполняться из кода Blueprint, только если он запущен на машине с доступом к сети.

2.2.4. Кодирование логики на C++

Как правило, в системе программирования Blueprint при добавлении компонентов автоматический создается переменная, где будет храниться ссылка на компонент, и этим компонентом может быть меч, оружия или какой-либо объект, таким образом, после создания переменной можно с ним работать. В язык программирования C++ для создания переменных функций или компонентов существует заголовочный файл, в котором необходимо объявить переменные, помечая их специальными макросами, дабы контролировать, что в редакторе будет доступно, а что нет.

При объявлении переменных их тип должен соответствовать значению, в архитектуре движка есть множество типов данных, в случае статического меша это UStaticMeshComponent, как видна ниже на рисунке переменная будет хранить объект по ссылке. Также необходимо создать компонент Scene он требуется для того чтобы создать иерархию, иными словами он нужен чтобы создать несколько компонентов под него, и перемещать этих компонентов относительно этой сцены (Рисунок 9).

```
#pragma once
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MyActor_class.generated.h"

UCLASS()
class MYPROJECT321_API AMyActor_class : public AActor{
    GENERATED_BODY()
public:
    AMyActor_class();
protected:
    virtual void BeginPlay() override;
    UPROPERTY(VisibleDefaultsOnly, BlueprintReadWrite, category = "MyComponents")
    UStaticMeshComponent* MyMeshComponent;
    UPROPERTY(VisibleDefaultsOnly, BlueprintReadWrite, category = "MyComponents")
    USceneComponent* MySceneComponent;
public:
    virtual void Tick(float DeltaTime) override;
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, category = "Myvariables")
    float var;
    UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, category = "Myvariables")
    bool Mybool;
```

Рисунок 9 — Создание компонентов StaticMesh и Scene.

После создания переменных для каждого компонента необходимо их инициализировать. Для этого их нужно создать внутри конструктора. Конструктором считается функция с названием класса, при первом обращении к классу создается конструктор, она позволяет назначить переменным или объектам какие-нибудь стандартные параметры, которые должны быть у объекта. Классы с GENERATED_BODY() получают конструктор по умолчанию.

Для того чтобы создать непосредственно сами компоненты, можно использовать CreateDefaultSubobject. Эта функция позволяет создать объект заданного типа, видимый в Редакторе, иными словами это является значения

полмолчания для типов данных компонент, потому что значения также могут загружаться из данных экземпляра класса для конкретного объекта. Таким образом, необходимо создать компонент каждого типа, адреса которых в памяти будут присвоены передаваемой переменной. После этого необходимо создать иерархию.

Класс Actor имеет несколько типов событий, которые могут вызываться в процессе игры.

Begin Play – Вызывается когда экземпляр этого класса помещается на сцену или сам класс, при старте игры.

Событие Tick будет вызываться в каждом фрейме, поэтому в контексте этого класса можно реализовать разного рода анимацию. Таким образом, необходимо добавить вращение объекта вокруг своей оси, запустив функцию вращения по ссылке (Рисунок 10).

```
#include "MyActor_class.h"
AMyActor_class::AMyActor_class()
{
    PrimaryActorTick.bCanEverTick = true;
    MyMeshComponent = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("MyMesh"));
    MySceneComponent = CreateDefaultSubobject<USceneComponent>(TEXT("MyScene"));
    RootComponent = MySceneComponent;
    MyMeshComponent->SetupAttachment(RootComponent);
}
void AMyActor_class::BeginPlay()
{
    Super::BeginPlay();
}
void AMyActor_class::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    MyMeshComponent->AddLocalRotation(FRotator(0.f, 1.f, 0.f), false);
}
```

Рисунок 10 — Реализация логики в исходном файле.

Расширение класса C++ с помощью Blueprint.

После создания логики в C++ можно создать класс Blueprint на основе этого класса. Как упоминалось выше, чаще всего разработчики расширяют логику C++ кода в Blueprint с целью инициализации компонентов, привязки актеров, необходимых для использования. Таким образом, удается

сэкономить время при назначении актеров по сравнению с поиском и настройкой актера в коде.

Чтобы расширить возможности класса C++ в унаследованном классе необходимо добавить анимацию по Z к нашему мешу, которая будет запускаться при старте игры (Рисунок 11).

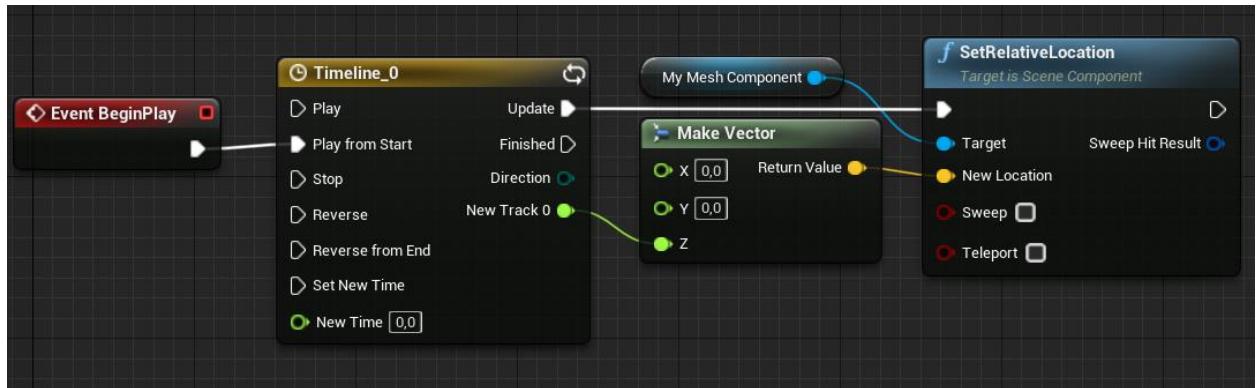


Рисунок 11 — Унаследованный класс от C++.

В результате оружие становится парящим с эффектом анимации колебания при вращении в воздухе, подобно тому, как в крупных играх происходит смена оружия в определенных местах игрового мира (Рисунок 12).



Рисунок 12 — Анимация с использованием C++ и Blueprint.

Глава 3. Программная реализация

3.1. Графическое оформление сцены

Формирование игрового мира начинается с проектирования дизайна сцены, то есть определения того, какая атмосфера должна быть в игре, это включает в себя добавление персонажа, добавление 3D-моделей и других инструментов, необходимых для игры.

Для создания сцены, персонажей и других моделей были использованы объекты из Интернет-ресурсов, таким образом, они размещаются или собираются внутри игрового мира для дальнейшего улучшения мира. 3D объекты могут представлять собой статические объекты, например, деревья, камень, дома, поезд, дорога и т.д. Еще существуют скелетные объекты, которым дается управление анимацией, обычно это персонаж, NPC или какие-либо животные. Другим очень важным аспектом является имитация природных явлений, то есть погода, атмосфера, облака, световые эффекты.

Для размещения некоторых объектов в сцене, таких как трава, камень, кактусы, также были использованы инструменты самой среды Unreal Engine, с целью улучшения производительности игры. Кроме этого были доработаны материалы для объектов, которые в этом нуждались (Рисунок 13).



Рисунок 13 — Проектирование дизайна сцены.

3.2. Применение анимации к персонажам

Анимационные ресурсы для этой игры были получены из торговой площадки Unreal Engine. И требуемые из них для игры были применены на персонажей одним из методов Retarget, этот метод срабатывает в том случае, когда анимации применяемые скелетным персонажам, использует тот же скелет, что и персонаж, для которого изначально была создана анимация.

Поскольку игра была разработана на основе предлагаемого Unreal Engine шаблона Third Person (Рисунок 14), в котором присутствует стандартный персонаж с анимацией для ходьбы и прыжка, которая в свою очередь было переделано. Таким образом, изменение персонажа происходило путем преобразования костей добавляемого персонажа, следовательно, кости персонажа и применяемые анимации с использованием вышеописанного метода функционируют вполне успешно.

После применения анимации к персонажу, их необходимо включить в игру для использования во время игры. Для этого были использованы такие компоненты как Animation Blueprint и Blend Space.

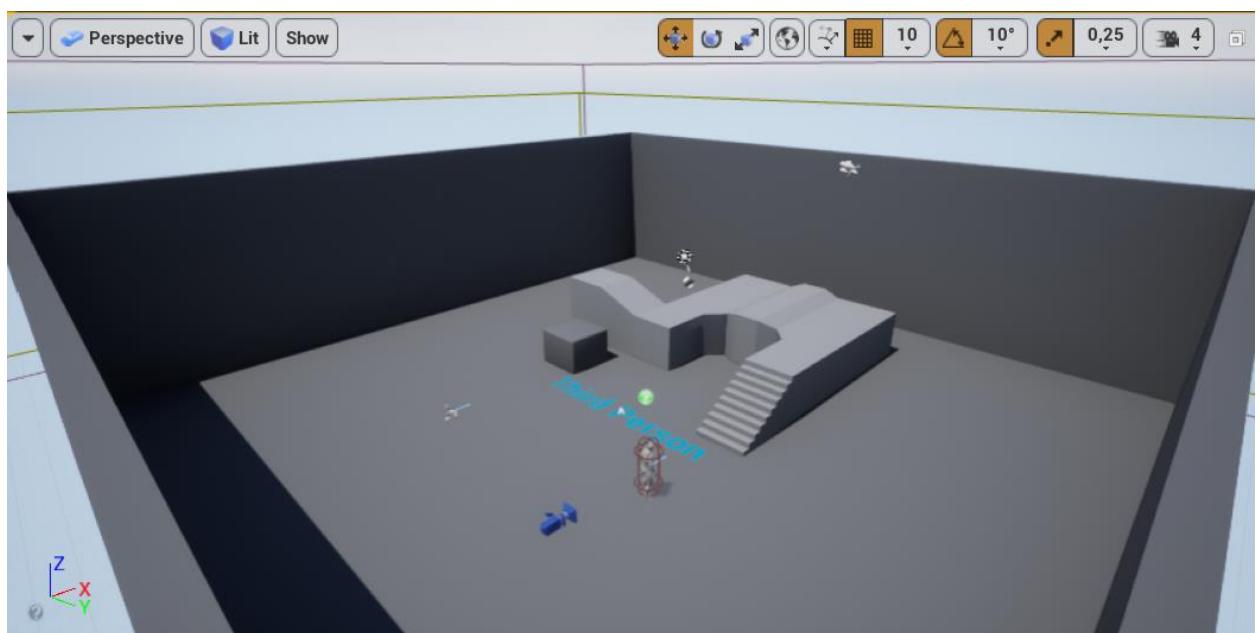


Рисунок 14 — Шаблон Third Person.

Movement component

Этот компонент, как следует из названия, обеспечивает движение. В играх объект перемещается, с простым столкновением по игровому миру.

Область столкновения будет представлять собой что-то вроде капсулы, сферы, куба и так далее. Компонент движения отвечает за скорость, направления движения, которые применяются к движению игрока.

