Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент *Шендрик Даниил Андреевич*
(фамилия, имя, отчество)

Группа *РК6-84Б*

Тип практики *Преддипломная*

Название предприятия *НИИ АПП МГТУ им. Н.Э. Баумана*

Студент РК6-84Б _____ *Шендрик Д.А.*
(Группа) (подпись, дата)

Руководитель практики
от кафедры _____ *Витюков Ф.А.*
(подпись, дата)

Оценка _____

Москва, 2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой *РК-6*
(Индекс)

_____ *А.П. Карпенко*
(И.О.Фамилия)

«____» _____ *2024* г.

З А Д А Н И Е
на прохождение преддипломной практики (производственная)

Студент *Шендрик Даниил Андреевич, 4* курса, группы *РК6-84Б*
(фамилия, имя, отчество, № курса, индекс группы)

в период с *«13» мая 2024 г.* по *«26» мая 2024 г.*

Предприятие: *НИИ АПП МГТУ им. Н.Э. Баумана*

Подразделение: _____
(отдел/сектор/цех)

Руководитель практики от предприятия (наставник):
Киселев Игорь Алексеевич, директор НИИ АПП МГТУ им. Н.Э.Баумана
(фамилия имя отчество полностью, должность)

Руководитель практики от кафедры:
Витюков Федор Андреевич, старший преподаватель
(фамилия имя отчество полностью, должность)

Задание.

1. Разработать несколько частей видеоигровой боевой системы:
    а) система здоровья;
    б) система атак с умной буферизацией;
    в) система центрирования камеры на определенном противнике (Attach-режим).
2. Дополнительно создать отображение здоровья с помощью виджетов и объектов.
3. Реализовать анимацию движения персонажа, зависящую от направления движения.

Дата выдачи задания *«13» мая 2024 г.*

Руководитель практики от предприятия _____ Киселев И.А.
(подпись, дата)

Руководитель практики от кафедры _____ Витюков Ф.А.
(подпись, дата)

Студент РК6-84Б _____ Шендрик Д.А.
(группа) (подпись, дата)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Unreal Engine — это мощный и универсальный движок для создания игр и визуальных приложений. Разработанный компанией Epic Games, Unreal Engine широко используется в игровой индустрии, а также в различных отраслях, связанных с визуализацией и виртуальной реальностью.

Unreal Engine предоставляет разработчикам инструменты для создания интерактивных 3D-приложений, использующих передовые технологии визуализации и физики. Он имеет множество прикладных функций, таких как работа с текстурами и материалами, рендеринг графики, создание и управление объектами в игровом поле, мощный высокоуровневый язык C++ для написания игровой логики и многое другое.

Одной из основных сфер применения Unreal Engine является видеоигровая индустрия. На Unreal Engine создаются крупные блокбастеры по всему миру, позволяя игрокам насладится различными мирами, и испытать неповторимый опыт с различными игровыми механиками, реализованными на очень гибком движке. Но не только крупные студии, но даже простые пользователи могут попробовать освоить UE, и создать нечто простое или настоящий инди проект мечты.

Как пример возможностей UE для простых разработчиков, в данной работе была создана часть простой видеоигровой боевой системы.

Обычная боевая система включает в себя базовый набор действий и маневров для игрового персонажа под управлением пользователя, обеспечивающих подвижность персонажа и его взаимодействие с миром. Персонаж сможет выполнять различные действия, такие как ходьба, бег, удары, перекаты, а также занимать защитную стойку для блокировки ударов.

Кроме того, для демонстрации боевой системы обеспечивается умное поведение противника. Искусственный интеллект, управляющий противником, обладает аналогичным набором действий и умениями, как и у игрового персонажа.

Работа направлена на определение оптимальных комбинаций этих инструментов и программ для достижения выдающегося уровня интерактивности и реализма виртуальных персонажей и игровых механик, для взаимодействия с игровым миром в Unreal Engine.

Цель работы: ознакомится и применить на практике возможности движка Unreal Engine 4 для создания части видеоигровой боевой системы, включающей в себя систему здоровья, систему атак с умной буферизацией и систему центрирования камеры.

1. ПОСТАНОВКА ЗАДАЧИ

Исследовать и разработать несколько частей видеоигровой боевой системы:

а) система здоровья;

б) система атак с умной буферизацией;

в) система центрирования камеры на определенном противнике (Attach-режим).

Дополнительно создать отображение здоровья с помощью виджетов и объектов.

Реализовать анимацию движения персонажа, зависящую от направления движения (в Attach-режиме).

2. КРАТКИЙ ОТЧЕТ О ВЫПОЛНЕННЫХ РАБОТАХ

Unreal Engine – распространенный движок для разработки различных видеоигр. Среди таких видеоигр распространен жанр Action RPG, это трехмерные игры, где игрок управляет лишь одним персонажем с видом от первого или третьего лица, когда основной упор в игре сделан на бои в ближнем бою с использованием различного вооружения. Для подобных игр характерна возможность передвижения персонажа по поверхности карты и свободный обзор камеры. В контексте боев могут быть несколько типов атак с разной скоростью, уроном и анимацией; также и защитные действия также могут быть различными: блокирование оружием или щитом, уклонения или перекаты.

