Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р техн. наук, профессор
_____ С.П. Сущенко
«06» июня 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ В ЖАНРЕ 3D ПЛАТФОРМЕР

по направлению подготовки 09.03.03 Прикладная информатика,
направленность (профиль) «Прикладная информатика»

Левкевич Владислав Игоревич

Руководитель ВКР
канд. техн. наук
_____А.В. Приступа
«02» июня 2022 г.

Автор работы
студент группы № 931803
_____В.И. Левкевич
«02» июня 2022 г.

Томск – 2022

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

УТВЕРЖДАЮ
Руководитель ООП
д-р техн. наук, профессор
_____ С.П. Сущенко
подпись
«11» ноября 2021 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра обучающемуся
Левкевичу Владиславу Игоревичу
Фамилия Имя Отчество обучающегося
по направлению подготовки 09.03.03 Прикладная информатика, направленность (профиль) «Прикладная информатика»

1 Тема выпускной квалификационной работы
Разработка компьютерной игры в жанре 3D платформер

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис / деканат – _____ б) в ГЭК – _____

3 Исходные данные к работе:
Объект исследования – Компьютерные игры
Предмет исследования – Разработка компьютерных игр в жанре 3D платформер
Цель исследования – Разработать компьютерную игру в жанре 3D платформер

Задачи:
изучить средства для разработки;
спроектировать механику передвижений;
спроектировать боевую систему и специальные умения персонажей;
спроектировать противников, их поведение и их особые умения;
спроектировать диалоговую систему;
реализовать приложение.

Методы исследования:
экспериментальный на ЭВМ

Организация или отрасль, по тематике которой выполняется работа, –
Томский государственный университет

4 Краткое содержание работы
В данной работе подробно описываются этапы разработки компьютерной игры в жанре 3D платформер на игровом движке Unreal Engine 4

Руководитель выпускной квалификационной работы
Приступа Андрей Викторович _____ / _____
доцент кафедры ТОИ *подпись* *А.В. Приступа*

Задание принял к исполнению
Левкевич Владислав Игоревич _____ / _____
Студент группы 931803 НИ ТГУ *подпись* *В.И. Левкевич*

АННОТАЦИЯ

Выпускная квалификационная работа, 44 страниц, 19 рисунков, 11 источников.

Ключевые слова: 3D платформер, Unreal Engine 4, Blueprints, Behavior Tree.

Цель работы: разработать компьютерную игру в жанре 3D платформер на игровом движке Unreal Engine 4.

Результаты работы: Спроектировано игровое приложение и разработан прототип, состоящий из 2 готовых уровней.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

Боевая система — часть правил игры, отвечающая за столкновения игрока с противниками.

QTE — динамичная сцена в игре, в которой игроку предлагается нажать комбинацию клавиш в определённом порядке или с определённой частотой, чтобы завершить её.

Dash — способность игрока, позволяющая совершать рывок

Projectile — объект, который используется для реализации дальнобойной атаки

Gravity up — способность, которая позволяет менять гравитацию.

Push — способность, позволяющая использовать толчок силовым полем

Уровень — локации на которых происходят действия игры.

Спаун — инициализация объекта в определённом месте на уровне.

НПС — неигровой персонаж.

Стан — состояние персонажа, при котором он недееспособен.

ВВЕДЕНИЕ

Бурное развитие технологий XXI века принесло человечеству огромное количество плодов. В настоящее время тяжело представить жизнь без компьютерной техники. Она проникла во все сферы: экономика, медицина, тяжёлая и лёгкая промышленность и т.д. Сфера отдыха и развлечений не стала исключением. Компьютерные игры тесно вплелись в жизни подрастающего поколения и не только. С учётом сегодняшнего развития технологий крупные компании могут позволить себе создавать игры высочайшего уровня в самых различных жанрах. Одними из самых распространённых являются игры в жанре платформер.

Платформер – это жанр компьютерных игр, в котором основной чертой игрового процесса являются прыжки по платформам, стенам и другим объектам окружения.

Платформеры бывают трех поджанров: двумерные платформеры(2D), трехмерные платформеры(3D) и 2,5D-платформеры. В рамках дипломной работы разрабатывалась игра в жанре 3D платформер. Главным отличием является включение всех трёх измерений для передвижения или использование трёхмерных полигонов в реальном времени для отрисовки уровней и героев. Яркими примерами 3D платформеров являются такие игры, как Little Nighmare [1], Inside [2], Pole [3], Crash Bandicoot [4].

Противники, как правило, разнородные, обладают искусственным интеллектом, стремясь максимально приблизиться к игроку, либо не обладают им вовсе, перемещаясь по круговой дистанции или совершая повторяющиеся действия. Соприкосновение с противником обычно отнимает жизненные силы у героя или вовсе убивает его. Уровни, как правило, изобилуют секретами (скрытые проходы в стенах, высокие или труднодоступные места), нахождение которых существенно облегчает прохождение и подогревает интерес игрока.

При создании игры особое внимание было обращено на игру Little

Nighmare. Она является одной из лучших в своём жанре, а сюжет и квестовая составляющая заставляют играть в неё снова и снова.

Little Nightmares (в переводе с англ. «Маленькие кошмары») — мультиплатформенная компьютерная игра в жанре платформера с элементами квеста и хоррора, разработанная шведской компанией Tarsier Studios и выпущенная компанией Bandai Namco Entertainment. Сюжет повествует о 9-ти летней девочке, внезапно проснувшейся на подводном корабле. Её целью является понять, что происходит вокруг, как она тут очутилась, и выжить, ведь весь корабль наполненным различного рода монстрами.

Управление персонажем тактильное. Нажатие, удерживание и опускание каждой кнопки означает определённое действие, совершаемое героиней. Набор действий, который она способна выполнять, невелик: лазить, прыгать, хватать и использовать предметы, использовать зажигалку, принимать пищу. [5].

Отличительной особенностью разрабатываемой в рамках ВКР игры является её нестандартный набор механик для подобного жанра.

Цель работы: разработать компьютерную игру в жанре 3D платформер на игровом движке Unreal Engine 4.

Для достижения цели необходимо решить следующие **задачи**:

1. спроектировать механику передвижений;
2. изучить средства для разработки;
3. спроектировать боевую систему и специальные умения персонажей;
4. спроектировать противников и их особые умения;
5. спроектировать поведение противников;
6. спроектировать диалоговую систему;
7. реализовать приложение.

1 Инструменты для разработки приложения

1.1 Unreal Engine 4

Для разработки игры выбран движок Unreal Engine 4, т.к. он имеет большее количество бесплатных материалов и ассетов, а также внутри него можно вручную создавать все необходимые модели.

Unreal Engine 4 — игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Написанный на языке C++, движок позволяет создавать игры для большинства операционных систем и платформ, таких как Microsoft Windows, PlayStation, GameCube и др., а также на различных портативных устройствах, например, устройствах Apple (iPad, iPhone), управляемых системой iOS и прочих. [6].

В Unreal Engine 4 используется три основных объекта — это Actor, Pawn и Character.

Actor — любой объект, который может быть расположен на уровне. Он имеет несколько базовых параметров:

1. Transform — данный параметр описывает локацию, поворот и размер объекта при его инициализации на уровне.
2. Mesh — модель объекта.
3. Material — параметр того, как будет выглядеть Mesh, т.е. его раскраска, цвет, текстура.
4. Collision — является параметром, отвечающим за столкновения с другими объектами, обработку триггеров и обработку ударов.

Pawn — наследуется от Actor. Главным отличием является то, что игрок может управлять этим объектом через системы ввода, а также

искусственный интеллект с помощью контроллера.

Character — использует не обычный Mesh, а скелетированный. Что означает, что он может иметь собственный скелет для анимаций.

Также почти каждый объект движка имеет ряд компонентов:

1. Capsule Component, данный компонент используется для настройки базовой коллизии.
2. Arrow component показывает, в какую сторону повернут персонаж: это важно при спауне персонажа и передвижении, чтобы определять в какую сторону двигаться.
3. Character Movment Component описывает базовую логику передвижения персонажа.

Для всех персонажей, в том числе противников и подконтрольного персонажа используется тип Character. Так как они имеют скелет и должны обладать анимациями [7].

1.2 Microsoft Visual Studio

Программирование велось на языке программирования C++ в среде разработки Visual Studio. Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight [8].

2 Проектирование

2.1 Концепция и сюжет

Игра представляет собой короткую историю, рассказывающую о том, как человек переживает горе от утраты родного человека. За основу взята система принятия горя российского психолога Ф.Е. Василюка [9]. Она включает в себя 5 стадий:

1. Шок и оцепенение
2. Фаза поиска
3. Фаза острого горя
4. Фаза “остаточных толчков и реорганизации”
5. Фаза завершения

Каждый уровень игры воплощает одну из этих стадий. Прохождение каждой локации знаменует преодоление очередной фазы. Все противники, с которыми герою предстоит столкнуться, показывают его искажённое видение мира. Например, коллег по работе он представляет в виде бесформенных и обезличенных чудовищ, представленных на рисунке 1.

Рисунок 1 − концепт-арт противника-коллеги

По ходу развития сюжета герой будет сталкиваться с противниками, атаки которых переносят в локацию под названием “Тёмный мир”. Данный мир является олицетворением всех его негативных переживаний. Во время

пребывания в тёмном мире герою необходимо собрать пазл из трёх частей, который образует фрагмент травмирующих воспоминаний из прошлого. Далее во время заключительной фазы эти фрагменты сыграют важную роль.

Таким образом, пройдя все испытания, главный герой принимает произошедшее и продолжает свой жизненный путь.

2.2 Базовый противник

Базовый противник – это монстр, который будет атаковать игрока, как только увидит. Этот противник разрабатывается с целью создания основных состояний, которые потом смогут использовать другие типы противников. В ходе разработки создано 4 типа подобных противников. На рисунке 2 представлена диаграмма состояний одного из типов базового противника, атакующего игрока кислотой на расстоянии.