В среде UE анимация обрабатывается с помощью Animation Blueprint. Она предоставляет графический способ создания потока анимации. Она содержит в себе Anim Graph и Event Graph.

1. Anim Graph – используется для смешивания или перехода от одной состояния анимации к другой.
2. Event Graph – она используется для написание логики по которой будет меняться состояния анимации персонажей.

В Animation Blueprint можно создать конечный автомат, который будет управлять состоянием анимации. Для перехода от одного состояния в другой используются определенные условия, которые в свою очередь контролируются из Event Graph. В конечном автомате существует Idle/Run, который представляет собой состояние по умолчанию, а другие состояния будут выполняться на основе условий (Рисунок 15).



Рисунок 15 — Конечный автомат для анимации прыжка.

Для того чтобы игрок мог двигаться вперед-назад влево-вправо, следует переделать стандартную анимацию, поскольку в ней присутствует только анимация ходьбы вперед. Таким образом, для анимации передвижения необходимо рассчитать направление и скорость движения,

чтобы игрок мог ходить и бегать, для этого необходимо создать переменные, которые будут хранить координаты направления и скорость движения.

Для определения скорости нужно получить ссылку на персонажа и узнать скорость движения, скорость задается с помощью movement component, как правило, стандартная скорость устанавливается для того, чтобы, игрок мог ходить, а в случае необходимости этот скорость может увеличиться для анимации бега. Затем эту скорость необходимо записать в переменную, которая в свою очередь потом будет использован для смешивания анимации.

Также требуется переменная Direction, она нужно, чтобы определить, куда идет персонаж. Из персонажа по ссылке необходимо достать две параметры Get Velocity и GetActorRotation с помощью них можно рассчитать направления передвижения, используя Calculate Direction, после расчёта следует результат записать в переменную (Рисунок 16).

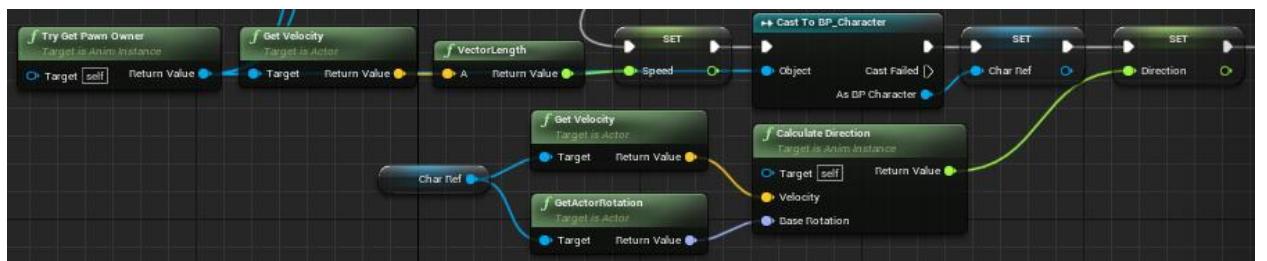


Рисунок 16 — Определение направления движения.

Использование анимации приседания происходит так же, как описано выше, то есть мы создаем переменную типа bool, которая будет определять, происходит ли приседание в определенный момент времени. Таким образом, когда мы вызовем crouch в нашем персонаже, Anim Blueprint поймет это и запустить анимацию приседания (Рисунок 17).

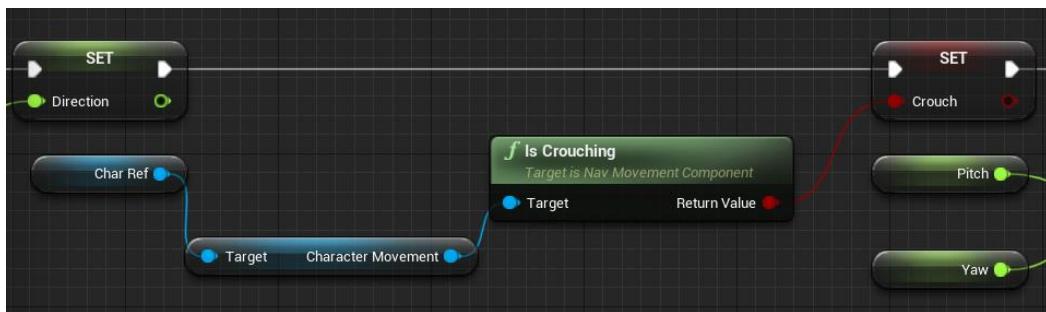


Рисунок 17 — Проверка для приседания персонажа.

Кроме конечного автомата для воспроизведения анимации используется Blend Space, она позволяет смешивать анимации на основе определенных условий, то есть необходимо указать входные данные которые будут использоваться для смешивания анимации, в нашем случае это переменные, которые указывают направление и скорость движения. Анимации (белые ромбы ниже) будут настроены для смешивания соответствующим образом на основе значений этих входных данных, что приведет к финальной позе (зеленый ромб ниже). Для перемещения было создано 3 шкалы скорости – 0, 300, 600. То есть при 0 стоит, при 300 ходит, при 600 бегает (Рисунок 18).



Рисунок 18 — Использование Blend Space для создания анимации.

Для приседания использовался две шкалы скорости от 0 до 100. Также для этой BS было использовано вышеописанные переменные, для того чтобы игрок мог сидя передвигаться (Рисунок 19).

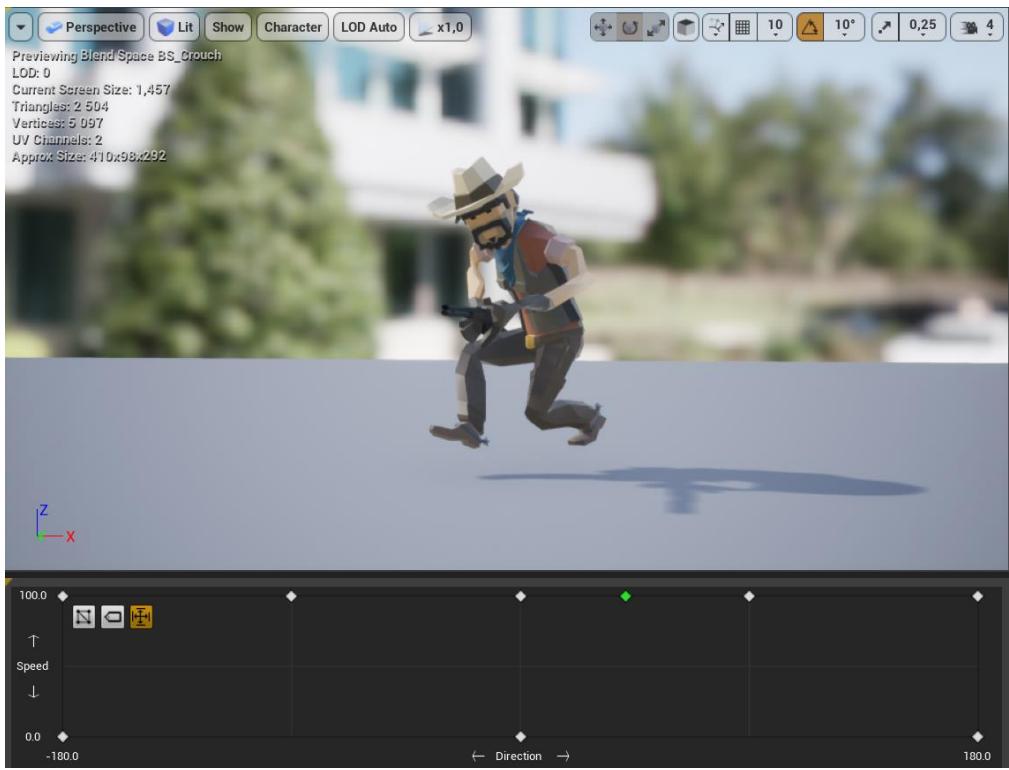


Рисунок 19 — Реализация анимации приседаний.

Ниже на рисунке представлено переменные, которые используются для смешивания анимации (Рисунок 20).

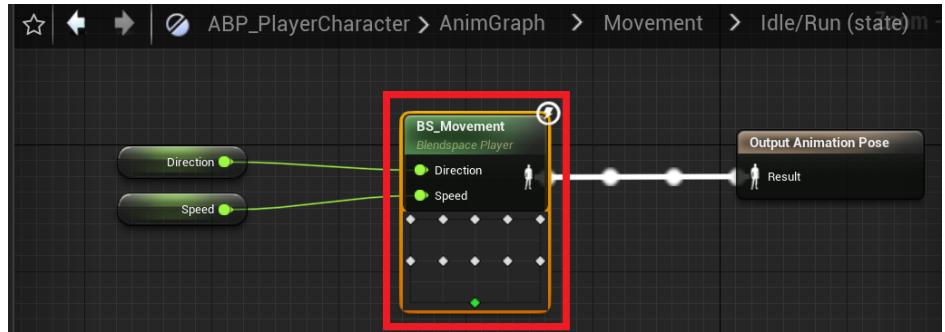


Рисунок 20 — Переменные для смешивания анимации.

Запуск этих анимаций происходит в Blueprint персонажа, здесь нужно определить события, по которым будут запускаться те или иные анимации, ниже на рисунке видно, что анимации запускаются по нажатию кнопки, иными словами, бег будет запускаться, когда игрок нажмет Shift, в этот момент происходит плавное изменение скорости от 300 до 600. А при отжатой кнопке скорость вернется к стандартному значению.

Приседание управляется двумя специальными функциями, которые при нажатии активируют приседание, при отжатой кнопке деактивируют

приседание, затем в Anim Blueprint, как показано на рисунке выше, проверяем наличие приседания, если оно true, то анимация будет воспроизводиться, при условии, что character movement имеет включенное приседание (Рисунок 21).