Для учебной разработки был выбран тип разрабатываемой игры от третьего лица и с определенным набором действий для персонажей: ходьба, бег, перекаты, удар, блок. Была доработана программа на C++ для Unreal Engine 4.2, которая уже умеет управлять действиями персонажей, считывает нажатие определенных клавиш игроком и создает неплохой прототип боевой системы. Далее к ней добавляются части кода, реализующие подсистемы, описанные в задаче на разработку.

2.1. Реализация системы здоровья

Система здоровья является одной из ключевых механик боевой системы, обеспечивая реалистичное и интуитивное взаимодействие персонажей в игре. В данной системе предусмотрены различные компоненты, которые позволяют как игроку, так и противникам эффективно управлять своим здоровьем и реагировать на полученные повреждения.

2.1.1. Здоровье персонажей

Каждый персонаж, будь то герой или противник, обладает переменной здоровья, которая определяет их жизнеспособность в игре. Для реализации этой механики в классах персонажей (**New_Protagonist** и **New_Antoganist**) введены две основные переменные:

- **Health** – текущее количество здоровья персонажа. Эта переменная изменяется в ходе игры при получении урона или лечении.
- **Max_Health** – максимальное количество здоровья, которое может иметь персонаж. По умолчанию это значение установлено на уровне 120, однако разработчики оставили возможность изменения этого параметра через редактор, чтобы обеспечить гибкость в настройке персонажей.

Уменьшение здоровья происходит при получении персонажем урона. В этот момент переменная **Is_get_damage** устанавливается в значение **true**, что инициирует процесс уменьшения текущего здоровья (**Health**). Важно отметить, что внутри классов персонажей не вызываются дополнительные события при изменении здоровья, за исключением достижения здоровья нулевого уровня. Когда здоровье персонажа падает до нуля, переменная **Is_dead** устанавливается в значение **true**, сигнализируя о смерти персонажа. Это событие запускает анимацию умирания и последующие действия, такие как респавн противника или появление окна с оповещением для героя.

2.1.2. Визуальные элементы, показывающие здоровье

Для обеспечения визуальной обратной связи игроку реализованы различные элементы интерфейса, отображающие текущее состояние здоровья персонажей.

Рисунок 1 – Виджет здоровья персонажа

Для отображения здоровья игрового персонажа используется виджет здоровья героя (**Health_Protagonist**), который был разработан с использованием системы Unreal Motion Graphics (UMG). Этот виджет включает в себя элемент **Progress Bar**, который графически отображает текущее состояние здоровья героя (Рисунок 1). При инициализации виджет получает значение максимального здоровья (**Max_Health**). Далее, каждый игровой такт, виджет обновляет процент заполнения **Progress Bar**, деля текущее здоровье (**Health**) на максимальное. Blueprint виджета, который выполняет процесс его изменения изображен на Рисунок 2. Это позволяет игроку всегда видеть актуальное состояние здоровья своего персонажа, что важно для принятия тактических решений в бою.