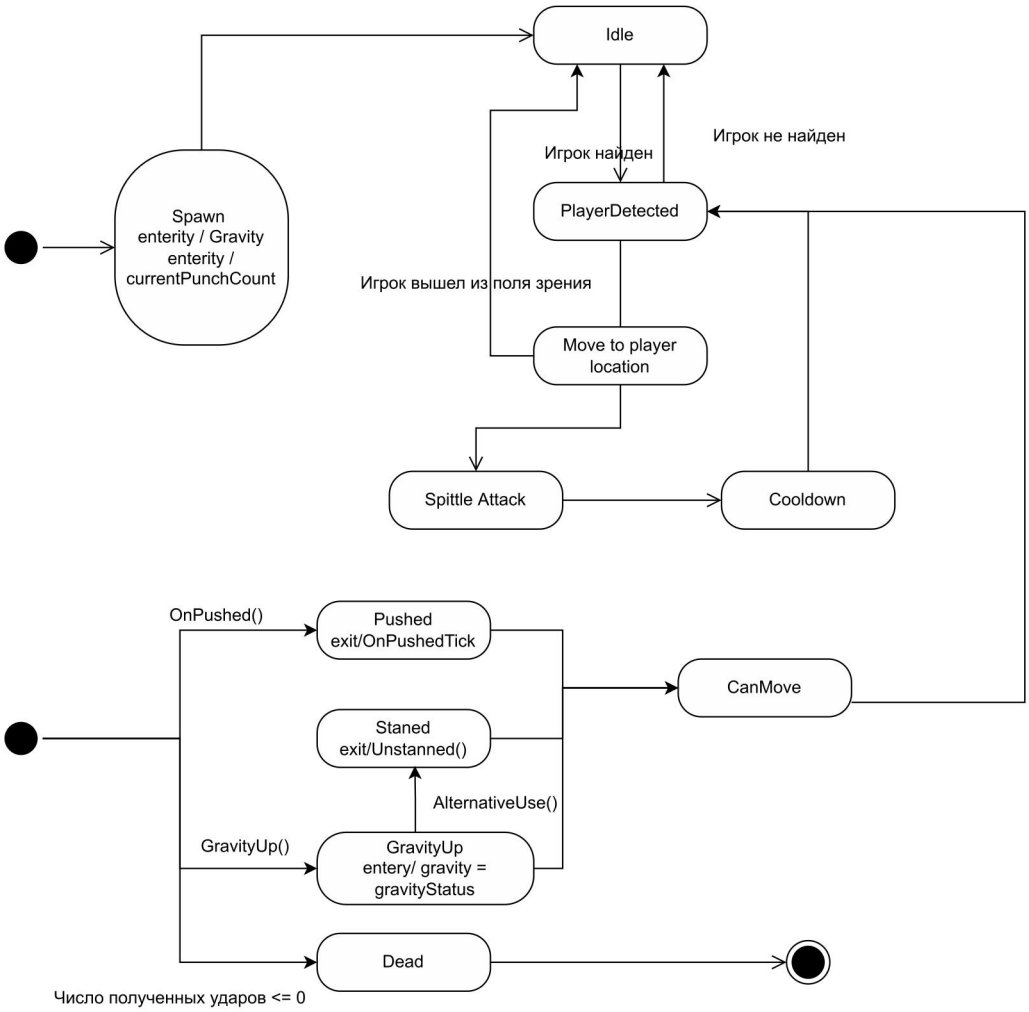

Рисунок 2 − Диаграмма состояний базового противника

Ниже приведено описание всех состояний диаграммы.

1. **Spawn**. На входе противник попадает именно в это состояние. В этом состоянии происходят первоначальные настройки при загрузке уровня. Как только настройки завершены, противник переходит в состояние Idle и стоит на месте в ожидании персонажа, подконтрольного игроку.

2. **Idle**. В этом состоянии противник будет стоять на месте до тех пор, пока в его поле зрения не попадёт игрок. Если игрок попадает в его поле зрения, то происходит переход в состояние PlayerDetected.

3. **PlayerDetected**. Состояние, в котором противник подаёт знак того, что обнаружил игрока. Он запоминает его местоположение и расстояние до игрока, после чего сразу переходит в состояние Move to player location.

4. **Move to player location.** Состояние, в котором противник движется к игроку до того расстояния, для которого он имеет возможность наносить удары. И как только он подбирается на такое расстояние, то переходит в состояние Spittle Attack.

5. **Spittle Attack**. Состояние, в котором противник будет атаковать игрока с расстояния кислотой. После атаки противник переходит в состояние Cooldown.

6. **Cooldown**. Состояние, в котором противник находится в ожидании для совершения следующего удара. Это состояние необходимо для того, чтобы атаки противника не были беспрерывными. Как только время прошло, то противник снова переходит в состояние PlayerDetected, и действия зацикливаются до тех пор, пока противник не победит игрока или же не будет убит.

7. **Pushed**. Состояние, в которое противник может войти при условии, что игрок использовал способность Push. В этом состоянии противник теряет равновесие, отталкивается в противоположную от игрока сторону и теряет возможность двигаться на некоторое время. После чего переходит в

состояние CanMove.

8. **GravityUp.** Состояние, в которое противник может войти при условии, что игрок использовал способность GravityUp. В этом состоянии гравитация противника отключается, и на некоторое время его тело поднимается в воздух и не имеет возможности взаимодействия с чем-либо.

a. Если игрок применил альтернативное использование способности GravityUp, то противник с силой бьется о землю и переходит в состояние Staned.

b. Если игрок не применил альтернативное использование способности GravityUp, то через некоторое время тело противника опустится на землю и переходит в состояние CanMove.

9. **Staned.** В этом состоянии противник лежит на земле и не имеет возможности двигаться и взаимодействовать с игроком некоторое время. После истечении времени противник переходит в состояние CanMove.

10. **CanMove.** Состояние, в котором противник проверяет, вышли ли таймеры, которые запрещали ему взаимодействовать с игроком. Если вышли, то он переходит в состояние PlayerDetected.

11. **Dead**. Состояние, в которое противник может войти из любой стадии при условии, что он получил урон, и количество возможных полученных ударов опустилось до нуля. Из этого состояния противник переходит в конечное состояние.

2.3 Смерть с косой

Этот тип противников будет стараться подходить к игроку как можно ближе. При сближении с игроком, он попытается заскочить на него и начать пожирать. Смерть будет использовать множество состояний, описанных у базового противника, но также привнесёт и специфические ему. На рисунке 3 представлена диаграмма состояний для смерти с косой.

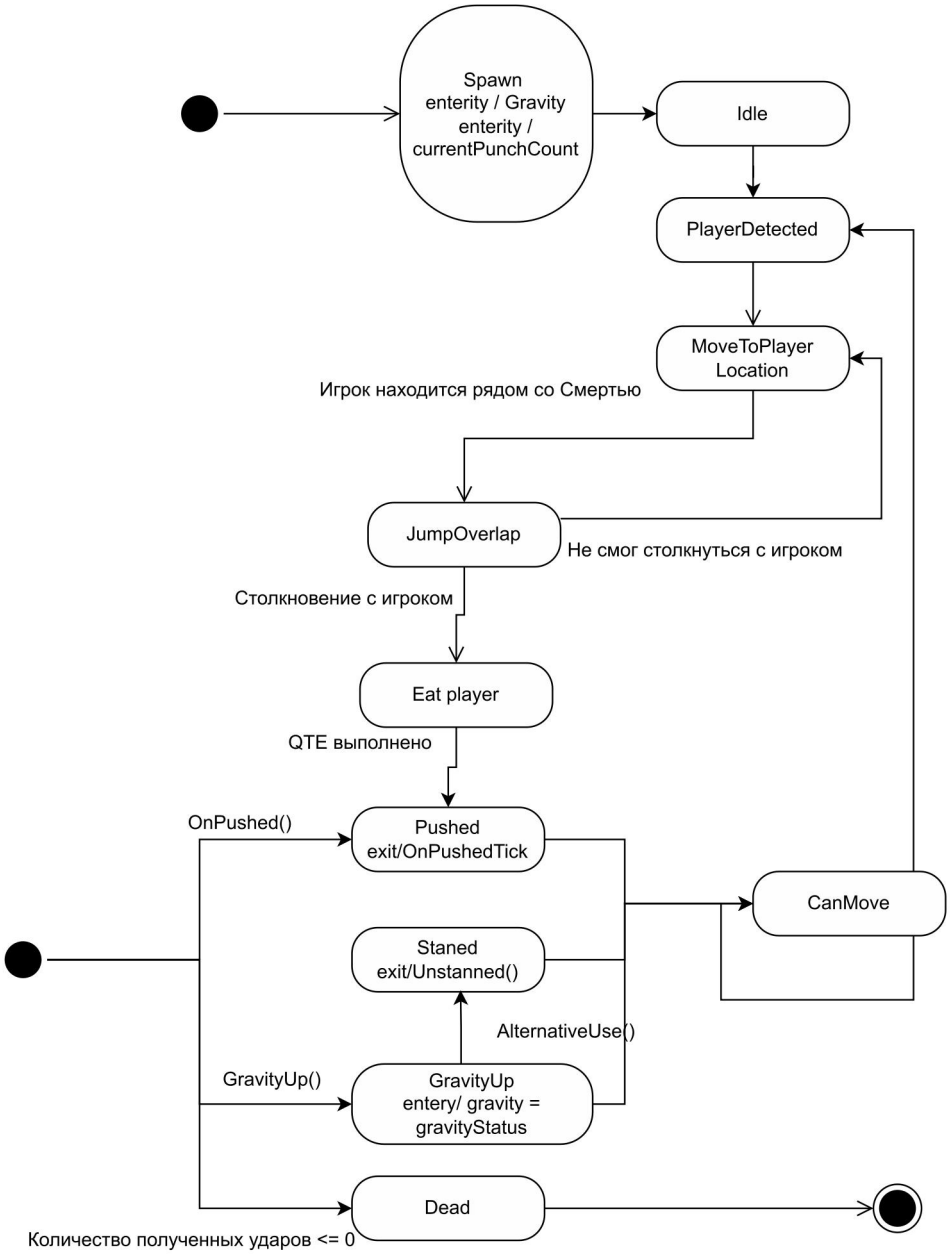

Рисунок 3 − Диаграмма состояний противника “Смерть с косой”

Ниже приведено описание всех состояний диаграммы.

1. **Spawn**. На входе противник попадает именно в это состояние. В этом состоянии происходят первоначальные настройки при загрузке уровня. Как только настройки были завершены, противник переходит в состояние Idle и стоит на месте в ожидании персонажа подконтрольного игроку.

2. **Idle**. В этом состоянии противник будет стоять на месте до тех пор, пока в его поле зрения не попадёт игрок. Если игрок попадает в его поле зрения, то происходит переход в состояние PlayerDetected.

3. **PlayerDetected**. Состояние, в котором противник подаёт знак того, что обнаружил игрока. Он запоминает его местоположение и расстояние до игрока, после чего сразу переходит в состояние Move to player location.

4. **Move to player location.** Состояние, в котором противник движется к игроку до того расстояния, для которого он имеет возможность наносить удары. И как только он подбирается на такое расстояние, то переходит в состояние Spittle Attack

5. **JumpOverlap**. Состояние, в котором противник будет прыгать на игрока и пытаться закрепиться на его теле.

a. Если ему удалось соприкоснуться с игроком, то противник переходит в состояние Eat player.

b. Если ему не удалось и он не соприкоснулся с игроком, то противник переходит в состояние Move to player location.

6. **Eat player**. Состояние, в котором противник находится на теле игрока и наносит ему урон.

a. Если игрок успешно прошел QTE, в таком случае противник переходит в состояние Pushed.

b. Если игрок не сможет выполнить QTE, то он умрёт, а противник спустится с него и перейдет в состояние Idle.

7. **Pushed**. Состояние, в которое противник может войти при условии, что игрок использовал способность Push. В этом состоянии противник теряет равновесие, отталкивается в противоположную от игрока сторону и теряет возможность двигаться на некоторое время. После чего переходит в состояние CanMove.

8. **GravityUp.** Состояние, в которое противник может войти при условии, что игрок использовал способность GravityUp. В этом состоянии гравитация противника отключается и на некоторое время его тело поднимается в воздух и не имеет возможности взаимодействия с чем-либо.

a. Если игрок применил альтернативное использование способности

GravityUp, то противник с силой бьется о землю и переходит в состояние Staned.

b. Если игрок не применил альтернативное использование способности GravityUp, то через некоторое время тело противника опустится на землю, и произойдёт переход в состояние CanMove.

9. Staned. В этом состоянии противник лежит на земле и не имеет возможности двигаться и взаимодействовать с игроком некоторое время. По истечении времени противник переходит в состояние CanMove.