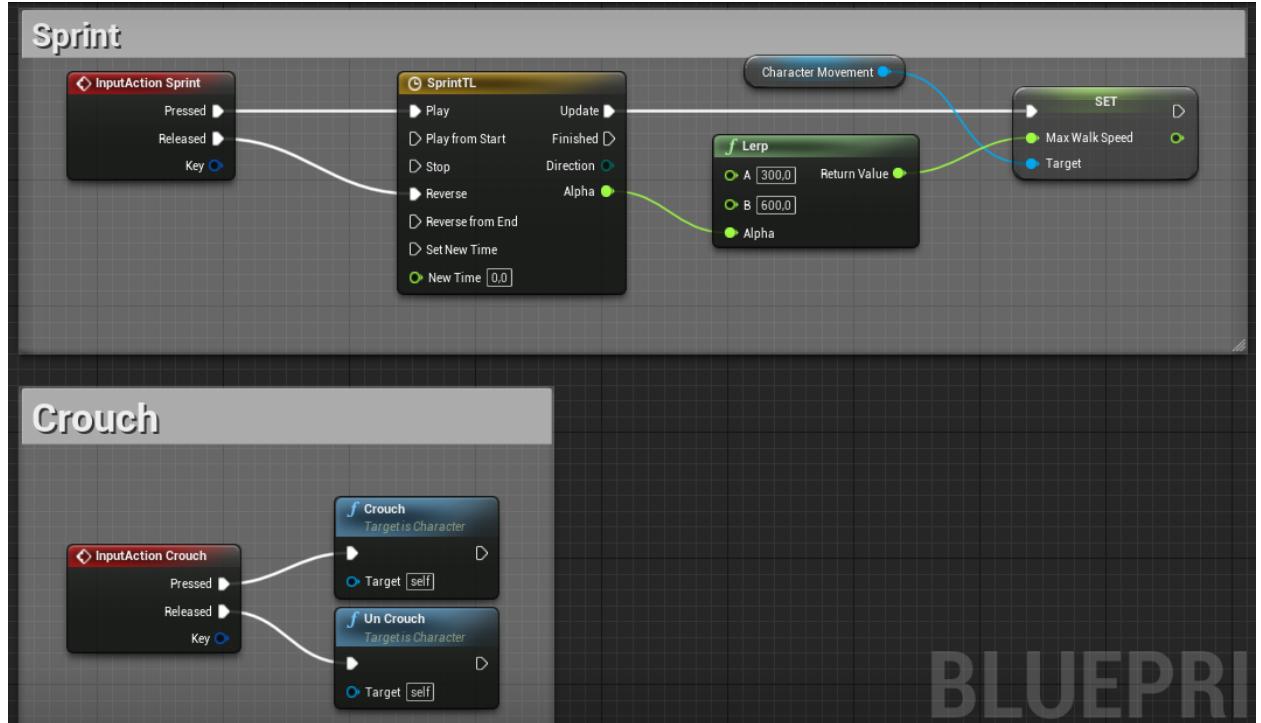


Рисунок 22 —Запуск анимации с помощью событий.

Чтобы смешивать двух машин состояний, которые содержать анимации, использовался Blend Poses by bool которая выполняет смешивания двух поз на основе времени, используя логическое значение в качестве ключа. То есть когда будем вызывать crouch, переменная crouch будет true и будет использоваться анимация приседания, причем анимацией по умолчанию является Movement. Кроме того, можно использовать BS без машины состояний, в таком случае AimOffset запускается без необходимости использования условий. Он позволяет смотреть вверх-вниз влево вправо (Рисунок 23).

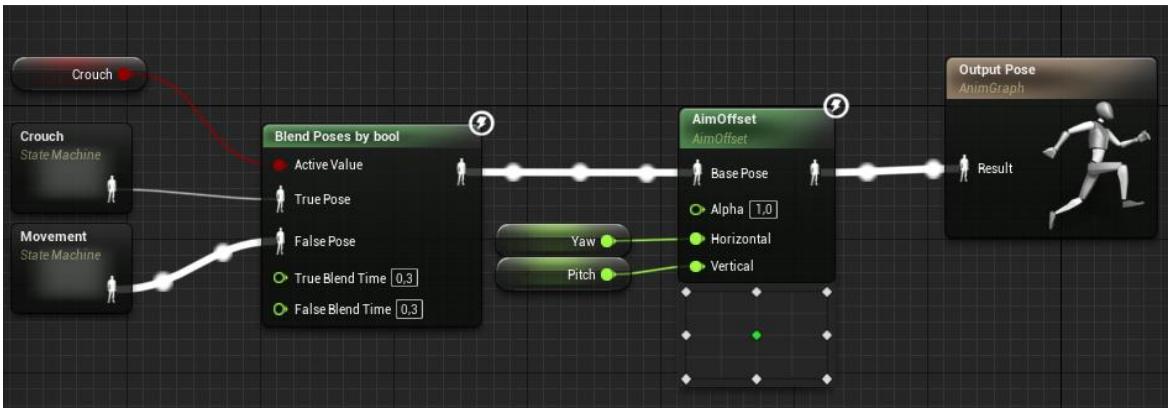


Рисунок 24 — Переключение между анимациями в Anim Graph.

Для открытия и закрытия двери по нажатию кнопки, как описано выше, целесообразно использовать Timeline, с помощью которого будет происходить плавное изменение координаты двери на 90 градусов. Кроме этого было использована триггер с текстом, в результате, чего игрок узнает, что он находится в зоне для открытия и закрытия двери. За пределами этого триггера логика управления дверью не будет выполняться, благодаря отключению управления для этого класса за пределами триггера (Рисунок 25).

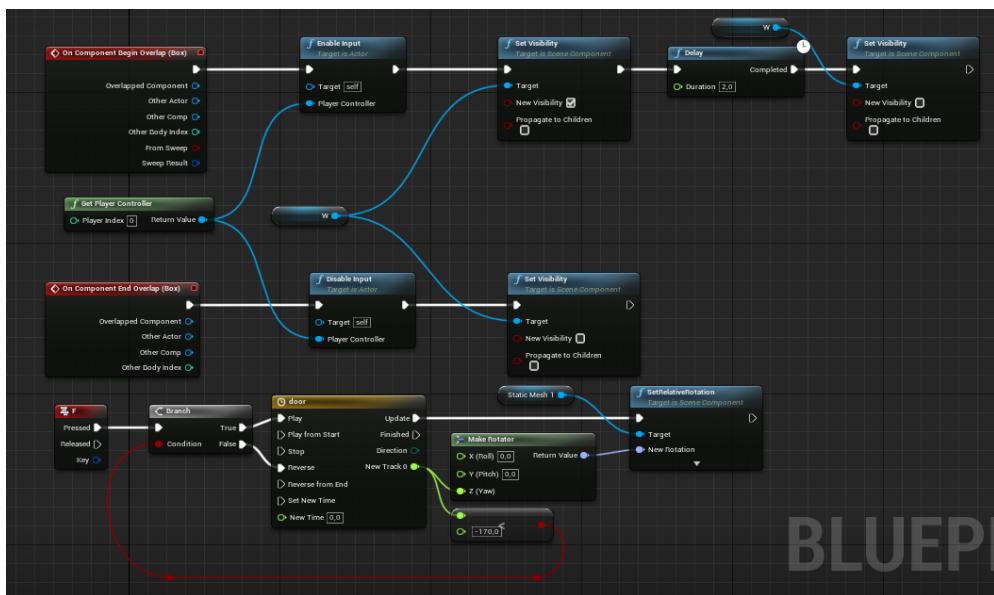


Рисунок 25 — Открытие и закрытие двери по нажатию кнопки.

3.3. Система вооружения

Перестрелки в данной игре являются важной частью разработки, следовательно, после создания дизайна сцены и анимации, необходимо реализовать вооружение наших персонажей. Для этого сначала нам необходимо создать класс оружия, и добавить в него компонент скелета оружия. А также в скелете нашего персонажа необходимо создать точку крепление для оружия, то есть сокет с оружием, и таким образом в классе нашего персонажа собственно реализуем функцию крепление оружие, которая будет запускаться при старте игры.

Кроме того, чтобы оружие появилось у игрока там, где он находится, получим его местоположение, а затем создадим экземпляр класса оружия с помощью `SpawnActorClass`. После создания экземпляра нам нужно записать наше оружие в переменную и прикрепить его к персонажу с помощью `AttachActorToComponent` (Рисунок 26).

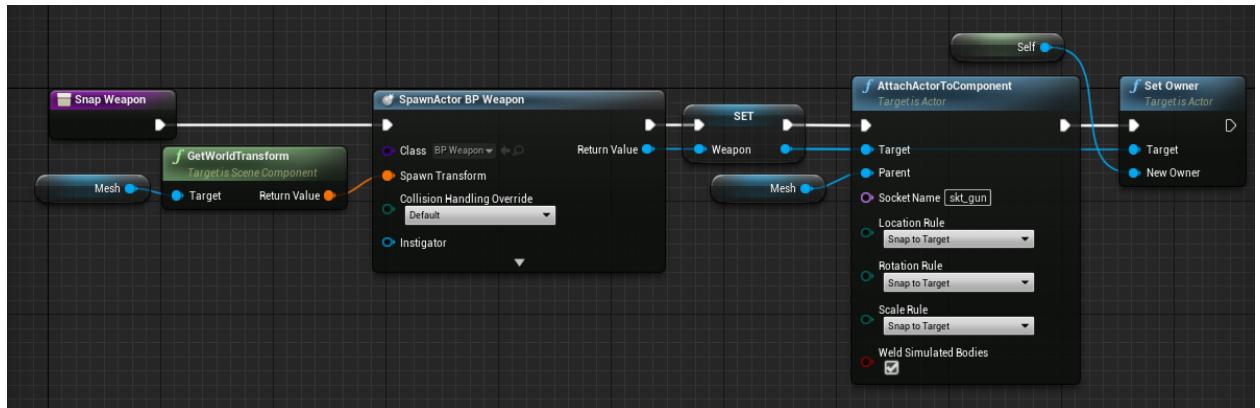


Рисунок 26 — Прикрепление оружие к персонажу.

Стрельба из оружия

После прикрепления оружия к персонажу необходимо было реализовать логику стрельбы из оружия, для этого в классе оружия было реализовано пользовательское событие, которое запускает стрельбу при нажатии игроком ЛКМ, после чего начинается стрельба. Соответственно, в классе оружия событие имеет параметр `boolean`, и его необходимо проверить. Если игрок нажмет левую кнопку мыши в определенный момент, параметр

вернет true, и функция стрельбы будет запускаться каждые 0,1 секунды, пока условие не вернет false, то есть если игрок решит не стрелять, стрельба прекратится (Рисунок 27).

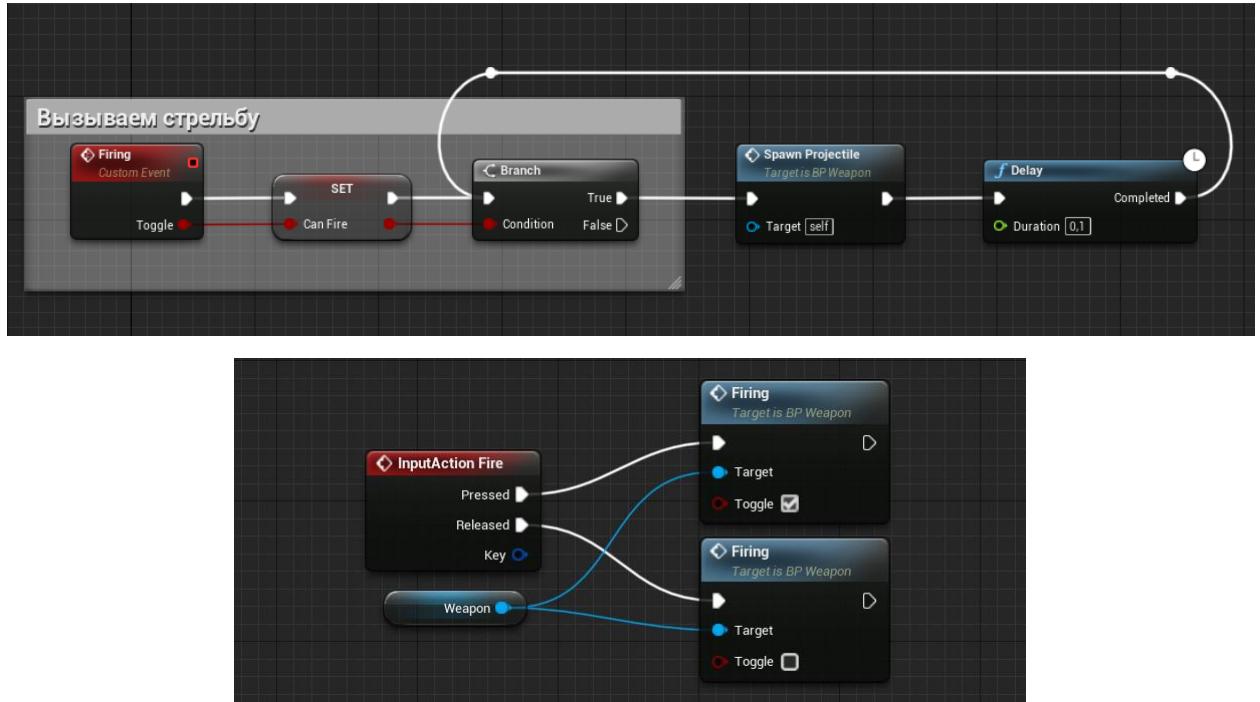


Рисунок 27 — Стрельба из оружия.