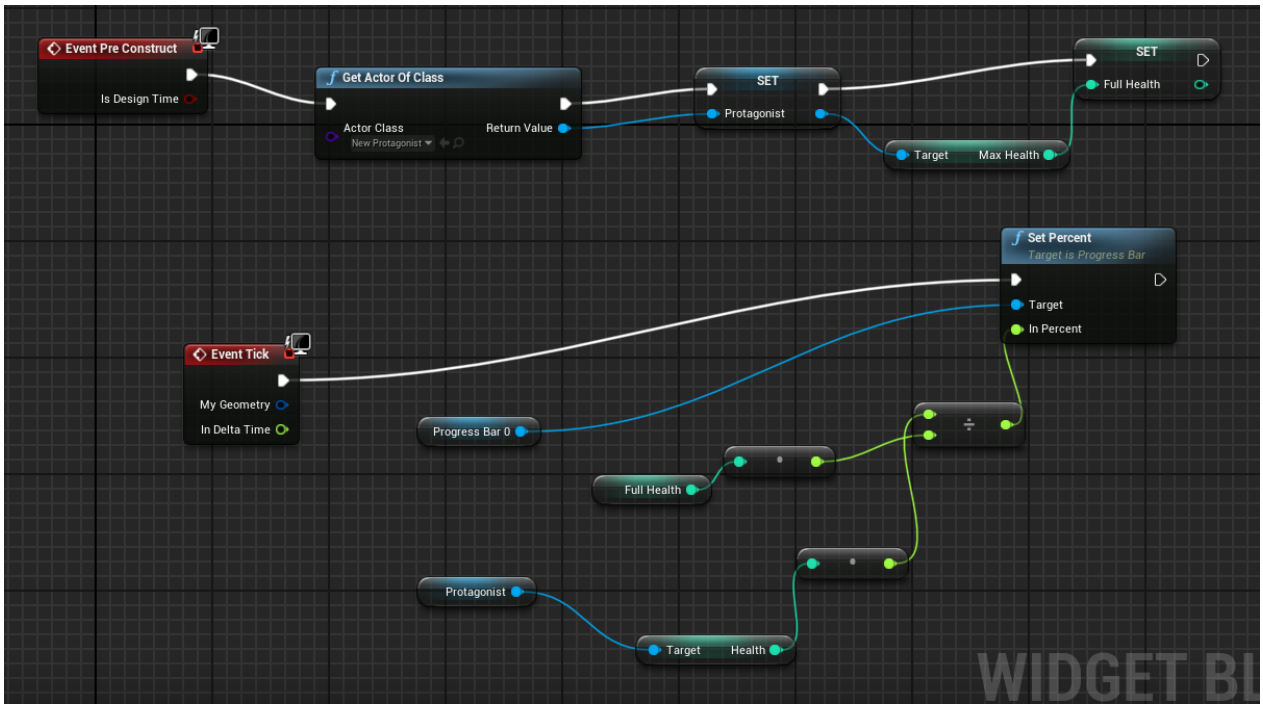

Рисунок 2 – Blueprint для виджета здоровья игрока

Для отображения здоровья противников используется физический объект (**Health_Bar_Antagonist**), представляющий собой элемент класса **Plane**. Этот объект отображается над головой противника и динамически изменяется в зависимости от его текущего здоровья.

Рисунок 3 – Отображение здоровья противника с помощью объекта Plane

При инициализации объект получает значение максимального здоровья противника и затем, каждый игровой такт, масштабируется по оси Y, деля текущее здоровье на максимальное. Положение **Health_Bar_Antagonist** обновляется через вызов **GetActorLocation** у объекта класса **New_Antoganist**, обеспечивая точное соответствие позиции шкалы и противника. Поворот объекта высчитывается через разницу между положением игрока и противника. Затем данный вектор преобразуется к типу FRotator, отвечающий за угол поворота объекта по всем трем осям. Весь процесс изменения объекта можно увидеть на рисунке Рисунок 4, где изображен EventGraph шкалы здоровья противника.

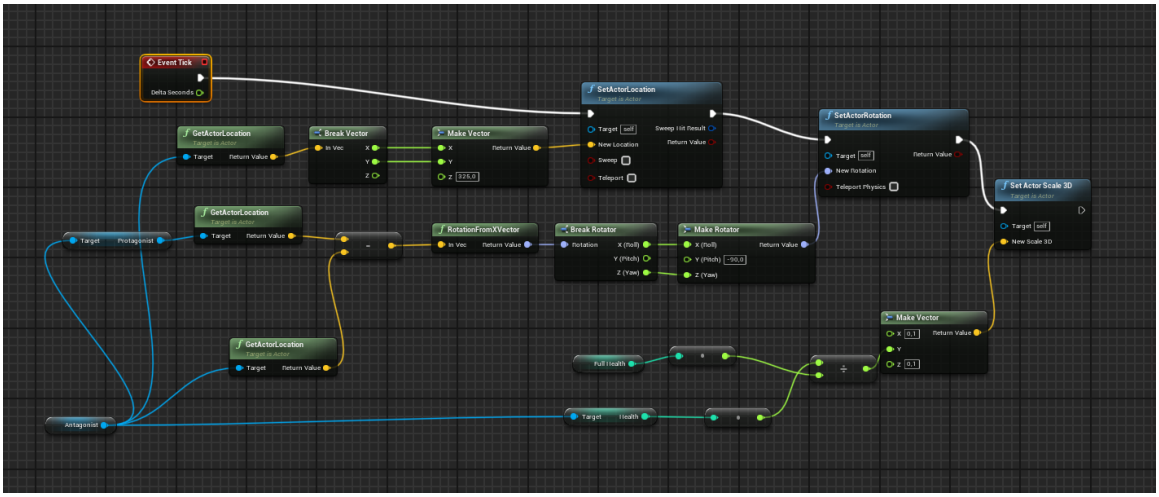

Рисунок 4 – Blueprint для виджета здоровья противника

2.1.3. Процесс респавна противников

Процесс респавна противников был разработан для поддержания постоянного уровня вызова для игрока и динамичного игрового процесса. Когда здоровье противника достигает нулевого уровня, и заканчивается анимация смерти, система вводит паузу на 3 секунды. Это создаёт реалистичный эффект задержки перед исчезновением тела. Затем вызывается функция **DestroyActor**, которая удаляет противника из игрового мира.

Для выбора новой позиции респавна используется случайный выбор точки в радиусе 1000 метров от исходной позиции. Новый противник создаётся с помощью функции **SpawnActor** в этой позиции и получает все параметры, аналогичные уничтоженному персонажу. Такой подход обеспечивает постоянное присутствие противников в игре, поддерживая напряжение и динамику боя. Фрагмент Level Blueprint, осуществляющий данную функцию представлен на рисунке Рисунок 5.