10. CanMove. Состояние, в котором противник проверяет, вышли ли таймеры, которые запрещали ему взаимодействовать с игроком. Если вышли, то он переходит в состояние PlayerDetected.

11. **Dead**. Состояние, в которое противник может войти из любой стадии при условии, что он получил урон, и количество возможных полученных ударов опустилось до нуля. Из этого состояния противник переходит в конечное состояние.

3 Реализация

В этой главе описано создание игровых механик в среде Unreal Engine 4 с использованием её компонентов и классов.

Далее речь пойдёт о трёх основных аспектах, которые необходимо реализовать:

- подконтрольный персонаж;
- противники;
- диалоги.

3.1 Персонаж

Персонаж – объект, которым управляет игрок. В соответствии с нажатыми игроком клавишами персонаж способен осуществлять следующие действия:

- перемещение по всем осям координат;
- прыжок;
- обычная серия атак;
- атаки с помощью оружия;
- рывок;
- способность, изменяющая гравитацию;
- толчок силовым полем.

У персонажа есть три основные характеристики:

- здоровье;
- прогресс смерти - отвечает за QTE в борьбе со Смертью с косой;
- страх. Страх убывает во время битвы с боссом. При значении

меньше 20%, управление игрока инвертируется. В случае, если у игрока значение страха ровно 0, то он теряет управление над персонажем.

На рисунке 4 представлена диаграмма классов, связанных с игроком.

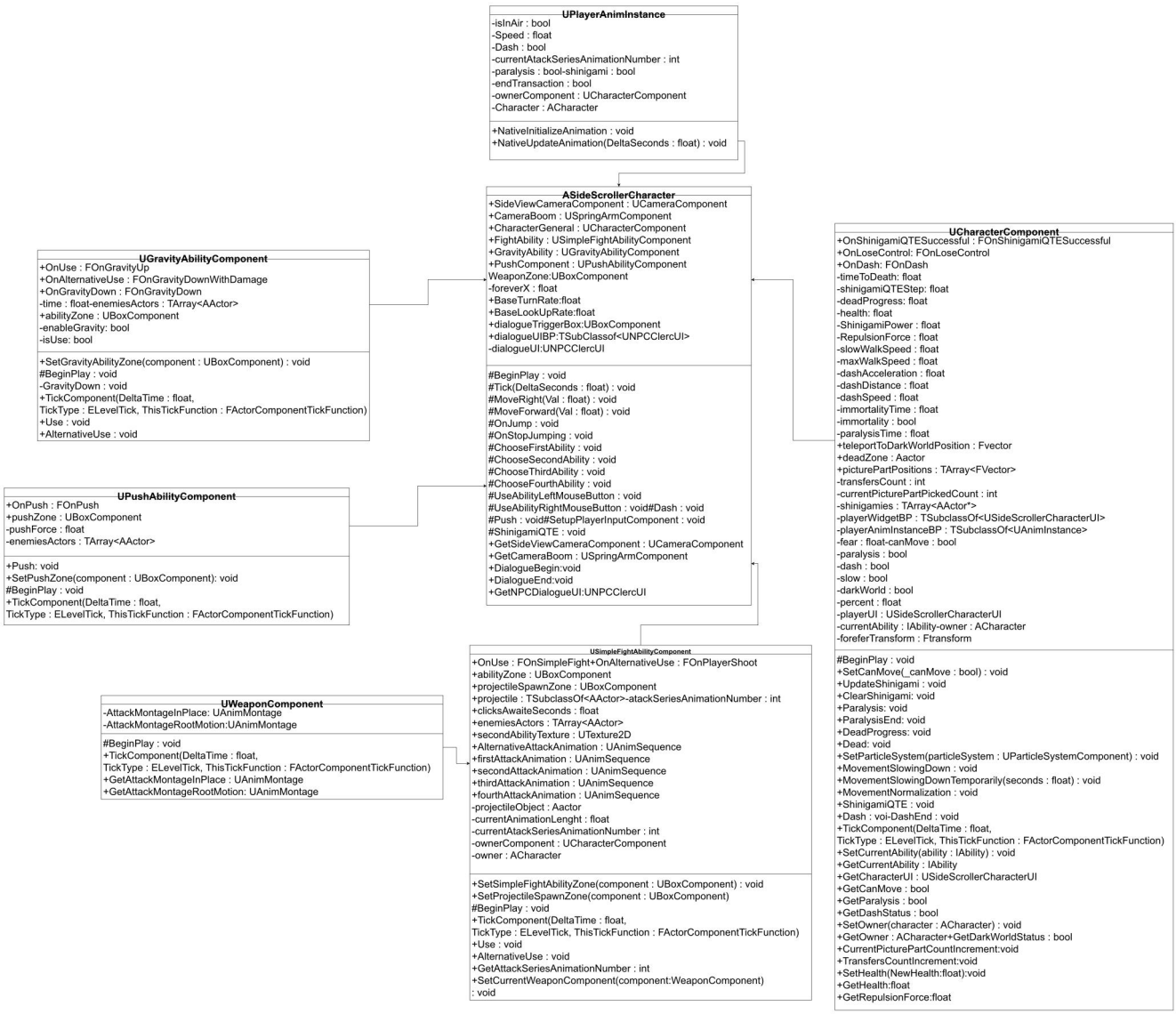

Рисунок 4 − Диаграмма классов игрока

Наш персонаж, кроме базовых компонентов, имеет свои собственные. В их число входит три коллизионных коробки (BoxCollision). Первая отвечает за зону, в которой можно наносить урон по противникам с помощью обычных ударов. Вторая — это зона, в которой материализуются Projectile. Третья определяет зону, в которой противники подвержены способности управления гравитацией. Также персонаж использует систему частиц для визуального представления способности «Рывок». Данные коллизионные коробки продемонстрированы на рисунке 5.

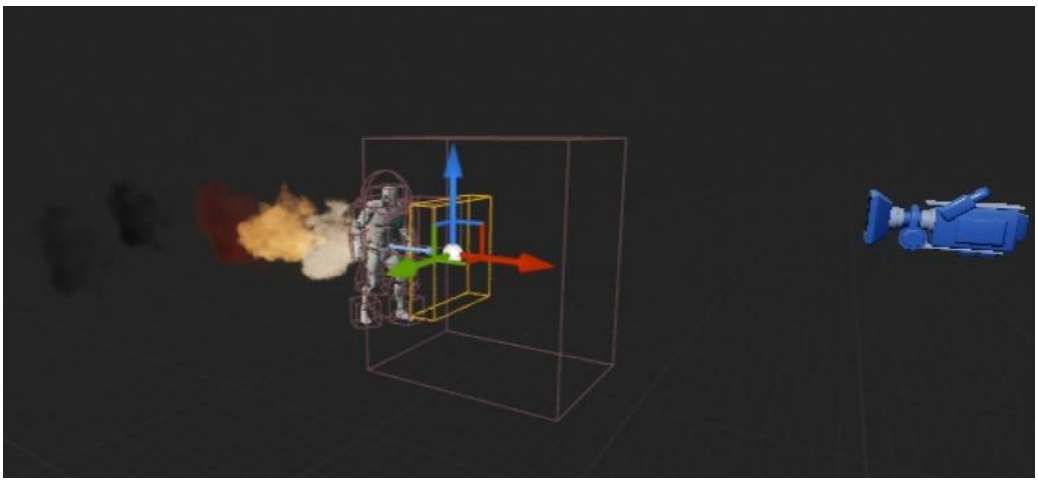

Рисунок 5 − BoxCollision для модели игрока

3.1.1 Управление

За управление передвижением персонажа отвечает встроенный в движок компонент MovementComponent. Значения того, в какую сторону движется персонаж, взимается из класса ASideScrollerCharacter и находится в методе MoveRight() и MoveForward(). При нажатии клавиши W на вход системе подастся положительное значение, и персонаж продвинется вперёд. При нажатии клавиши S на вход подастся отрицательное значение, и персонаж будет двигаться назад. Движения вправо и влево осуществляются похожим образом. Назначение клавиш определяется в методе SetupPlayerInput().

3.1.2 Боевая система

За управление боевой системой отвечает класс USimpleFightUbilityComponent.

Перечислим методы класса USimpleFightUbilityComponent:

1. **BeginPlay()** – метод, вызывающийся при инициализации игры.
2. **TickComponent()** – метод, который вызывается каждый фрейм. В нём проверяются, какие были нажаты кнопки, возможно ли атаковать, находится ли персонаж в дееспособном состоянии.
3. **Use()** – метод, отвечающий за использование серии обычных ударов.

Серия состоит из 4 ударов, при каждом нажатии кнопки ЛКМ счётчик стартует с 0 и когда он достигает значения 4, то после этого серия начинается заново. При каждом значении счетчика проигрывается определённая анимация.

4. **AlternativeUse()** – метод, отвечающий за дистанционную атаку. Создается projectile и направляется в сторону противника из соответствующего BoxCollision.
5. **SetSimpleFightAbilityZone()** – метод, в котором определяется зона нанесения обычных ударов. Вызывается и работает с помощью Blueprints.
6. **SetProjectileSpawnZone()** – метод, в котором определяется зона создания projectile. Вызывается и работает с помощью Blueprints.
7. **GetAttackSeriesAnimationNumber()** – метод получения текущего номера серии ударов.
8. **SetCurrentWeaponComponent()** – метод, в котором назначается текущее оружие.

3.1.3 Оружие

За использование и хранение информации об оружии отвечает класс UWeaponComponent.

Перечислим методы класса UWeaponComponent:

1. **BeginPlay()** – метод, вызывающийся при инициализации игры.
2. **TickComponent()** – метод, который вызывается каждый фрейм. В нём проверяются, какие были нажаты кнопки, возможно ли атаковать, находится ли персонаж в дееспособном состоянии.
3. **GetAttackMontageInPlace()** – метод, отвечающий за особую анимацию атаки с оружием, когда персонаж стоит на месте
4. **GetAttackMontageRootMotion()** – метод, отвечающий за особую анимацию атаки с оружием, когда персонаж находится в движении.

3.1.4 Толчок силовым полем

За использование персонажем способности «Силовой толчок» отвечает класс UPushAbilityComponent.

Перечислим методы класса UPushAbilityComponent:

1. **BeginPlay()** – метод, вызывающийся при инициализации игры.
2. **TickComponent()** – метод, который вызывается каждый фрейм. В нём проверяются, какие были нажаты кнопки, возможно ли атаковать, находится ли персонаж в дееспособном состоянии.
3. **Push()** – метод, который отталкивает от игрока всех, кто попал в зону действия способности.
4. **SetPushZone() -** метод, в котором определяется зона действия способности. Вызывается и работает с помощью Blueprints, изображённого на рисунке 6.