Функция для стрельбы

Эта функция будет порождать пули из нашего оружия, чтобы точно указать, из какой части будут выпускаться эти пули, и в какую направлению, мы используем Get Socket Transform, в ней нам нужно указать имя сокета, из которого будут порождаться пули, другими словами, с какой части оружия будет производиться стрельба.

Get Socket Transform – позволяет нам получить положение сокета в меше относительно самого меша. Это означает, что когда мы будем поворачивать поднимать и опускать оружие, выстрел будет происходить оттуда, откуда мы указали.

Эффекты для стрельбы

Данная игра должна включать в себя спецэффекты, как и любая другая хорошая игра. Эффекты и звуки были получены с интернет ресурса. Таким образом, после определения мета порождения пули нам нужно создать эффекты вспышки и звука из дула нашего оружия. Для этого нам нужно

прикрепить эффекты и звуки к skeletal mesh оружия, указав сокет от дула (Рисунок 28).

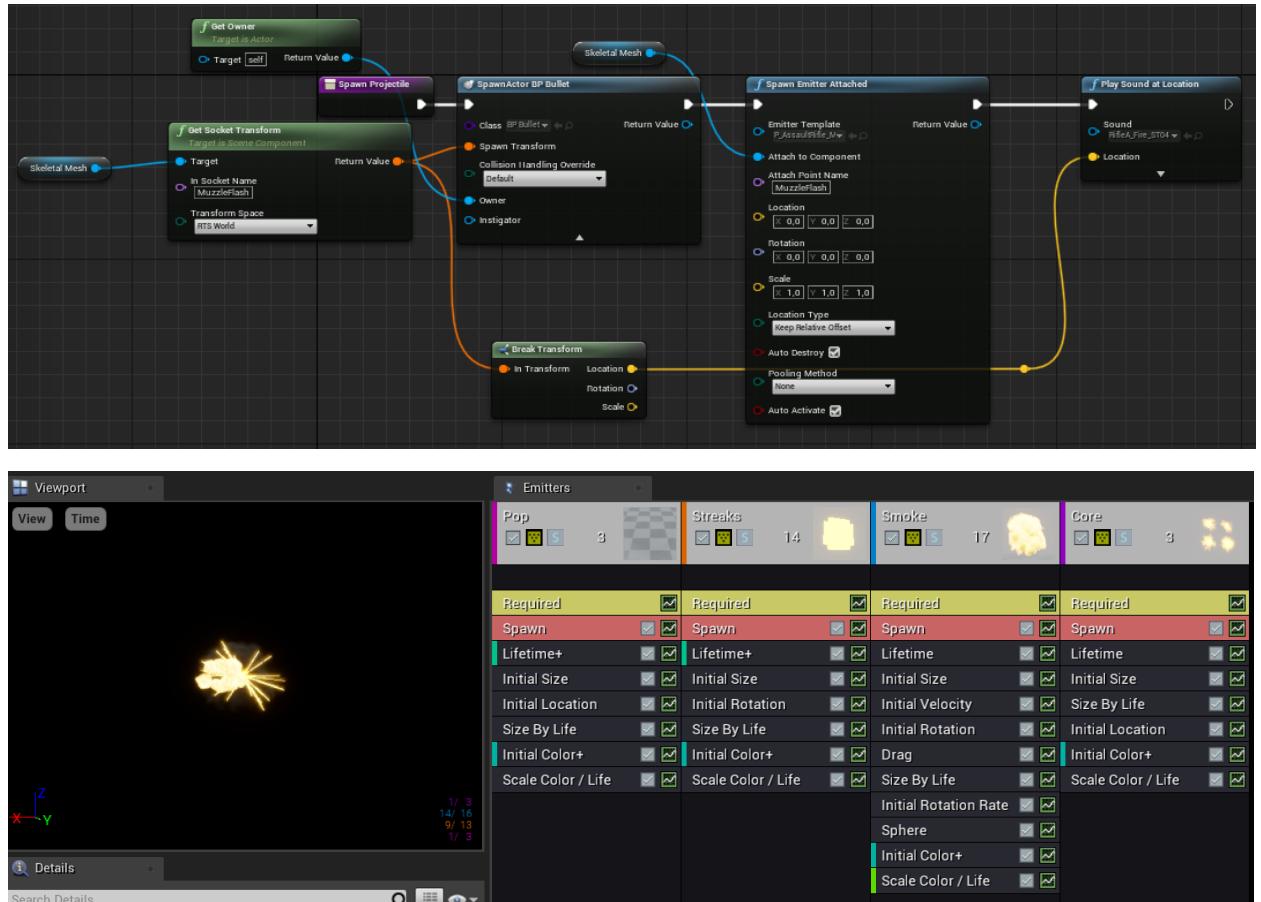


Рисунок 28 — Создание функций и эффектов для стрельбы.

Пуля для оружия

В качестве пули была использована система частиц, которая в свою очередь использовала статическую меш с материалом красно-желтого оттенка, как видно на рисунке ниже. Что же касается функциональности класса, то для удара и столкновения пули необходимо было использовать событие hit, как следует из названия, оно срабатывает при столкновении нашей пули. Поэтому для сферы пули столкновение должно быть включено. Кроме того, при столкновении пули с любой поверхностью мы также создадим эффект удара, а после удара нам нужно будет уничтожить пули.

В процессе создания пули, помимо столкновений и эффектов, необходимо было установить скорость выстрела, чтобы пули летели с определенной скоростью. Чтобы это осуществить, используется Projectile

Movement, оно позволяет снарядам лететь с большой скоростью, отскакивать при необходимости, а также можно задать гравитацию (Рисунок 29).

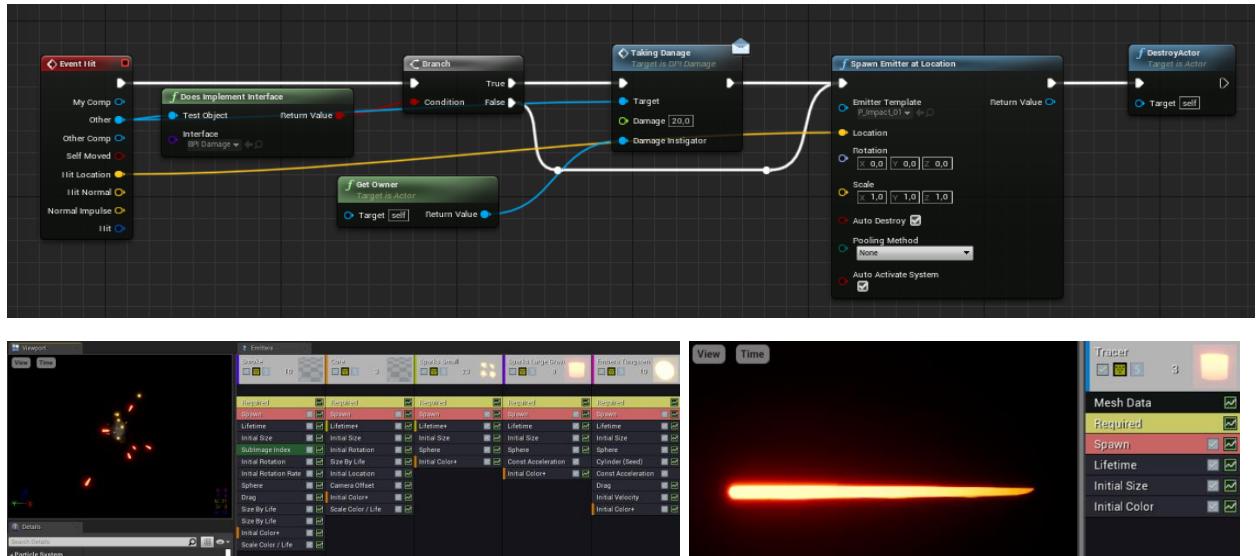


Рисунок 29 — Реализация пули из ружья.

В результате мы получаем реалистичный выстрел с эффектами вспышки и звуком выстрела (Рисунок 30).



Рисунок 30 — Реалистичный выстрел со спецэффектами.

3.4. Создание искусственного интеллекта бота NPC

Искусственный интеллект в играх создает иллюзию человеческого интеллекта, что позволяет игрокам более сильно погрузиться в игру.

Искусственный интеллект — это широкий набор алгоритмов, которые управляют поведением NPC (неуправляемый игровой персонаж). При этом стоит учитывать, что игровые искусственные интеллекты не способны

мыслить, и их поведение предопределено разработчиками. В современных играх используют множества вариантов и подходов создания ИИ, но общий принцип поведение искусственного интеллекта не меняется, то есть: получение информации, обработка этой информации, и действия.

Нейронные сети для управления поведением NPC в последние годы стремительно развиваются, но их необходимо долго обучать перед выпуском игры, иначе ИИ может работать некорректно. К тому же игровые уровни будут меньше контролироваться создателями, что приведет к тому, что любые тщательно продуманные миры могут быть пропущены, а сам искусственный интеллект может вести себя странно, разрушая погружение. Следовательно, их нужно обучать, а поведение этих искусственных интеллектов также будет предсказуемым, например, в большинстве крупных играх при прохождении миссии искусственный интеллект ведет себя так, как предопределили разработчики, тем самым продвигая сюжетные линии видеоигры. Однако многие разработчики видеоигр не решаются встраивать в свои игры продвинутый ИИ, опасаясь потерять контроль над игровым процессом в целом. Таким образом, главная задача ИИ в играх состоит в том, чтобы создать иллюзию поведения настоящего игрока.