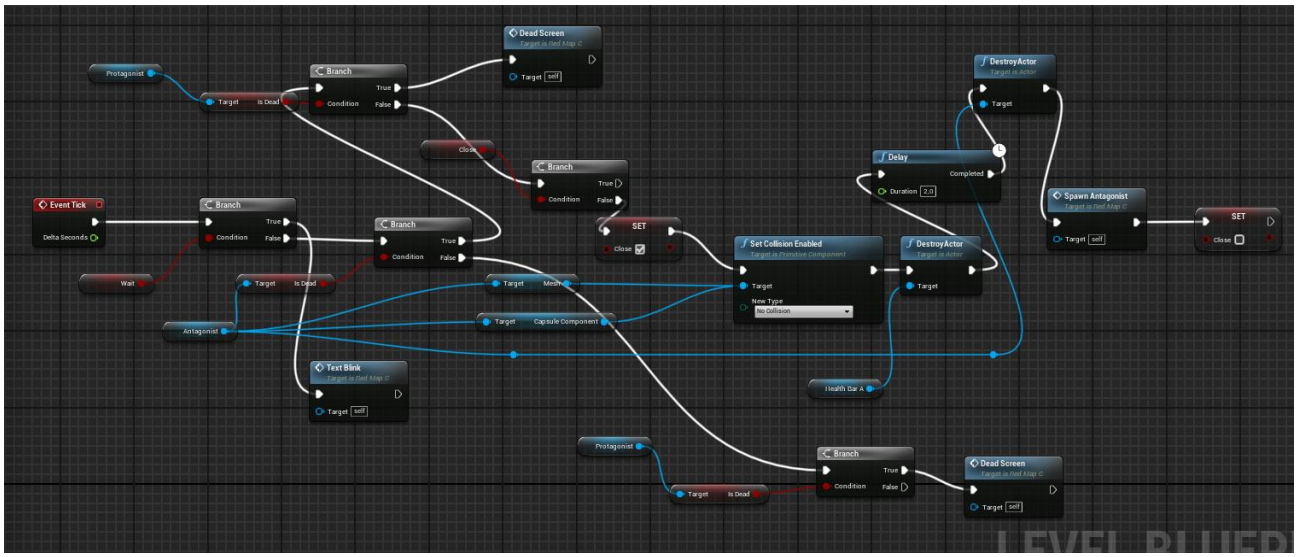

Рисунок 5 – Level Blueprint для выбора действий после смерти персонажа

2.1.4. Смерть игрового персонажа

После смерти игрового персонажа выводится специальный интерфейсный элемент, информирующий игрока о гибели героя. Для этого используется виджет **Died_Screen**, который состоит из трёх элементов **Image**. Когда

переменная **Is_dead** становится истиной, срабатывает анимация смерти персонажа. Также это событие вызывает срабатывание соответствующего кода в **Level Blueprint** (Рисунок 5). Виджет начинает плавно изменять свою прозрачность от 0 до 1, заполняя весь экран.

В этот момент все действия игрока, включая вращение камеры, становятся невозможными, что реализовано в классе **New_Protagonist**. На экране появляется картинка с оповещением о смерти персонажа и мигающая надпись "Press Enter to continue". По нажатию клавиши **Enter** переменная **Is_dead** сбрасывается в значение **false**, позволяя игроку снова управлять персонажем. Виджет становится прозрачным, и игровой процесс продолжается (Рисунок 6). Этот элемент интерфейса создан для обеспечения четкой и понятной обратной связи игроку в случае гибели его персонажа.

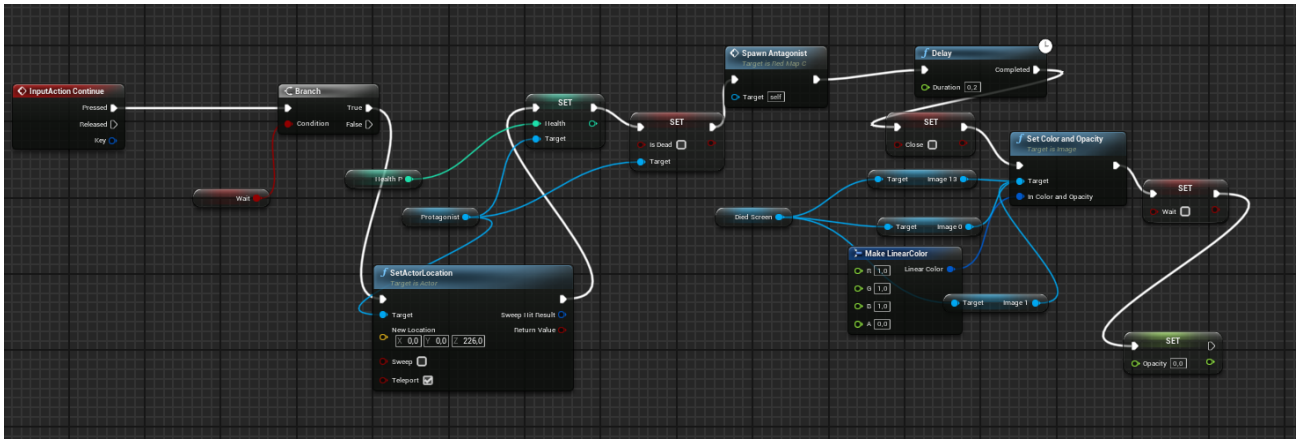

Рисунок 6 – Фрагмент Level Blueprint для «возрождения» игрока

2.2. Реализация системы атак с умным буфером

Система умной буферизации атак в игре была разработана для повышения отзывчивости и плавности боевых действий. В отличие от традиционного режима атак, где последовательные нажатия кнопки мыши могут прерывать текущие анимации ударов, умная буферизация позволяет игроку непрерывно кликать, и анимации будут корректно воспроизводиться. Это достигается за счет использования простого буфера, который заполняется в момент нажатия кнопки мыши во время выполнения атаки. Когда первая анимация удара завершается, система проверяет состояние буфера и, если он

активен, запускает следующую анимацию атаки. В противном случае персонаж переходит в состояние покоя.