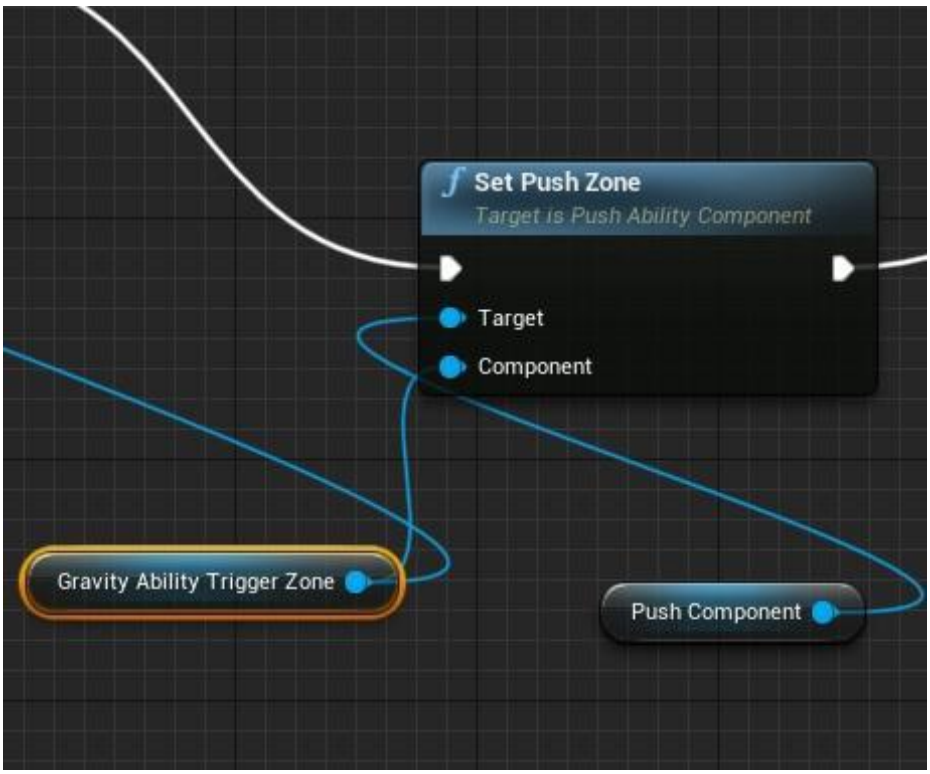

Рисунок 6 − Blueprints метода SetPushZone

Перечислим значения нодов:

1. Gravity Ability Trigger Zone – определяет зону действия BoxCollision для способности.
2. Push Component – ссылка на компонент PushAbilityComponent.

3.1.5 Способность управления гравитацией

За способность управления гравитацией отвечает класс UGravityUpComponent.

Перечислим методы класса UGravityUpComponent:

1. **BeginPlay()** – метод, вызывающийся при инициализации игры.

2. **TickComponent()** – метод, который вызывается каждый фрейм. В нём проверяются, какие были нажаты кнопки, возможно ли атаковать, находится ли персонаж в дееспособном состоянии.

3. **Use()** – метод, отвечающий за использование способности

управления гравитацией. Он отключает гравитацию противника и поднимает его в воздух.

4. **AlternativeUse()** – метод, отвечающий за включения гравитации, при его использовании противник бьется о землю и получает урон.

5. **GravityDown() -** метод, который вызывается в Use() и через некоторое время возвращает гравитацию противнику без нанесения урона.

6. **SetGravityAbilityZone** - метод, в котором определяется зона действия способности. Вызывается и работает с помощью Blueprints.

3.1.6 Анимация персонажа

Анимации для персонажа задаются через класс UPlayerAnimInstance и передаются объекту персонажа.

UPlayerAnimInstance позволяет организовать и поддерживать набор анимаций и ассоциативных транзакций для персонажа или объекта. На рисунке 7 представлено окно аниматора Unreal Engine 4 с диаграммой состояний с переходами для персонажа.

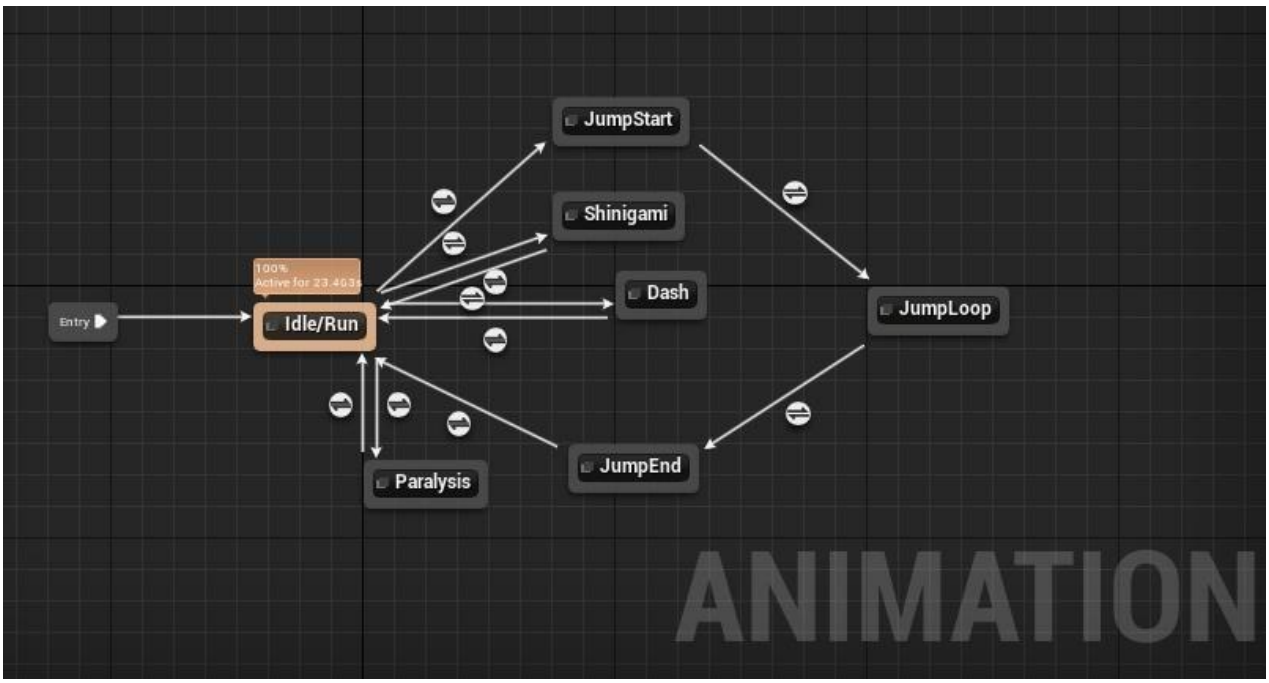

Рисунок 7 − Диаграмма состояний с переходами
для персонажа

В данной схеме представлены 7 возможных состояний анимации персонажа и 10 правил перехода между ними. Перечислим эти состояния:

1. **Idle/Run** – состояние персонажа, в которое он попадает сразу после точки входа Entry. В зависимости от скорости передвижения, он может принимать следующие анимации:
   a. Стоит на месте - скорость 0;
   b. Пешая ходьба - скорость 90;
   c. Бег - скорость 375.

Значения скорости берется в методе NativeUpdateAnimation класса UPlayerAnimInstance.

2. **JumpStart** – состояние, в которое можно попасть из Idle/Run, при условии, что персонаж находится в воздухе. В данном состоянии проигрывается анимация подпрыгивания. Проверка проводится в методе NativeUpdateAnimation класса UPlayerAnimInstance.
3. **JumpLoop** – состояние, в которое можно попасть из JumpStart, при условии, что предыдущая анимация завершена на 90%. В данном

состоянии проигрывается анимация парения в воздухе.

4. **JumpEnd** – состояние, в которое можно попасть из JumpLoop, при условии, что персонаж встал на землю и не находится в воздухе. Если анимация JumpEnd завершена более, чем на 90%, то происходит переход в Idle/Run. В данном состоянии проигрывается анимация приземления.

5. **Dash** – состояние, в которое можно попасть из Idle/Run. В данном состоянии проигрывается анимация способности «Рывок» при условии, что игрок использовал данную способность.

6. **Shinigami** – состояние, в которое можно попасть из Idle/Run, при условии, что меш противника «Смерть с косой» соприкоснулось с мешем подконтрольного персонажа. В данном состоянии подконтрольный персонаж лишён возможности передвигаться и постепенно теряет очки здоровья. Для того чтобы выйти из этого состояния игрок должен пройти QTE.

7. **Paralysis** – состояние, в которое попадают, если персонаж парализован. В данном состоянии подконтрольный персонаж не имеет возможности двигаться определённое время.

3.1.7 Базовые механики персонажа

За настройку и работу базовых механик отвечает класс UCharacterComponent. Перечислим методы этого класса:

1. **BeginPlay()** - метод, в котором создаётся интерфейс для персонажа.

2. **TickComponent()** - метод, в котором происходит покадровое обновление полосок страха, здоровья и смерти. В случае, если персонаж не может двигаться, то здесь накладывается этот запрет.

3. **QTESuccecfull()** - метод, отвечающий за QTE в борьбе со Смертью с косой. В случае успеха все противники отлетают в противоположное от игрока расстояние.

4. **SetCanMove()** - метод, в котором задаётся статус перемещения персонажа.

5. **UpdateShinigami()** - метод, отслеживающий нападение и соприкосновение тела Смерти с косой с телом подконтрольного игроку персонажа.

6. **ClearShinigami()** - метод, который отчищает список прикреплённых Смертей с косой.

7. **Paralysis()** - метод, отвечающий за паралич персонажа. Назначается таймер и по его окончании вызывается метод ParalysisEnd().

8. **ParalisysEnd()** - метод, отвечающий за окончание паралича.

9. **DeadProgress()** - метод, отвечающий за уменьшение значения здоровья в секунду. По формуле - health = health - (100.f - (timeToDeath - (shinigamies.Num() * ShinigamiPower)) / percent);

a) health - текущий прогресс смерти;

b) timeToDeath - время по окончании которого персонаж погибнет;

c) shinigamies.Num() - количество противников;

d) ShinigamiPower - сила противников;

e) percent - время, через которое персонаж погибнет / 100;

В этом же методе проверяется количество здоровья персонажа, и если оно ровно 0, то вызывается метод Dead().

10. **Dead()** - метод, гибели персонажа. Перезапускает уровень.

11. **MovementSlowingDown()** - метод, отвечающий за снижение скорости персонажа.

12. **MovementSlowingDownTemporary()** - метод, отвечающий за снижение скорости персонажа на определённое время.

13. **MovementNormalization()** - метод, возвращающий скорость в норму.

14. **ShinigamiQTE** - метод, отвечающий за QTE составляющую во время сражения с противником “Смерть с косой”.

15. **Dash()** - метод, отвечающий за использование способности Рывок. Во время использования запускается назначенная анимация и по её окончании вызывается метод DashEnd().

16. **DashEnd()** - метод, отвечающий за окончание действия способности Рывок.

17. **SetCurrentAbility()** - метод, отвечающий за назначение текущей способности.

18. **GetCurrentAbility()** - метод, отвечающий за получение данной способности.

19. **GetCharacterUI()** - метод, отвечающий за получение интерфейса персонажа.

20. **GetCanMove()** - метод, отвечающий за получение возможности двигаться.

21. **GetParalysis()** - метод, отвечающий за получение состояния парализации.

22. **GetDashStatus()** - метод, отвечающий за получение статуса рывка. В этом состоянии игрок неуязвим.

23. **SetOwner()** - метод, отвечающий за назначение хозяина-персонажа компонента.

24. **GetOwner()** - метод, отвечающий за получение хозяина-персонажа компонента.

25. **GetDarkWorldStatus()** - метод, отвечающий за получение статуса,

в темном мире ли находится персонаж. Значение True - да, значение false - нет

26. **CurrentPicturePartCountInrement()** - в данном методе происходит суммирование полученных частей пазла.

27. **TransferCountIncrement()** - в данном методе происходит проверка на то, какое количество частей пазла собрано.