Что касается адаптивного искусственного интеллекта, действия которого непредсказуемы, по сути, он также следует правилам, предопределенным разработчиком. Такой искусственный интеллект может адаптироваться, ориентируясь на основе поведения игрока, к примеру, сколько раз игрок убил противников в ночное время, как незаметно прошел базу, в результате чего противники начнут носить приборы ночного видения, устанавливать больше противников в зонах проникновения. То есть процесс адаптация происходит отслеживанием прошлых решений игрока и оценить их успешности. Адаптивный ИИ — это, по сути, ИИ, у которого есть множество различных стратегий, и в зависимости от успеха или неудачи, он мог бы выбрать другую стратегию. Таким образом, создается иллюзия неопределенности на основе алгоритмов предопределенным разработчикам.

Игровой персонаж состоит из многих компонентов. В основном, персонажи различаются по двум основным категориям: Игрок и ИИ. ИИ и игрок отличаются только тем, что игрок получает данные от аппаратуры, тогда как ИИ определяется решениями, которые принимает его "мозг".

Для создания ИИ потребовался специальный персонаж, которым будет управлять ИИ. Персонажи для ИИ создаются различными способами, в данной работе создание такого персонажа осуществлялось путем дублирования класса персонажа и анимации персонажа. Кроме того, чтобы боты отличались от игрока, было решено изменить самого персонажа NPC. Таким образом, был создан новый класс персонажа для ИИ, используя существующий, персонаж игрока в качестве отправной точки (Рисунок 31).

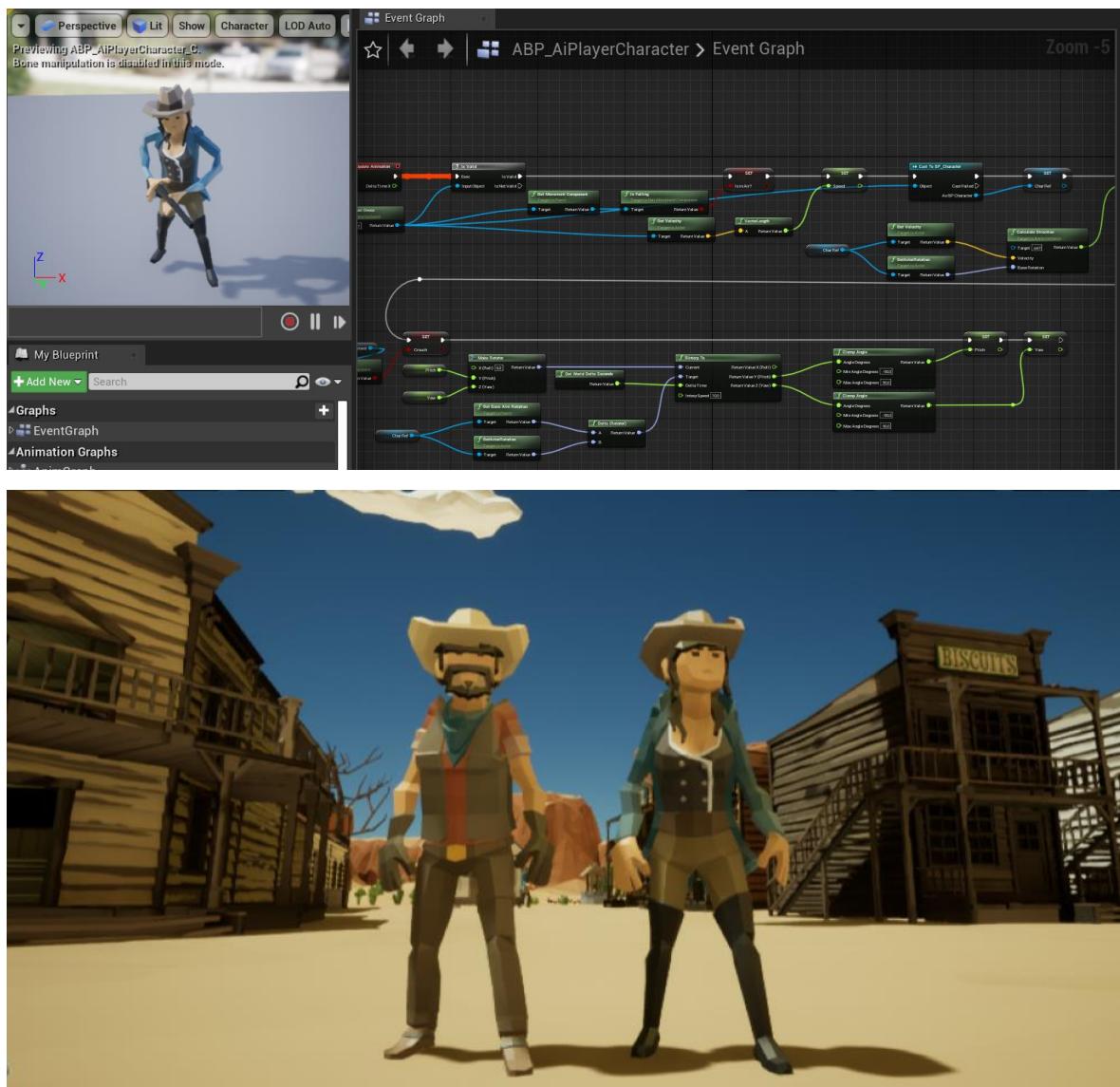


Рисунок 31 — Создание персонажа NPC.

В данной игре для управления NPC, было использовано принцип поиск пути, анализ данных, действие.

В Unreal Engine встроен надежный поиск пути и движение ИИ. Используя навигационную сетку и персонажа, можно создать простой или продвинутый ИИ, взаимодействующий с миром и игроком. На самом деле, он генерирует область, по которой наш ИИ может перемещаться.

Навигация в окружающей среде является фундаментальной задачей для ИИ, но это не так просто, как перемещение из точки А в точку В. Существуют две основные части навигации: поиск пути и предотвращение препятствий. По сути, она представляет собой двухмерную сетку пространства перемещения, построенную с использованием данных о столкновениях из окружающей среды и измерений ИИ. Это означает, что поиск пути и обход дешев, быстр и надежен (Рисунок 32).



Рисунок 32 — Использование навигационной сетки для продвижения ИИ.

Продублировав персонажа, необходимо удалить все, что связанное с логикой управления, в том числе камеру, потому что ИИ не нуждается пользовательскому вводу, таким образом, нужно оставить только функцию привязки оружию к персонажу.

ИИ будет искать новые точки в пространстве в случайном порядке, пока условие возвращает true, другими словами, условие получает две

переменные – здоровье бота и здоровье игрока. Таким образом, когда игрок наносит урон боту, его здоровье будет меньше нашего, следовательно, бот будет атаковать игрока. Более подробно система атаки будет описана в подразделе "Боевая система".

Перемещение бота осуществляется с помощью AI Move To, как следует из названия, ему задается случайная точка в пространстве, и чтобы не бегать с места на место, необходимо сделать задержку, как видно на рисунке ниже, на функцию остановки устанавливается произвольное значение от одной до пяти секунд. То есть ИИ перемещается из точки А в точку Б и останавливается на некоторое время, после чего случайная величина вновь устанавливается на перемещение (Рисунок 33).

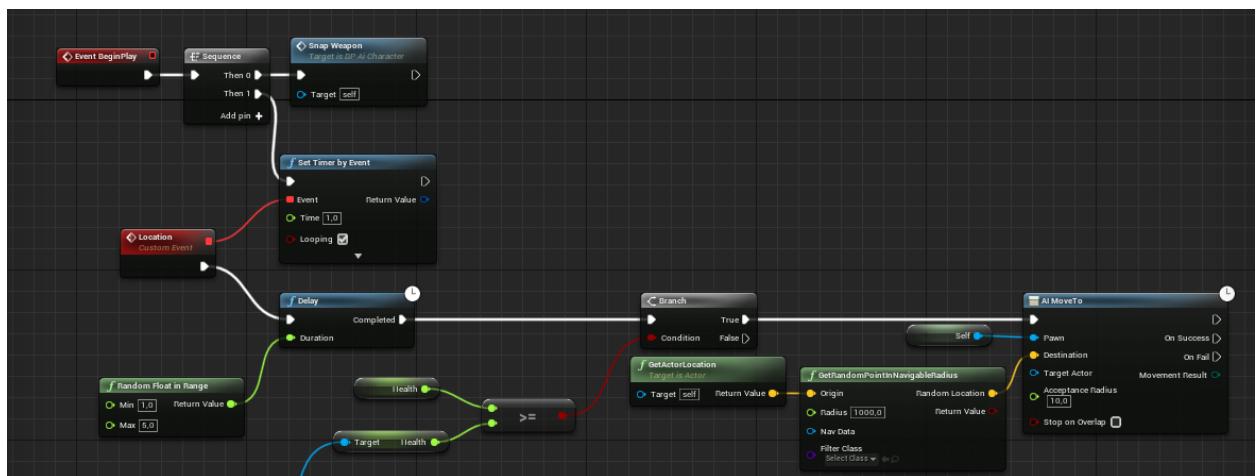


Рисунок 33 — Реализация перемещения ИИ по НС.

3.5. Боевая система

Одна из главных целей ИИ – иметь возможность противостоять игроку в зависимости от текущего условия. Следовательно, он должен иметь возможность атаковать игрока при получении урона. Когда ИИ получает урон, у него отнимается здоровье, и если оно равно нулю, ИИ будет удален с уровня. В случае, когда он получает урон, но здоровье не равно нулю, он начнет преследовать игрока и атаковать, сосредоточившись на нем.

Для обмена определенной информацией используется интерфейс, которую необходимо добавить в обоих классах персонажей. И в классе пули,

соответственно, в интерфейс должна передаваться информация, т.е. сколько урона будет нанесено и по каким объектам.

В момент стрельбы можно попасть в дерево, камень или еще в какой-нибудь объект, поэтому необходимо при встрече пули с объектом сделать проверку. В классе пули будет выполнена проверка, если объект содержит определенный интерфейс, то происходит повреждение, в противном случае он выстреливается без нанесения урона. Стоить отметить, чтобы боты не попали сами по себе, и чтобы определить, кто кому нанес урон, необходимо установить владельца оружия в каждом классе персонажа и в пульке получить владельца (Рисунок 34).