Для реализации буфера используется булева переменная **Attack_buf** в классе игрового персонажа **New_Protagonist**. Эта переменная служит индикатором нажатия кнопки мыши и становится истиной каждый раз при нажатии левой кнопки мыши (ЛКМ). Переменная проверяется в момент завершения текущей анимации атаки, определяя, должна ли начаться следующая анимация.

Нажатия ЛКМ регистрируются с помощью системной функции **BindAction**, что позволяет корректно отслеживать все попытки игрока атаковать. Несмотря на то, что игрок может многократно нажимать кнопку мыши во время выполнения одной атаки, переменная **Attack_buf** является булевой, что гарантирует запуск только одной следующей атаки после завершения текущей. Таким образом, система эффективно предотвращает переполнение буфера и обеспечивает стабильное поведение атак.

Переход в состояние атаки определяется переменной **Is_attacking**, которая становится истиной при начале атаки. Это состояние контролируется и управляется кодом, обеспечивая четкое разграничение между активной фазой атаки и фазой покоя. Когда персонаж атакует, **Is_attacking** установлена в значение **true**, что сигнализирует системе об активном выполнении боевых действий.

На текущий момент система поддерживает две основные анимации атаки: удар справа и удар слева. Эти анимации плавно переходят друг в друга при постоянном нажатии ЛКМ, создавая иллюзию непрерывной атаки. Переходы между анимациями реализованы в **AnimGraph** (Рисунок 7), что позволяет гибко управлять последовательностью атак и обеспечивать плавные переходы между движениями. При этом вначале всегда идет анимация первого удара справа, но закончится и перейти в состояние покоя можно из любой из двух анимации атаки.

Конец текущей анимации определяется переменной **Is_attacking**, которая переходит в значение **false** после определенной задержки, установленной в коде класса игрового персонажа. Этот механизм позволяет точно определить момент завершения атаки и проверить состояние буфера **Attack_buf**. Если буфер активен, запускается следующая анимация атаки; если нет – персонаж возвращается в состояние покоя.

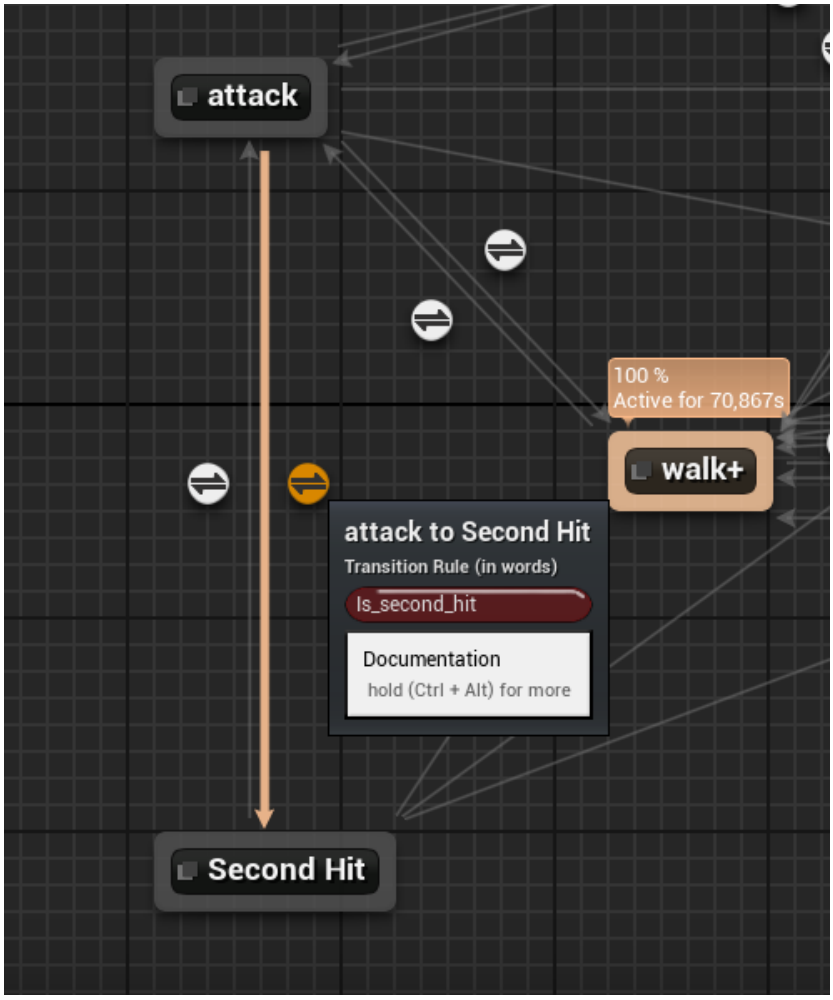

Рисунок 7 – Фрагмент AnimGraph, реализующий переход между анимациями атак и состоянием покоя

Система умной буферизации атак значительно улучшает игровой опыт, делая боевые действия более плавными и интуитивно понятными. Возможность непрерывного нажатия кнопки мыши и автоматического воспроизведения анимаций атак повышает уровень вовлеченности и удовлетворенности игроков. Тщательная реализация и оптимизация системы обеспечивают ее надежную работу и высокую производительность, что является важным аспектом успешного игрового проекта.