28. **SetHealth()** - метод, отвечающий за назначение текущего здоровья

29. **GetHealth()** - метод, отвечающий за получение текущего здоровья

30. **GetRepulsionForce()** - метод, отвечающий за получение той силы, с которой отталкивается Смерть с косой.

31. **SetParticleSystem()** - метод, отвечающий за назначение системы частиц. Работает с помощью Blueprints.

Как можно было заметить, в разработке персонажа часто использовалась система Blueprints. Blueprints — это визуальная, нодовая система программирования, которая используется в Unreal Engine 4. С помощью составления логических блоков нодов, программу можно «собрать» как из конструктора [10].

В данной работе она также использовалась для упрощения работы и связи между кодом и встроенными возможностями движка (рис. 8).

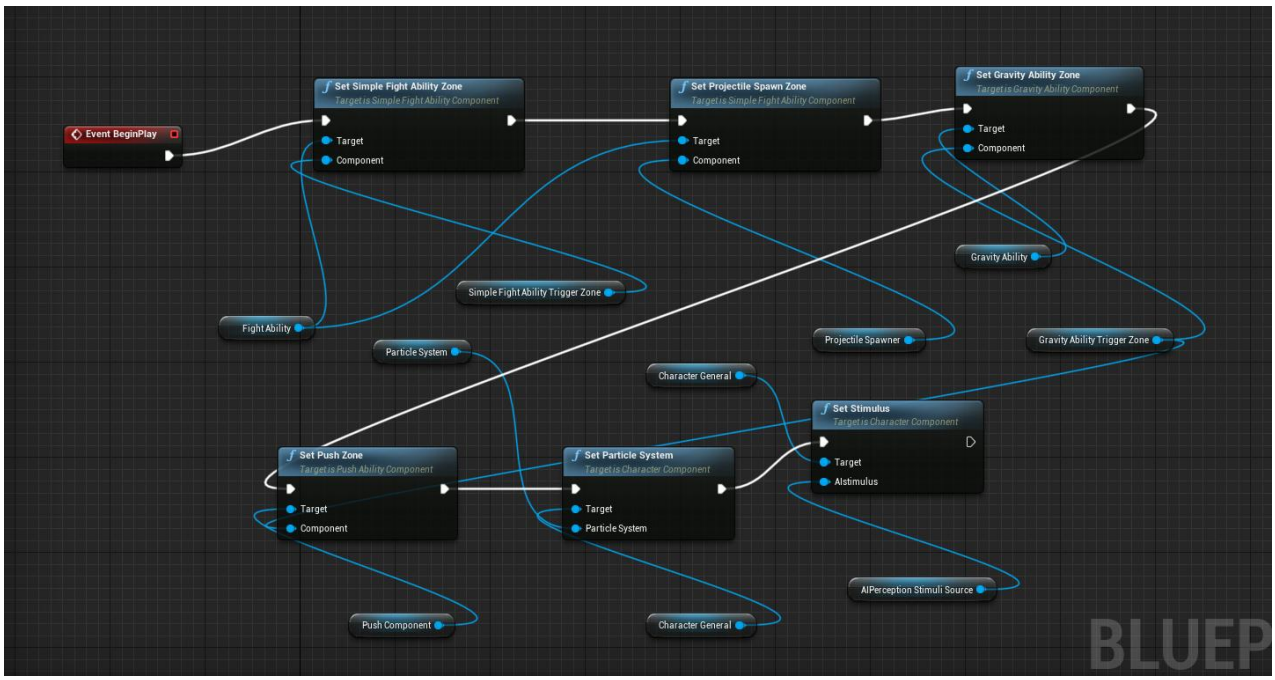

Рисунок 8 − Blueprints зоны действия способностей

В данной схеме обозначены 5 нодов, определяющих зону действия всех способностей и системы частиц.

3.2 Противники

Противник – сущности, обладающие определённым поведением, которые будут мешать игроку при прохождении игры.

3.2.1 Общая структура объекта-противника

На рисунке 9 представлена диаграмма классов базового противника.

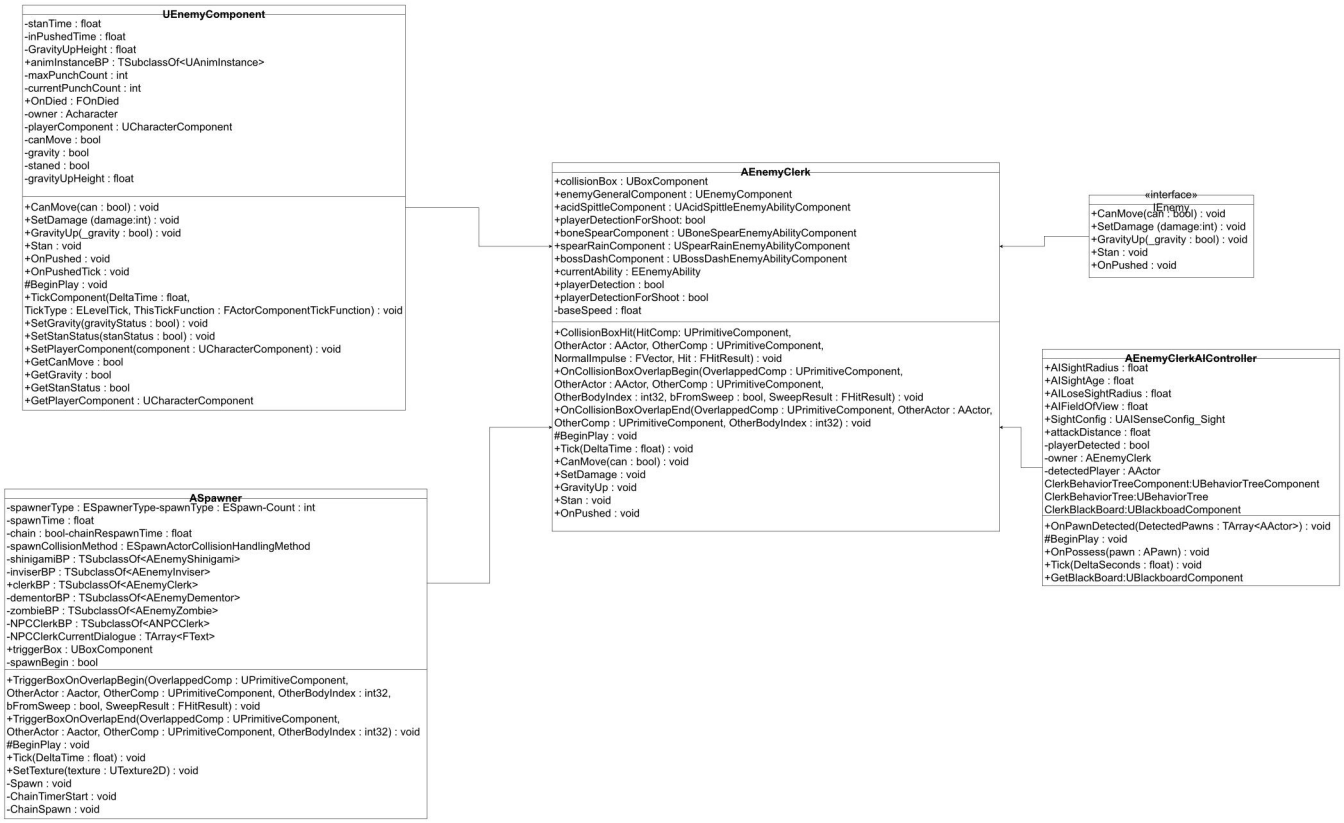

Рисунок 9 − диаграмма классов, связанных с противниками

Поведение противников реализовано с помощью контроллера и древа поведения. При определённых условиях противник должен переходить в следующее состояние, и должна проигрываться соответствующая анимация.

Роль контроллера выполняет класс AEnemyClercController. В его полях создаются и определяются такие параметры как:

1. Расстояние, на котором противник может увидеть игрока;
2. Время, через которое противник перестанет интересоваться игроком;
3. Расстояние, на котором противник отстаёт от игрока;
4. Угол обзора противника;

5. Класс-контейнер для хранения вышеперечисленных параметров.

За поведение персонажа отвечает BehaviorTree (Древо поведения). Behavior Tree – это ориентированный ациклический граф, узлами которого являются возможные варианты поведения ИИ [11].

На рисунках 10 и 11 изображены деревья поведения для двух типов противников. Приведём описание этих деревьев:

1. Selector - запускает своих потомков слева направо. Если один потомок преуспел, то выполнение переходит к следующему до тех пор, пока все не станут успешными.
2. Sequence – то же самое, что и Selector, но в его основу можно заложить некоторые предварительные действия перед запуском следующих нодов. На рисунках изображено действие Cooldown, которое определяет время ожидания между циклами.
3. PlayerDetected AI Task - задание для ИИ: если игрок обнаружен, то запоминается его местоположение.
4. Move to - движение к записанному местоположению игрока.
5. Spittle AI Task - начало специальной атаки кислотой.
6. Bone Spear Up AI Task - начало специальной атаки копьем.

Единственным отличием является тип атаки, которую будет использовать противник.