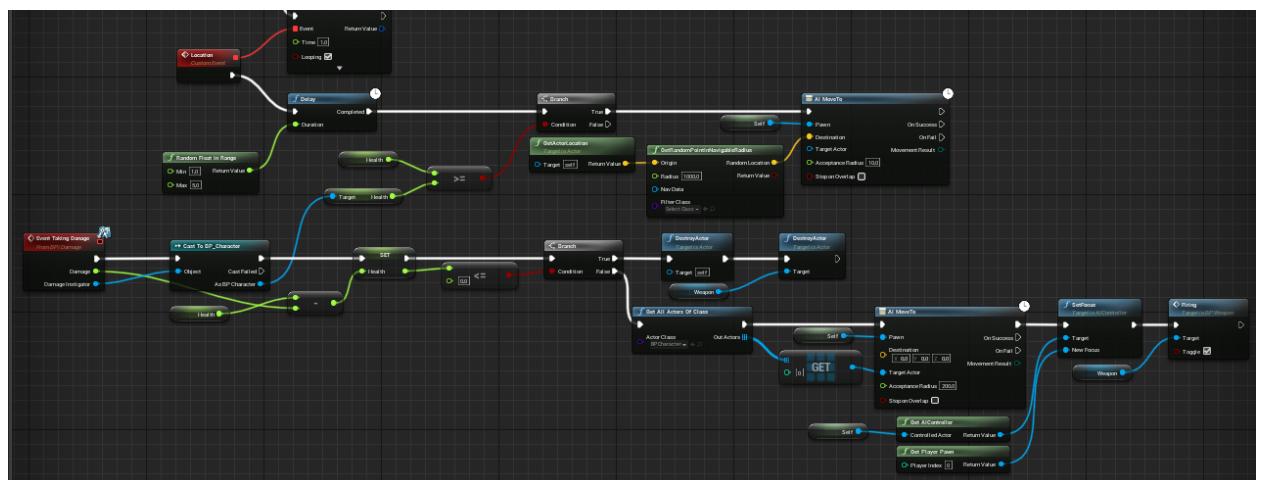


Рисунок 34 — Реализация системы атаки игрока и NPC.

После создания атаки NPC необходимо реализовать нанесения урона по игроку. Логика получения урона был реализован аналогично вышеописанному методу. После того как здоровье игрока будет равен нулю, он заново появиться на уровень там, где он умер. Заново порожденному классу персонажу необходимо назначить контроллер, чтобы вернуть управления над персонажем. Чтобы после перезапуска, боты не продолжали стрелять, необходимо выбрать всех ботов и пробежать по массиву отключив у них стрельбу. То есть из ботов необходимо достать их оружие и выключить стрельбу (Рисунок 35).

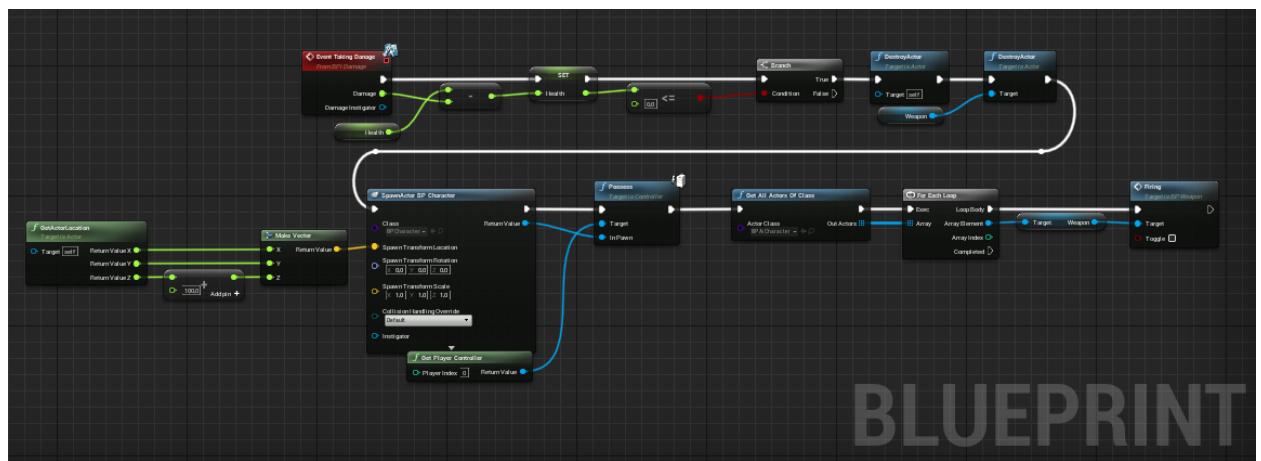


Рисунок 35 — Реализация нанесения урона игроку.

После реализации возрождения игрока, необходимо сделать это и для ботов. Для этого, при старте игры через таймер будет ежесекундная проверка если количество ботов на уровне меньше пяти, то будет порождения ботов до пяти (Рисунок 36).

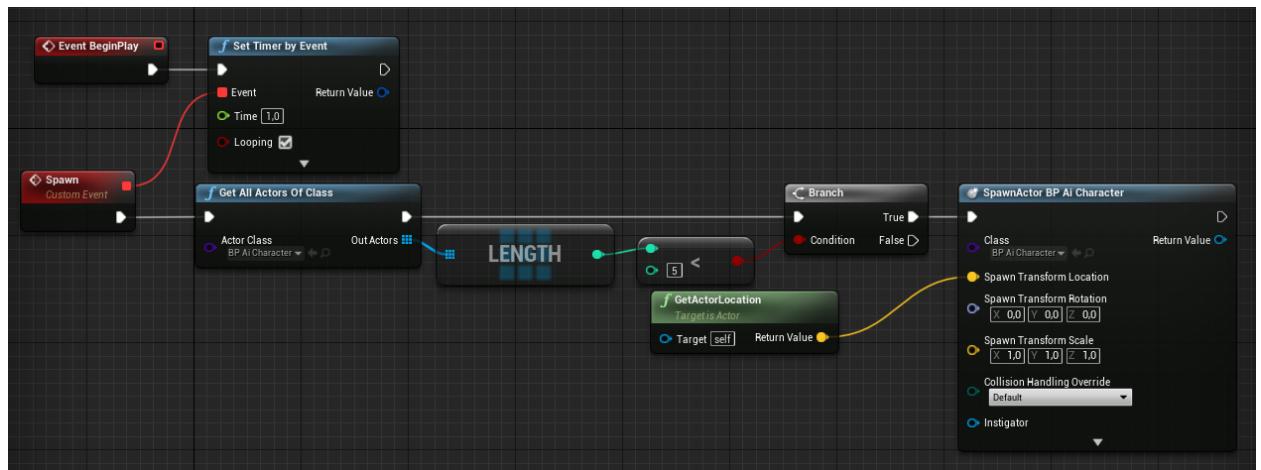


Рисунок 36 — Контроль количества ботов на уровне.

3.6. Cinematic

Sequencer является основным инструментом, который будет использоваться для создания кинематографического контента в Unreal Engine.

Sequencer Editor – это встроенный в Unreal кинематографический редактор для создания кинематографических сцен в игре. Sequencer представляет собой многодорожечный редактор, который работает с камерами и переключается между ними с помощью выбранной пользователем дорожки. Камеры анимируются с помощью ключевой кадровой анимации внутри уровня и на каждой из своих дорожек. Другие объекты, такие как актеры, звуки и анимация, также могут быть воспроизведены путем добавления дорожек через sequencer.

Sequencer оснащен мощными инструментами для работы с камерой, которые позволяют добиться желаемых результатов. Для осуществления съемки разработчик имеет возможность воспользоваться виртуальными рельсами, эффектами дрожания камеры и другими инструментами для создания качественного контента, обеспечивая полный контроль над камерой для съемки эпизодов (Рисунок 37).

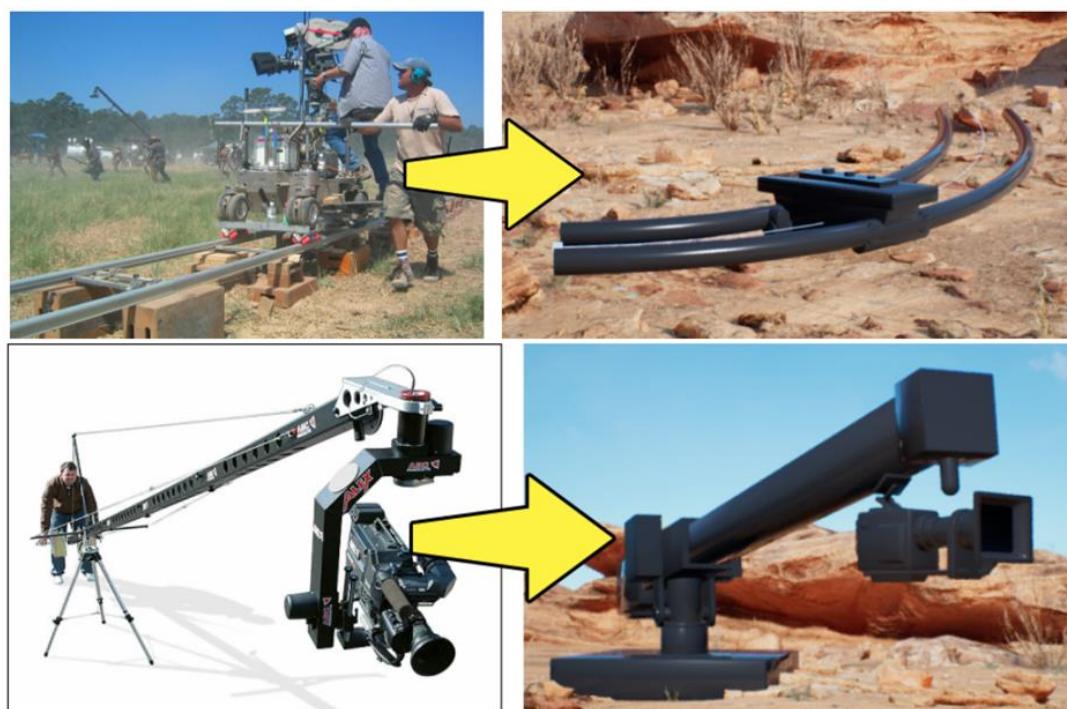


Рисунок 38 — Инструменты движения для съемки игрового фильма.

На стадии разработки кинематографического контента, прежде всего, необходимо было подготовить уровень с качественным освещением, что дает игроку больше погрузиться в эту атмосферу при просмотре трейлера, также были продуманы сценарии, как будет анимирован персонаж, в каком направлении и каким образом. Кроме того, необходимо было уделить пристальное внимание ракурсам камеры и общему темпу работы. Это помогло создать захватывающий и интересный кинематографический игровой фильм.

После того как создан sequencer, следует создать камеру, с помощью которой будет производиться съемка. Кадры, которые будут сниматься, отображаются в области временной шкалы sequencer. Каждый раздел кадра работает как большинство разделов, его можно перемещать, обрезать или редактировать. Таким образом, съемка осуществляется путем управления и добавления ключей (красные и зеленые точки) на дорожке. Если нужно хотите снимать с другого ракурса, можно добавить вторую камеру и управлять обеими, переключаясь между ними (Рисунок 39).