2.3. Реализация системы визуального захвата противника

Режим фиксации камеры, также известный как Attach-режим, предоставляет игроку возможность сосредоточиться на конкретном противнике, обеспечивая постоянное наблюдение за ним и упрощая управление боевыми действиями. Этот режим делает бой более интуитивным и удобным, позволяя игроку удерживать камеру и персонажа направленными на выбранную цель.

2.3.1. Работа камеры в режиме фиксации

Режим фиксации камеры может быть активирован двумя способами:

1. Самостоятельная активация. Игрок может вручную активировать режим фиксации камеры, нажав среднюю кнопку мыши (СКМ). В этот момент переменная **Is_target_set** класса игрового персонажа устанавливается в значение **true**, что сигнализирует системе о необходимости фиксации на цели.
2. Активация при блокировании. Переход в режим фиксации камеры также происходит автоматически при переходе в режим блока с помощью правой кнопки мыши (ПКМ). Ближайший противник становится целью, и система начинает фиксировать камеру на нем.

В обоих случаях целевым противником становится ближайший к игроку враг. Это позволяет игроку быстро и эффективно сосредоточиться на текущей угрозе.

Для реализации фиксации камеры используется функция **SetRotation** контроллера игрового персонажа. Угол поворота контроллера вычисляется как вектор между игроком и ближайшим противником. Эта функция вызывается каждый такт, что обеспечивает постоянное направление камеры на цель, независимо от ее движений.

При активации режима фиксации камеры персонаж автоматически поворачивается лицом к выбранному противнику. Это достигается путем установки ротации контроллера в каждый такт, что синхронизирует направление

персонажа и камеры с позицией противника. Таким образом, игрок всегда видит своего врага и может точно контролировать боевые действия.

Режим фиксации камеры может быть деактивирован двумя способами:

1. Повторное нажатие СКМ. Игрок может вручную отключить режим фиксации камеры, снова нажав среднюю кнопку мыши. Это позволяет быстро вернуться к обычному режиму управления камерой.
2. Выход из режима блока. При выходе из режима блока, активированного правой кнопкой мыши, режим фиксации камеры также автоматически отключается.

2.3.2. Анимации в режиме фиксации камеры

В режиме фиксации камеры анимации персонажа играют ключевую роль, обеспечивая реалистичность и плавность его движений. Поскольку в этом режиме персонаж всегда повернут лицом к выбранному противнику, а его движения могут быть направлены в разные стороны, используются специальные анимации, зависящие от направления движения. Это достигается за счет сочетания нескольких технологий и инструментов, доступных в Unreal Engine.

Одним из ключевых элементов системы анимации является использование вектора скорости (**Velocity**), который задается персонажу при нажатии клавиш направления. Хотя сам поворот персонажа остается неизменным и всегда ориентирован на противника, его движения осуществляются в различных направлениях, что позволяет использовать разнообразные анимации для каждой ситуации.

Персонажу задается вектор скорости, соответствующий направлению его движения. Этот вектор обновляется каждый такт, когда игрок нажимает клавиши движения (W, A, S, D). Вектор **Velocity** позволяет определить точное направление движения персонажа относительно его текущего поворота.

В Animation Blueprint направление движения вычисляется как угол от -180 до 180 градусов, который получается путем преобразования вектора **Velocity** и поворота персонажа (Рисунок 8). Этот угол используется для выбора

соответствующей анимации, что делает движения персонажа реалистичными и адаптивными к текущей ситуации.