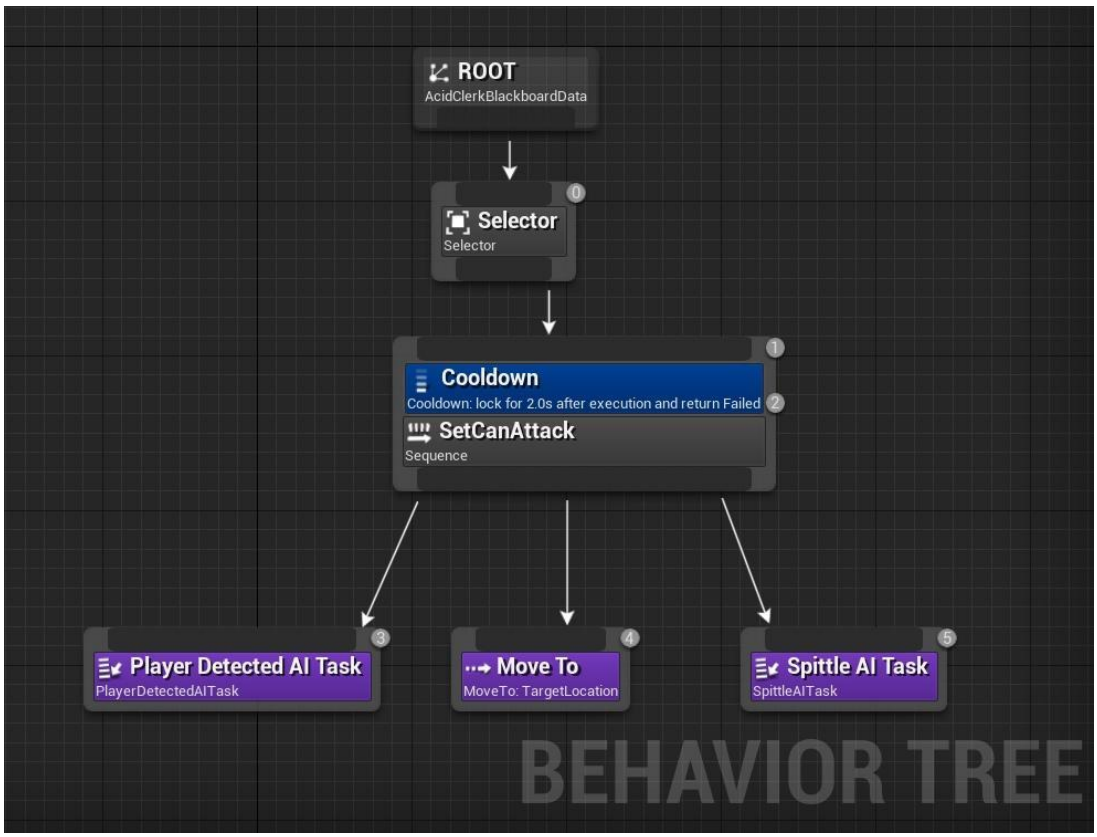

Рисунок 10 − древо поведения противника, атакующего кислотой

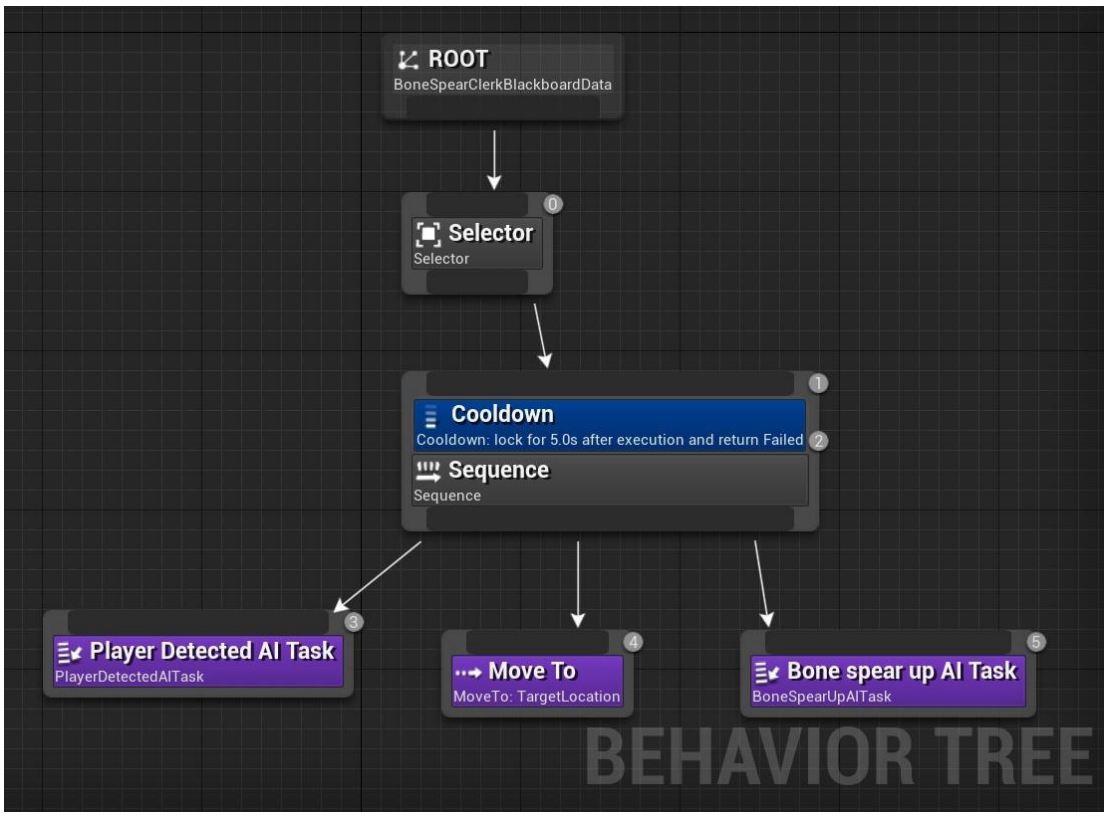

Рисунок 11 − древо поведения противника, атакующего копьями

На рисунке 12 изображено поведения третьего типа противников:

1. Selector - запускает своих потомков слева направо. Если один потомок преуспел, то выполнение переходит к следующему до тех пор, пока все станут успешными;
2. Sequence – то же самое, что и Selector, но в его основу можно заложить некоторые предварительные действия перед запуском следующих нодов. В данном примере они отсутствуют;
3. Spear Rain Task - противник поднимается в воздух и готовиться к атаке;
4. Spear Rain Task - противник начинает атаковать игрока копьями с неба;
5. Spear Rain Stop - спускается на землю;

Wait - ожидание перед следующей атаки.

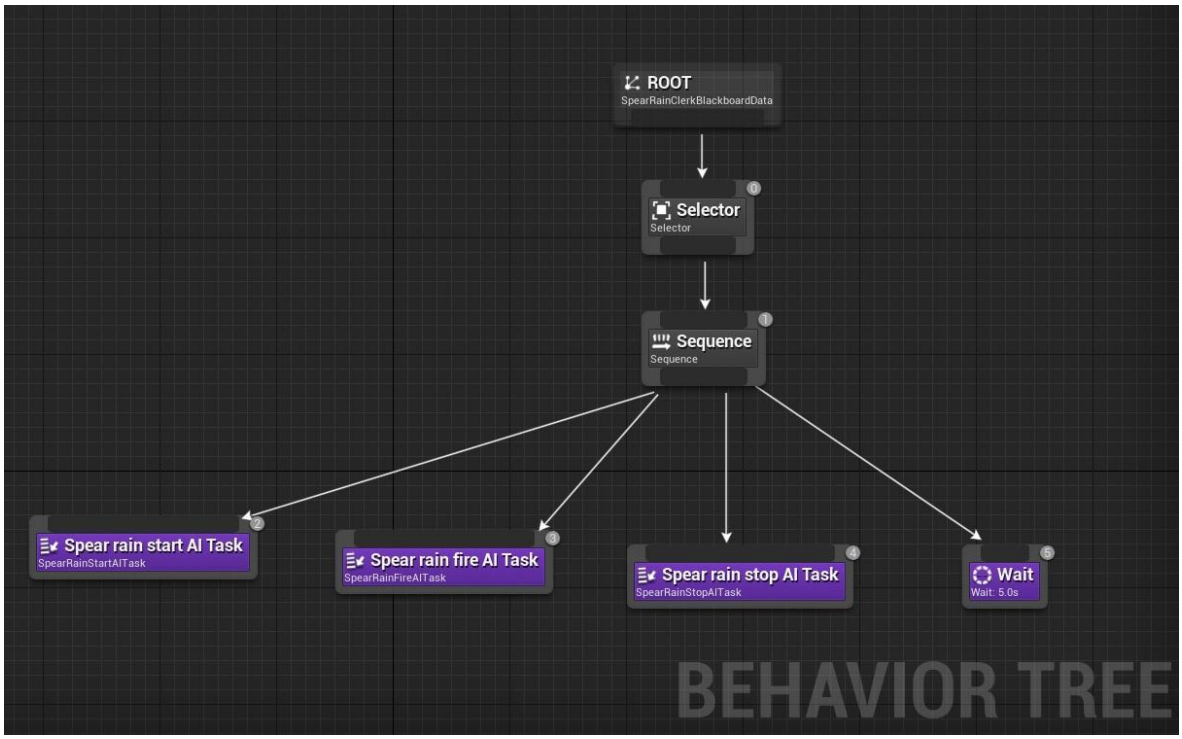

Рисунок 12 − древо поведения противника, призывающего копья с неба

На рисунке 13 изображено поведение четвёртого типа противников:

1. Selector - запускает своих потомков слева направо. Если один потомок преуспел, то выполнение переходит к следующему до тех пор, пока все станут успешными;
2. Sequence – то же самое, что и Selector, но в его основу можно заложить некоторые предварительные действия перед запуском следующих нодов. На

рисунках изображено действие Cooldown, которые определяют время ожидания между циклами;

3. BossDashStart - начало атаки противника;

4. BlackBoardBasedCondition - проверка на то, остановился ли противник или нет;

BossDashStop - окончание атаки.

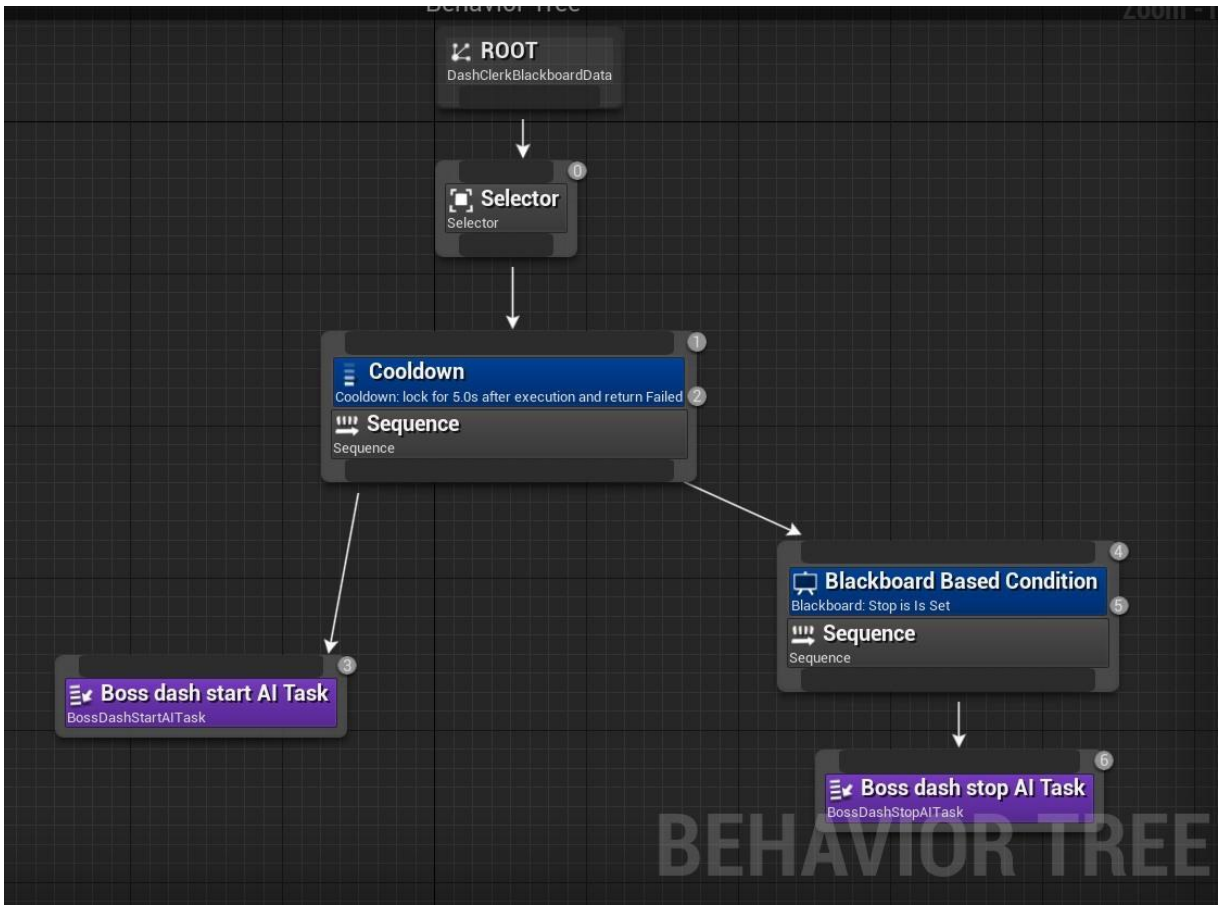

Рисунок 13 − древо поведения противника, использующего рывки для атаки

Противник является базовым и может иметь любое из этих четырёх древ поведения.