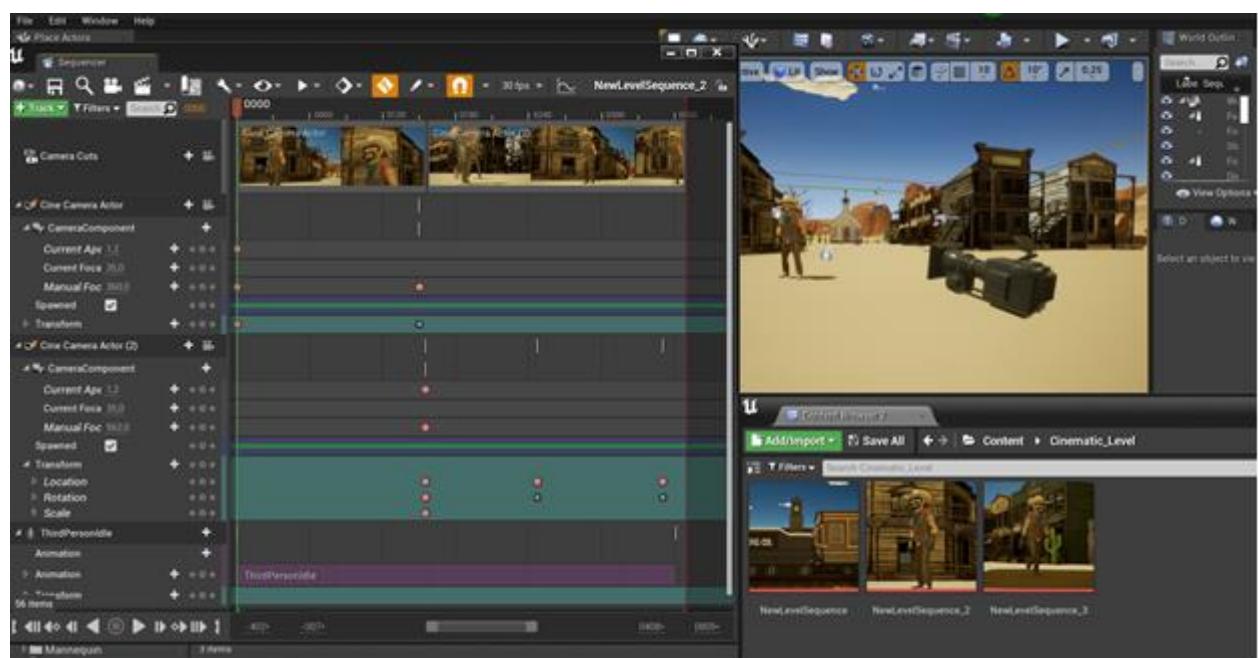


Рисунок 39 — Создание кадров с помощью Level Sequence.

Если нужно анимация идущего вперед персонажа, то в первую очередь мы должны добавить персонажа в качестве дорожки. Sequencer может добавить любой объект, который есть в нашей сцене, а затем он может

анимировать его атрибуты. Когда добавляем объект в sequencer, мы получаем доступ к анимации любого атрибута этого объекта. Таким образом, необходимо добавить анимацию ThirdPersonWalk, затем следует зациклить его до нужной отметки (Рисунок 40).

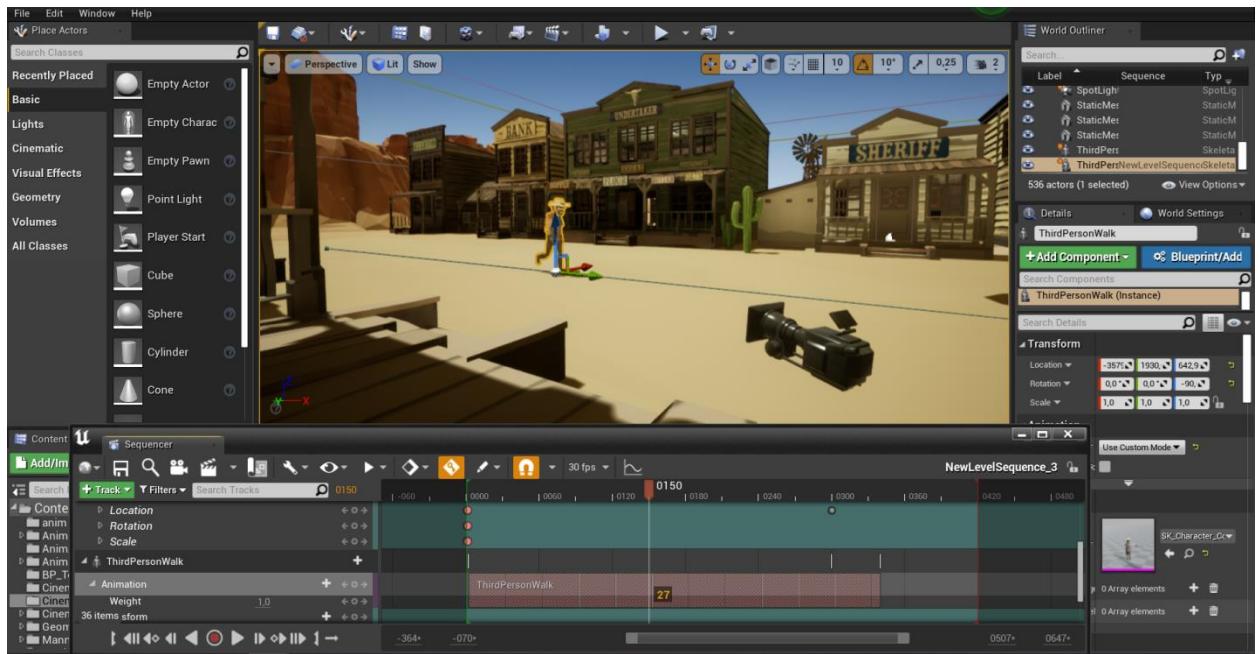


Рисунок 40 — Анимация персонажей.

Подобно тому, что происходит с реальными камерами, глубина резкости применяет размытие к сцене в зависимости от расстояния до или позади фокусной точки. Диафрагма определяет, насколько резкими или размытыми являются передний и задний план, в зависимости от диаметра диафрагмы, который контролируется диафрагмой. Она определяется фокусным расстоянием, деленным на число диафрагмы.

Фокусное расстояние — это расстояние от центра объектива камеры до объекта съемки, который находится в фокусе, создавая фокальную плоскость. Чем ближе камера к объекту, тем больше фон будет не в фокусе. Таким образом, задав меньшее значение диафрагмы 1,2 и фокусное расстояние 360, мы получаем именно такой результат (Рисунок 41).

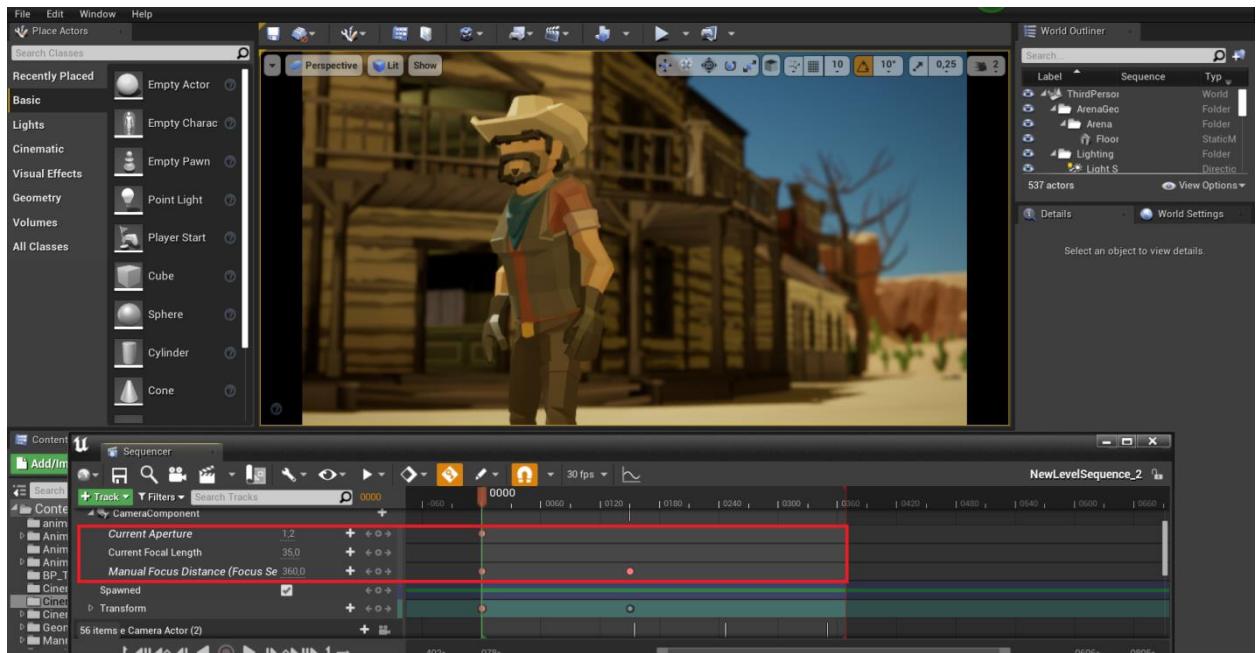


Рисунок 41 — Кинематографические настройки глубины резкости.

Если нужен большой контроль над камерой или другим объектом, применяется редактор кривых для изменения и точной настройки ключевых кадров. График редактора кривых содержит двумерное отображение ключевых кадров и сгенерированных интерполированных кривых. График отображает время и значение на горизонтальной и вертикальной осях соответственно, а ключевые кадры располагаются на графике в соответствии с этими свойствами (Рисунок 42).

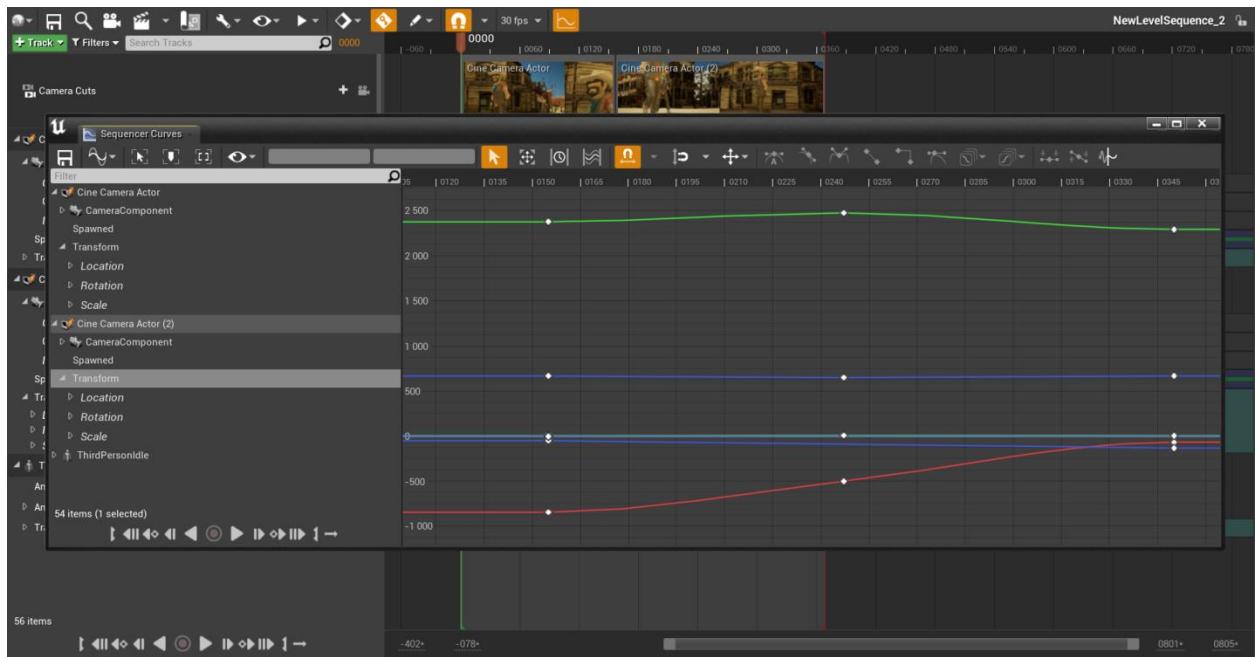


Рисунок 42 — Редактор кривых.