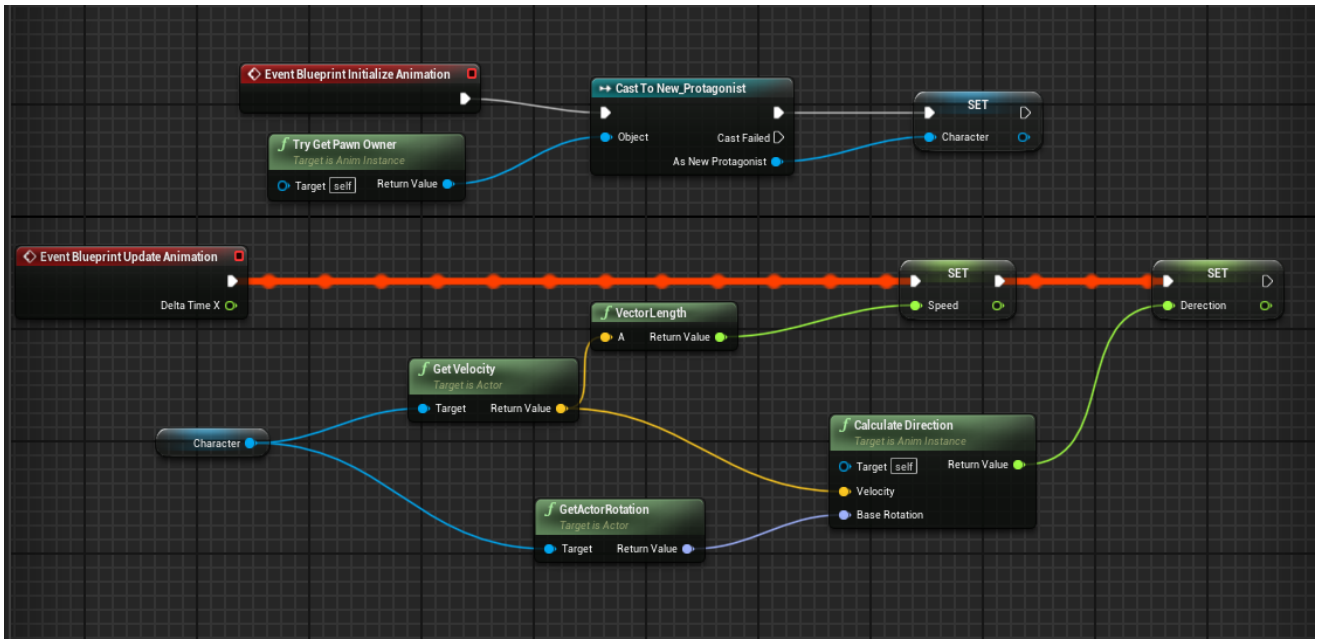

Рисунок 8 – EventGraph, вычисляющий направления и скорости анимации

Для обеспечения плавных переходов между анимациями используется структура **Blend Space**. **Blend Space** представляет собой набор анимаций, организованных по шкале направления и скорости. В зависимости от угла направления и величины скорости персонажа **Blend Space** выбирает подходящую анимацию или комбинацию анимаций, создавая плавный и непрерывный переход между различными состояниями движения (Рисунок 9).

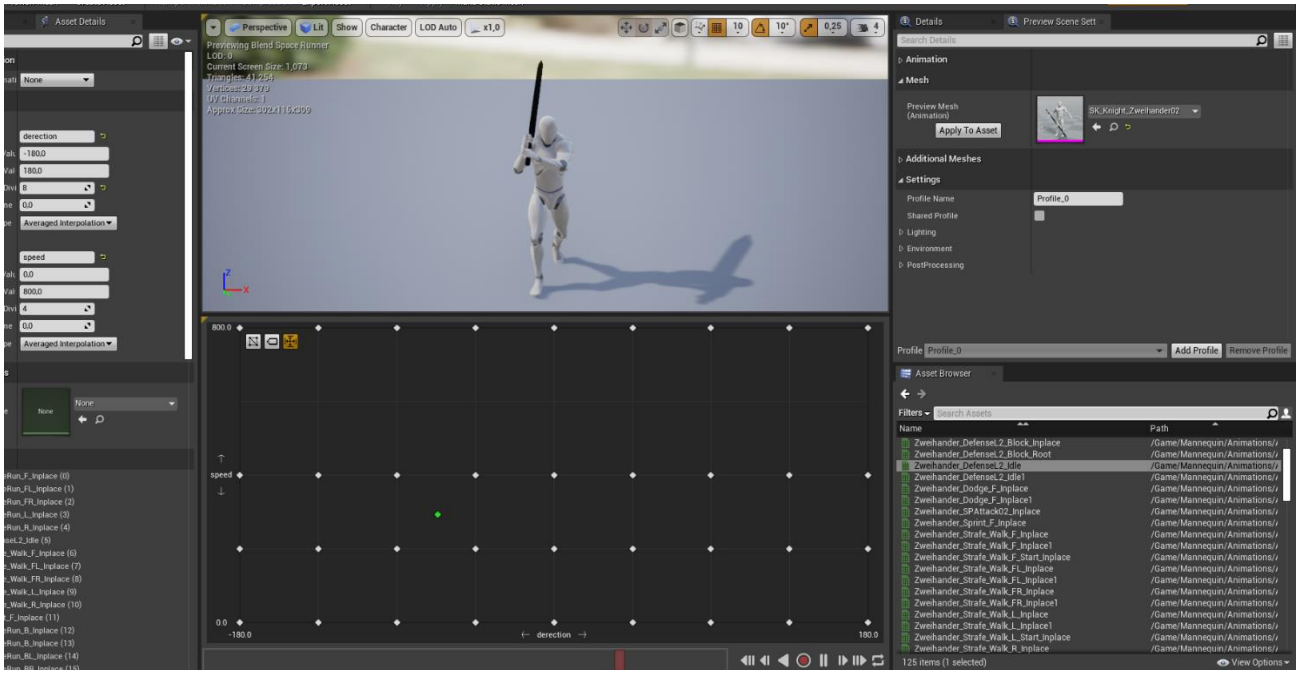

Рисунок 9 – Окно редактора Blend Space

Например, при движении персонажа вперед с высокой скоростью будет использована анимация бега вперед, в то время как при движении влево или вправо – соответствующие анимации бокового шага. При этом, если персонаж движется под углом, **Blend Space** позволяет смешивать анимации, создавая плавный переход, который соответствует реальному движению.

В режиме фиксации камеры предусмотрены следующие типы анимаций, зависящих от направления движения:

- **Ходьба:** Анимации ходьбы адаптированы для движения в различных направлениях. Например, при движении вперед используется анимация ходьбы вперед, а при движении назад – анимация шагов назад. Боковые шаги также имеют свои уникальные анимации, которые используются при движении влево и вправо.
- **Бег:** В режиме фиксации камеры бег сопровождается анимациями, соответствующими направлению движения. Например, бег вперед, назад, влево и вправо имеют свои специфические анимации. **Blend Space** позволяет плавно переходить между этими анимациями в зависимости от направления и скорости движения.
- **Перекаты:** Перекаты выполняются с учетом направления и позиции противника. Анимации перекатов направлены на то, чтобы персонаж мог

быстро уклониться от атаки или переместиться в более выгодное положение. При этом направление переката зависит от текущего движения персонажа и его ориентации относительно противника.