Также каждый противник имеет 4 основных составляющих:

1. класс-оболочка;

2. контроллер/класс, который задаёт древа поведения;

3. компонент противника, который описывает базовую логику и характеристики противника;

4. интерфейс, который отвечает за идентификацию противника другими

объектами.

Всем противникам присущи некоторые общие черты. В данном случае у каждой сущности имеется:

- количество ударов, которые может получить противник;
- состояние гравитации;
- количество ударов, которые осталось получить до смерти;
- анимации;
- подверженность способностям управления гравитации и толчку силовым полем.

За реализацию сущности противника отвечает класс UEnemyComponent. Перечислим его основные методы:

1. **CanMove()** – метод, используемый для способности двигаться.
2. **GravityUp()** – метод, описывающий логику отключения гравитации для всех противников.
3. **Stan()** – метод, описывающий для всех противников логику стана.
4. **Unstaned** – метод, описывающий для всех противников логику прекращения стана.
5. **SetDamage()** – метод, описывающий для всех противников логику получения уроня
6. **OnPushed()** – метод, описывающих для всех противников логику поведения на толчок силовым полем.
7. **OnPushedTick()** – метод, описывающий логику того, как противник встаёт после способности «Толчок силовым полем».
8. **BeginPlay()** - метод, в котором создаётся интерфейс для персонажа.
9. **TickComponent()** - метод, в котором происходит покадровая проверка жизненных показателей противника.
10. **SetGravity()** – метод, который принимает значение статуса гравитации.

11. **SetStanStatus()** – метод, который принимает значение статуса стана.
12. **SetPlayerComponent()** – в этом методе передаётся ссылка на игрока.
13. **GetCanMove()** – метод, получения возможности ходить..
14. **GetStanStatus()** – метод, получения статуса стана.
15. **GetCurrentPunchCount()** – метод, который подсчитывает сколько ударов осталось нанести до смерти.
16. **GetPlayerComponent()** – метод, в котором происходит получение компонента игрока.

За то, чтобы объекты могли отличать друг друга, отвечает интерфейс. Задачу интерфейса берет на себя IEnemy. Наличие интерфейса обязательно для каждого противника. Также все, кто имеют интерфейс, обязательно должны обеспечить реализацию 5 методов:

1. CanMove()
2. SetDamage()
3. GravityUp()
4. Stan()
5. OnPushed()

Для того чтобы игрок мог сталкиваться с противниками и сражаться с ними на уровне, необходимо расставлять специальные точки спауна противников. За это отвечает класс ASpawner.

Опишем его методы:

1. **BeginPlay()** - в зависимости от того какой тип спауна указан, данный метод работает по определённому сценарию:

а) Один раз при старте уровня. В таком случает вызывается метод Spawn() один раз при инициализации;

b) По триггеру. Если выбран этот тип спауна, тогда TriggerBoxOnOverLapBegin становится обработчиком события OnComponentBeginOverlap, (т.е когда игрок персонаж соприкасается с triggerBox) и вызывается метод Spawn();

c) По времени - спаун раз в какое-то время. Мы запускаем таймер, каждый период времени будет вызываться метод Spawn().

2. **OnComponentBeginOverlap** - метод события соприкосновения обьекта и triggerBox.

3. **Spawn()** - метод проверки на то, какой типа объекта будет спауниться.

a) Спаун конкретного противника, который был выбран. Противники могут спауниться столько раз, сколько было назначено. Противнику можно назначить спаун по цепи, т.е после смерти одного, через какое-то время спауниться следующий противник. Спаун по цепи происходит в методе ChainSpawn().

b) НПС, способные на диалог, спаунятся единовременно только раз за загрузку уровня.

4. **ChainSpawn()** - реализует спаун по цепи

5. **ChainTimeStart()** – метод, устанавливающий время для цепочного спауна.

3.2.2 Босс

В конце уровня игрока поджидает финальное испытание - Босс. Босс представляет собой компиляцию всех возможных типов противников в одном. Большая часть состояний были описаны на рисунках 10 − 13. Главными отличительными чертами поведения является два состояния, представленных на рисунках 14 − 15.

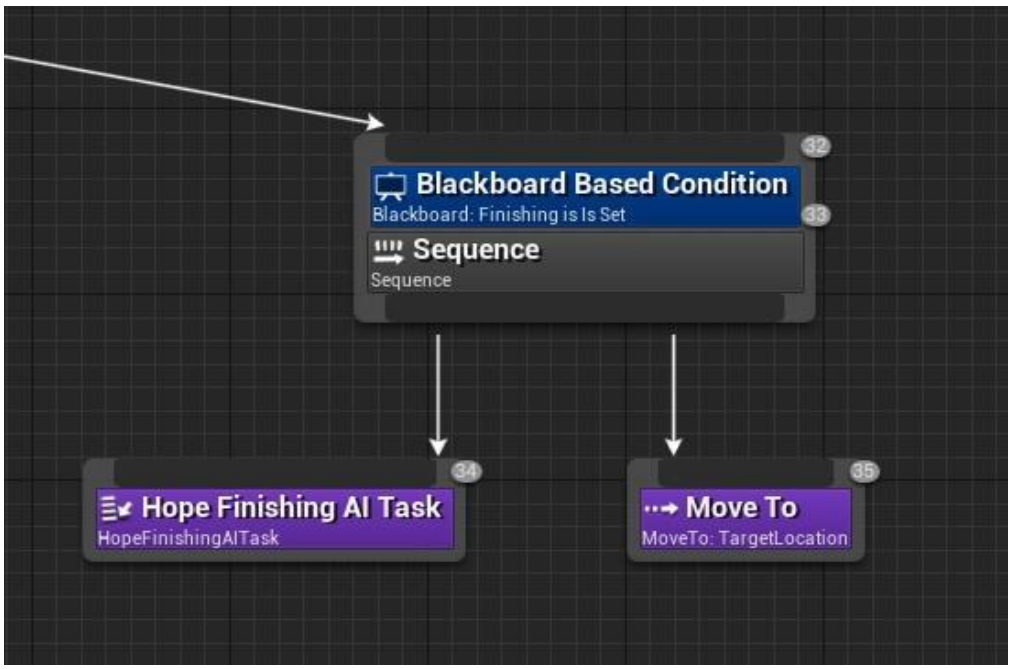

Рисунок 14 − Состояние мгновенного убийства игрока.

1. BlackBoard Based Condition - представляет собой проверку на то, закончилась ли шкала страха у игрока.

2. Hope Finishing AI Task - босс увеличивает свою скорость до максимальных значений и парализует игрока.

3. Move to - босс двигается к игроку и убивает его.

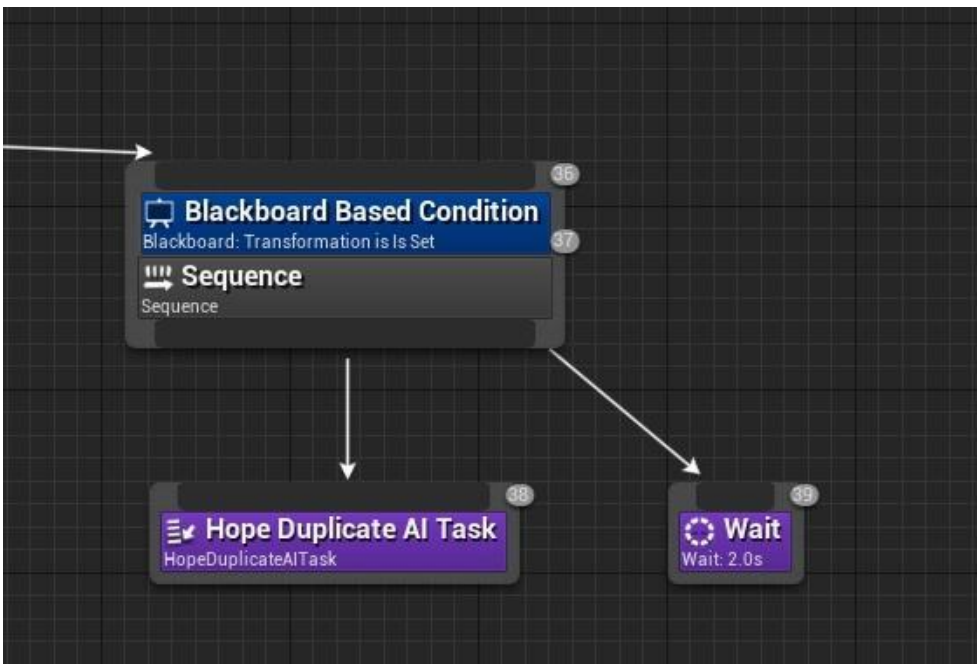

Рисунок 15 − дубликат босса

При снижении количества очков здоровья до определённого значения босс:

1. Hope Duplicate AI Task - создаёт свою копию и передаёт контроллеру информацию о том, что теперь он управляет сразу двумя объектами.
2. Wait - ожидание до возобновления действий.

3.3 Диалоговая система

В игре реализована система диалогов. Она представляет собой непосредственное общение между НПС, способными на диалог, и персонажем игрока. Разработана с целью ознакомления игрока с сюжетом компьютерной игры. На рисунке 16 изображена диаграмма классов НПС, способного на диалог.