Media Player

Помимо создания эпизодов в sequencer, их необходимо собрать в единый видеофайл с помощью стороннего инструмента с хорошей композицией. Для воспроизведения видеофайл был использован пользовательский интерфейс Widget Blueprint. К тому же при создании медиа-плеера автоматически создается медиа-текстура. Она отвечает за воспроизведение медиа-контента, следовательно, ее и нужно использовать при создании материала, который в свою очередь будет применяться к Widget. Таким образом, в Widget необходимо задать компоненты, материал и определить продолжительность самого видео. По истечении этого времени данный Widget будет удален (Рисунок 43).

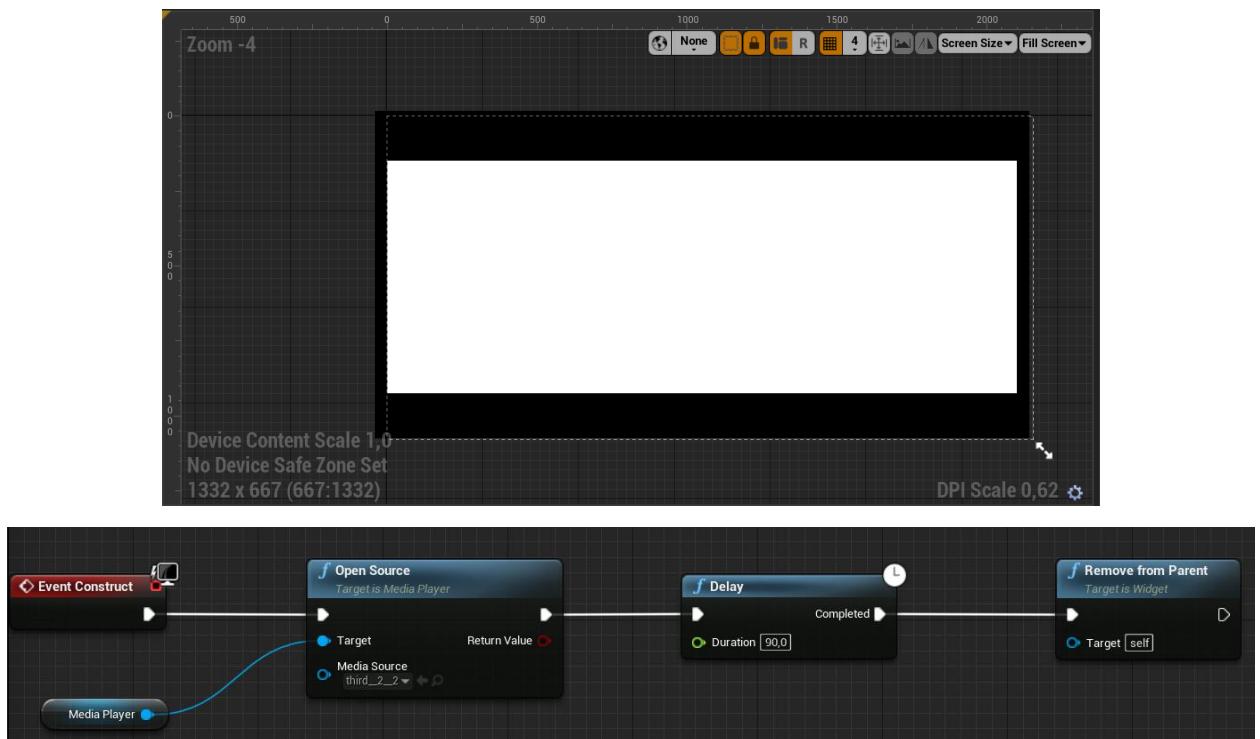


Рисунок 43 — Реализация Widget.

Widget может проигрываться на экране в определенных ситуациях, по триггеру, по нажатию кнопки и т.д. Для этого был создан класс, который будет воспроизводить игровой фильм по триггеру и по нажатию кнопки. Когда игрок входит в зону триггера появляется текст, с надписью нажмите Р для просмотра фильма, в результате чего игрок узнает, что при нажатии

можно посмотреть фильм. Также при просмотре этого фильма до его окончания управление будет отключено, а здоровье персонажа будет скрыто (Рисунок 44).

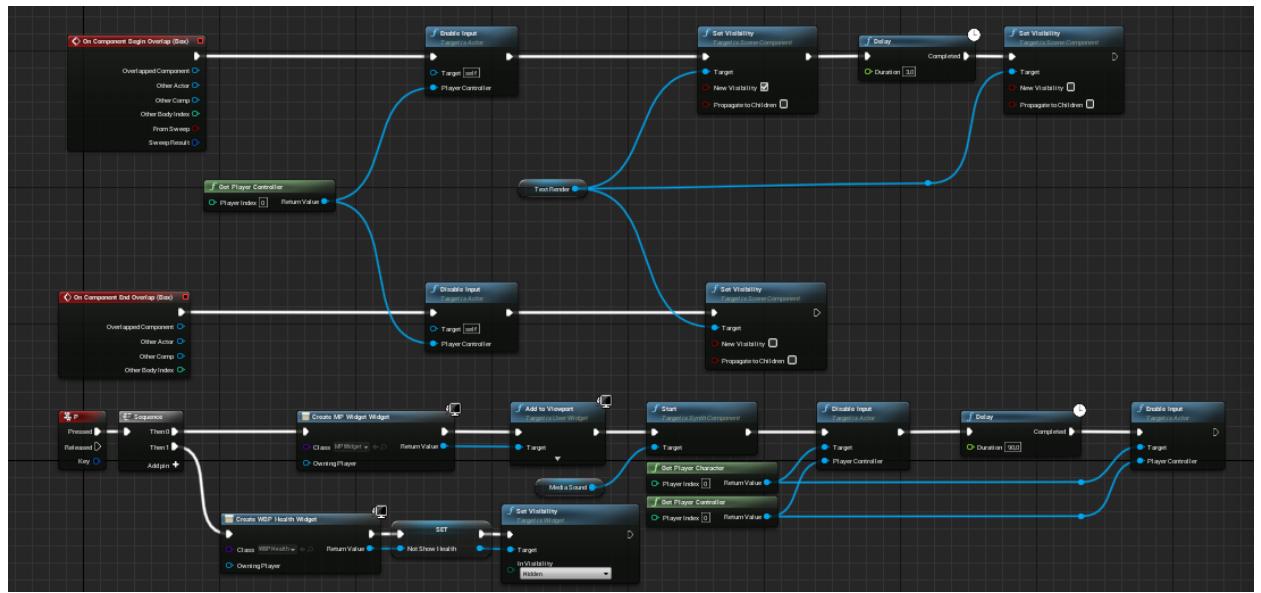


Рисунок 44 — Создание класса для воспроизведения фильма на экран.

Заключение

Во время выполнения выпускной квалификационной работы проводилось исследование в области разработки игры, и были проанализированы все связанные области. В ходе планирования была выбрана наиболее оптимальная среда Unreal Engine 4. Вся дизайнерская работа соответствует темам окружающей среды. Игровая механика максимально интегрировала культуру мира дикого запада, а также предоставила игроку полную свободу в рамках игрового мира.

Также были рассмотрены способы создания короткометражного игрового фильма, который повествует сюжетную линию игры.

Кинематографический игровой фильм показал весьма неплохие результаты, чтобы дать некоторое представление об игре.

Был разработан отзывчивый ИИ, который при получении урона будет преследовать, и атаковать игрока, концентрируясь на нем.

В результате выполнения этой работы была разработана качественная видеоигра в жанре Shooter от третьего лица, в художественном стиле Low poly. Цель заключалась в том, чтобы создать игру. Соответственно, все поставленные и вытекающие из процесса разработки цели были достигнуты.

Список литературы

1. Русскоязычное сообщество Unreal Engine 4 [Электронный ресурс] режим доступа: <https://uengine.ru/> (дата обращения: 12.12.2021).
2. Жанры компьютерных игр [Электронный ресурс] режим доступа: <http://gamesisart.ru/theory.html> (дата обращения: 14.01.2022).
3. Загарских А.С., Хорошавин А.А., Александров Э.Э., Введение в разработку компьютерных игр – СПб: Университет ИТМО, 2019. – 79 с.
4. Assets Unreal Engine [Электронный ресурс] режим доступа: https://uassets_unreal.com/assets-for-game-unreal-engine.html (дата обращения: 22.01.2022).
5. Розенталь, Алан Создание кино и видеофильмов от А до Я / Алан Розенталь. - М.: Триумф, 2016. - 352 с.
6. Основы создание кинематографических роликов [Электронный ресурс] <https://uengine.ru/category/site-content/docs/cinematic> (Дата обращения 23.04.2021).
7. Создание новой модели игрока в Unreal Engine [Электронный ресурс] режим доступа: <http://metalmedved.com/sozdanie-novoj-modeli-igroka-v-unreal-engine.html> (дата обращения: 18.02.2022).
8. Документация по Unreal Engine 4 | Actor Static Mesh [Электронный ресурс] режим доступа: <http://scriptix.ru/en-US/Engine/Actors/StaticMeshActor> (дата обращения: 14.02.2022).
9. Туториал по Unreal Engine: C++ [Электронный ресурс] режим доступа: <https://habr.com/ru/post/348600/> (дата обращения: 03.02.2022).
10. Unreal Engine 4. Учебник для начинающих: Введение в основы [Электронный ресурс] режим доступа: <https://stdpub.com/unrealengine/unreal-engine-4-uchebnik-dlya-nachinayushhih-vvedenie-v-osnovy>
11. Будущее искусственного интеллекта в играх [Электронный ресурс] режим доступа: <https://habr.com/ru/post/130729/> (дата обращения: 16.04.2022).

12. Искусственный интеллект [Электронный ресурс] режим доступа:
<https://uengine.ru/category/site-content/docs/ai> (дата обращения: 28.04.2022).
13. Animation Starter Pack [Электронный ресурс] режим доступа:
<https://www.unrealengine.com/marketplace/en-US/content-cat/assets/animations?count=20&sortBy=effectiveDate&sortDir=DESC&start=0> (дата обращения: 09.05.2022).
14. Создание реалистичных горных сцен в UE4 [Электронный ресурс] режим доступа: <https://ue4daily.com/blog/realistic-mountain-scenes-in-ue4> (дата обращения: 20.03.2022).