3. АНАЛИЗ РЕЗУЛЬТАТОВ

В ходе разработки и реализации боевой системы для трехмерной игры с видом от третьего лица на Unreal Engine были внедрены несколько ключевых механик, которые значительно улучшили игровой процесс. В этом разделе мы подведем итоги проделанной работы, проанализируем достигнутые результаты и оценим влияние каждой из внедренных систем на общий игровой опыт.

Одним из фундаментальных элементов боевой системы стала система здоровья. Каждому персонажу, включая игрового персонажа (New_Protagonist) и противника (New_Antagonist), были добавлены переменные Health и Max_Health. Эти переменные отслеживают текущее и максимальное количество здоровья соответственно. Реализована визуализация здоровья через виджеты для игрока и плавающие объекты для противников, что обеспечивает игрока важной информацией о состоянии персонажей в режиме реального времени.

Успешное отображение здоровья улучшает взаимодействие игрока с игровым миром, позволяя быстро оценивать боевую обстановку и принимать обоснованные тактические решения. Процесс респавна противников и появление экрана с оповещением после смерти игрока добавляют глубины и реалистичности игровому процессу, делая его более динамичным и увлекательным.

Вторая важная реализация – система атак из буфера. В отличие от обычного режима атак, где анимации могли прерываться, умная буферизация позволяет игроку непрерывно кликать мышью, и анимации атак будут корректно отображаться. Переменная Attack_buf типа bool регистрирует нажатия клавиш и активируется после окончания текущей анимации атаки, если она была нажата во время выполнения предыдущего удара.

Этот подход делает боевую систему более отзывчивой и интуитивно понятной для игрока, обеспечивая плавный переход между анимациями и позволяя сосредоточиться на стратегии боя. Возможность легко комбинировать атаки и поддерживать высокий темп сражения значительно повышает качество игрового опыта.

Одним из ключевых аспектов боевой системы стал режим фиксации камеры, или Attach-режим. Эта система позволяет камере автоматически фиксироваться на ближайшем противнике, обеспечивая постоянное отслеживание и выравнивание на него. Режим активируется через нажатие СКМ или переход в блок с помощью ПКМ, что делает его легко доступным в любой момент боя.

Фиксация камеры на противнике улучшает управление персонажем и делает боевые сцены более кинематографичными. Специальные анимации для ходьбы, бега и перекатов в режиме фиксации камеры обеспечивают плавный и реалистичный внешний вид персонажа. Это добавляет игре тактическую глубину и помогает игроку сосредоточиться на действиях противника.

Разработанные механики формируют прочный фундамент для дальнейшего развития и улучшения боевой системы.

ЗАКЛЮЧЕНИЕ

В процессе данной работы был исследован многогранный инструментарий Unreal Engine 4.2. Комплексное использование моделей и анимации было продемонстрировано в рамках разработки боевой системы для 3-х мерного видео игрового экшена для Unreal Engine 4.2. В рамках разработки был написан C++ код управляющий как действиями персонажа под управлением игрока, так и противника, который полностью управляется с помощью написанного кода.

Внедренные механики – система здоровья, система атак из буфера и режим фиксации камеры – вместе создают комплексную и сбалансированную боевую систему. Они обеспечивают игрока необходимыми инструментами для эффективного взаимодействия с игровым миром и противниками, делая игровой процесс более увлекательным и насыщенным.

Система здоровья не только обеспечивает реалистичное отображение состояния персонажей, но и добавляет стратегическую глубину, требуя от игрока внимательного контроля за своим состоянием и состоянием противников. Система атак из буфера делает боевые сцены более динамичными и позволяет игроку более точно контролировать свои действия. Режим фиксации камеры улучшает управление и делает сражения более зрелищными.

Эти механики формируют прочный фундамент для дальнейшего развития и улучшения боевой системы. Возможные будущие улучшения могут включать расширение набора атак и анимаций, внедрение дополнительных тактических возможностей и более глубокую интеграцию различных игровых механик. В конечном итоге, эти изменения направлены на создание захватывающего и насыщенного игрового опыта, который будет привлекать и удерживать игроков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unreal Engine 4 Documentation [Электронный ресурс] // Unreal Engine Documentation. URL: https://docs.unrealengine.com/. (Дата обращения: 14.05.2024)
2. Правила унификации при работе с UE [Электронный ресурс] // Naming Conventions in Unreal Engine URL: https://github.com/Allar/ue5-style-guide#structure. (Дата обращения: 17.05.2024)
3. Обучающий курс “Продвинутый курс по анимации в UE” [Электронный ресурс] // YouTube. URL: https://www.youtube.com/playlist?list=PLUi8nuTUEtTvTb8Wk6cBfvw8IEFAPZYqd. (Дата обращения: 20.05.2024)
4. Обучающий курс “Основы управление анимацией из C++” [Электронный ресурс] // YouTube. URL: https://www.youtube.com/watch?v=ftjflBJwFAU. (Дата обращения: 20.05.2024)
5. Цикл статей «Создание Survival Horror в стиле RE2 на Unreal Engine и C++» [Электронный ресурс] // Habr. URL: https://habr.com/ru/articles/676642/