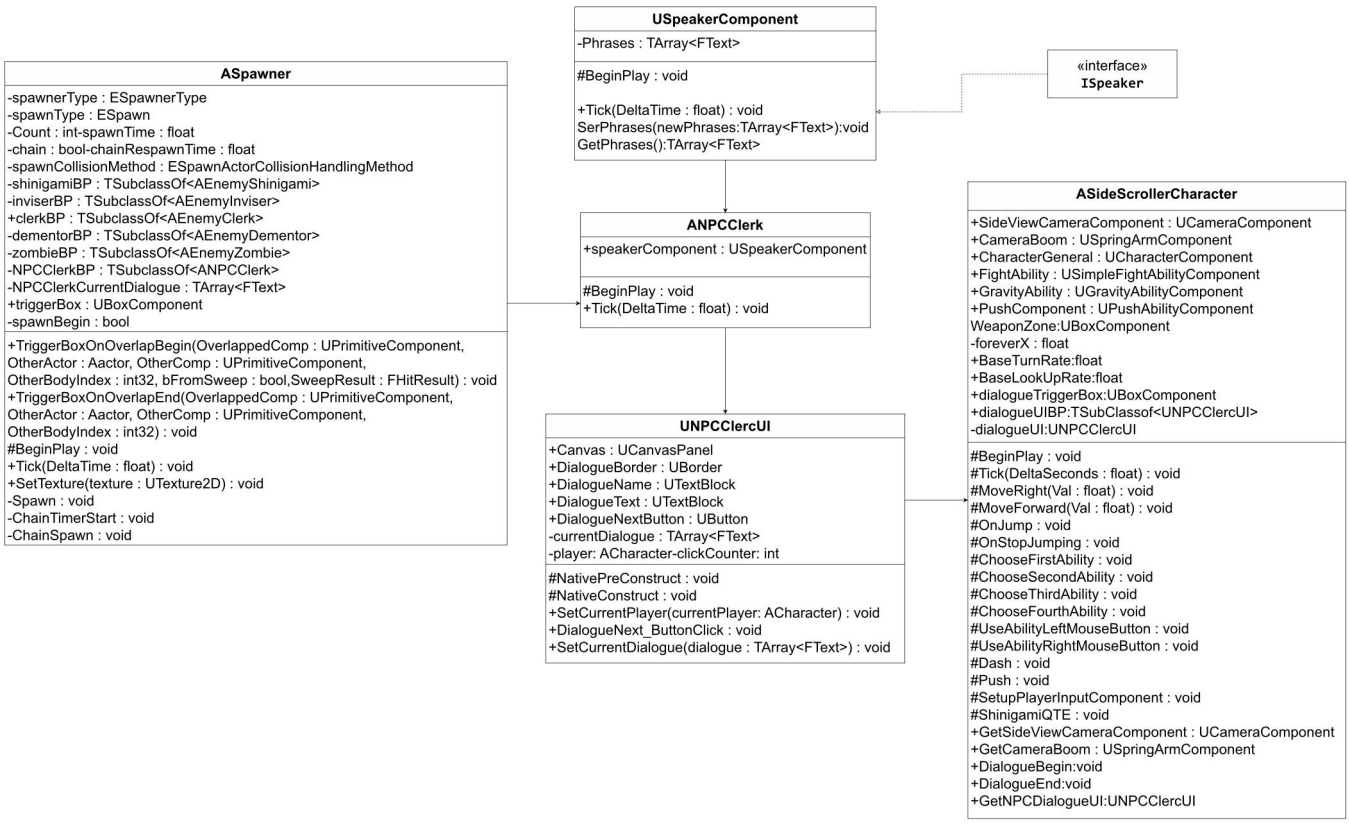

Рисунок 16 − Диаграмма классов диалоговой системы

За жизнедеятельность всей диалоговой системы в первую очередь отвечает интерфейс ISpeaker и класс USpeakerComponent.

ISpeaker - представляет собой интерфейс, который определяет типы объектов способных на разговор. Все, кто обладают данным интерфейсом, имеют возможность входить в диалог.

USpeakerComponent - представляет собой класс, который определяет содержимое диалога.

ANPCClerk - Класс-оболочка

Базовый персонаж, способный на диалог, имеет все три компонента: интерфейс, класс-оболочка, компонент диалога.

За отображение диалога отвечает класс UNPCClercUI: он реализует непосредственный вывод текста на экран. Он содержит панель диалогового окна, имя персонажа, с которым мы ведём диалог, сам текст диалога и кнопку, переводящую на следующую стадию разговора.

Описание методов:

1. **SetCurrentPlayer()** - метод назначения игрока, который инициировал персонаж.
2. **DialogueNext_ButtonClick()** - обработчик события нажатия на кнопки пропуска диалога. С одной страницы диалога происходит переход на другую.
3. **NativePreConstract()** – метод, описывающий прорисовку диалогового окна до разговора
4. **NativeConstruct()** – метод, описывающий прорисовку диалога во время разговора.
5. **SetCurrentDialogue()** - метод назначения текущего диалога. При его вызове высвечивается первая страница диалога.

ASpawn - способен спаунить, кроме противников, мирных НПС способных на диалог. Диалоги заготавливаются и распределяются заранее в данном классе.

Любой разговор инициирует игрок. При нажатии клавиши F запускается диалоговое окно, в случае если в области действия есть кто-либо, способный на диалог. За это отвечают методы DialogueBegin() класса ASideScrollerCharacter. Строчки диалога прописывается заранее как для НПС,

так и для подконтрольного персонажа и хранится в USpeakerComponent.

При нажатии на кнопку пропуска диалога срабатывает событие DialogueNext_ButtjnClick класса UNPCClercUI, после чего на экран выводится следующая строка. Путем общения с НПС и чтения диалогов, игрок будет постепенно узнавать больше о происходящих событиях и углубляться в повествование.

На рисунках 17 - 18 изображены примеры диалога.

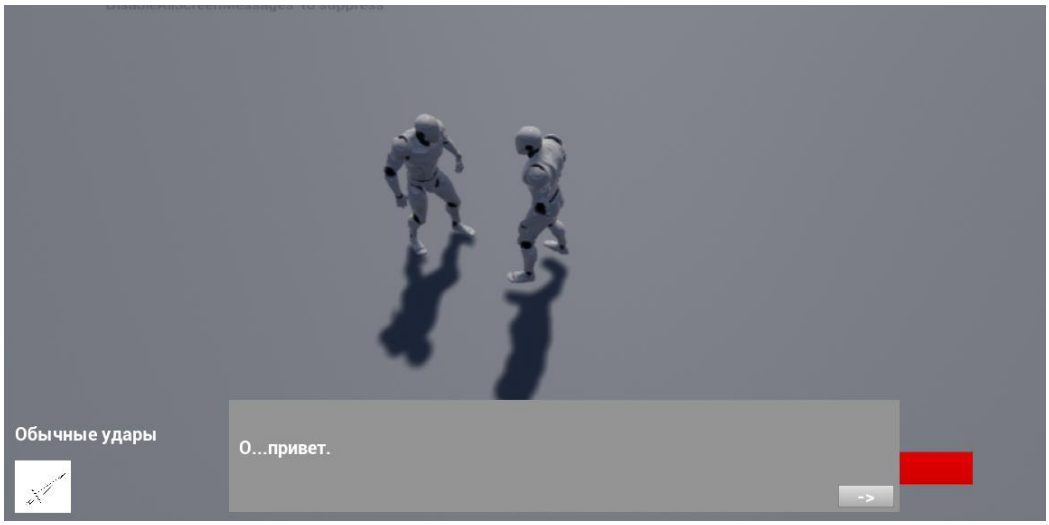

Рисунок 17 − пример работы диалогового окна

Рисунок 18 − пример работы диалогового окна

4 Тёмный мир

Особая локация, в которую игрок может попасть в случае, если первый раз встречает противника Дух. При обнаружении игрока, Дух на максимальной скорости соприкасается с ним и переносит его в тёмный мир, где необходимо собрать пазл из 3 частей, для того чтобы вернуться обратно. Всё время пребывания в тёмном мире игрок получает урон. На рисунке 19 изображена диаграмма классов перемещения в тёмный мир.

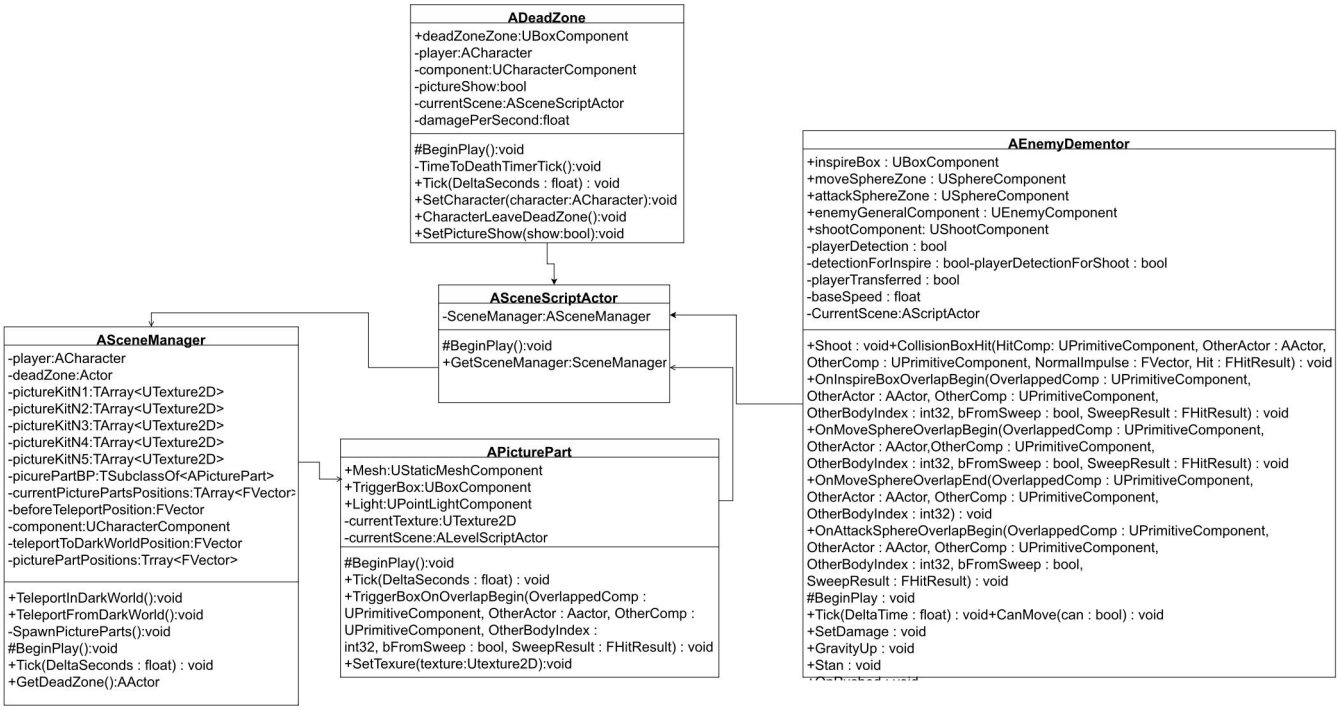

Рисунок 19 − диаграмма классов реализации темного мира

Перемещение в другой мир осуществляет противник Дух. При столкновении inspireBox Духа с игроком происходит событие OnComponentBeginOverlap(), которое обрабатывается методом OnInspireBoxOverlapBegin(). В этом методе идет обращение к SetCharacter() класса ADeadZone. В процессе работы SetCharacter() вызывается метод TeleportInDarkWorld() класса ASceneManager. Также в SetCharacter() запускается TimeToDeathTimerTick, в котором у игрока начинает ежесекундно уменьшаться количество здоровья. По завершению работы метода TeleportInDarkWorld() игрок телепортируется в темный мир на определенную координату. В этот момент запускается метод

SpawnPictureParts класса ASceneManager. В нем выбирается случайный комплект, хранящий в себе 3 картинки, которые являются кусочками пазла, и затем размещает их на локации темного мира. Если персонаж соберет все 3 картинки, то вызываются методы TeleportFromDarkWorld() и CharacterLeaveDeadZone(), после чего игрок перестанет терять здоровье и вернется на предыдущую локацию.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы спроектировано игровое приложение в жанре 3D платформер и реализован прототип на Unreal Engine 4, состоящий их двух готовых уровней.

Решены все поставленные задачи:

- спроектирована механика передвижений;
- спроектирована боевая система и специальные умения персонажей;
- спроектированы противники и их особые умения;
- спроектировано поведение противников;
- спроектирована диалоговая система;
- реализован прототип приложения.

Дальнейшие перспективы развития игры:

- улучшение ИИ противников путём добавления новых состояний и модификаций старых;
- добавление уникальных 3D моделей для улучшения визуальной составляющей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Little Nighmare [Электронный ресурс] – Электрон. Дан. – Режим доступа: http://littlenightmares.softclub.ru/, свободный.
2. Inside [Электронный ресурс] – Электрон. дан. – Режим доступа: https://playdead.com/games/inside/, свободный.
3. Pole [Электронный ресурс] – Электрон. дан. – Режим доступа: https://iuvenisstudios.com/, свободный.
4. Crash Bandicoot [Электронный ресурс] – Электрон. дан. – Режим доступа: https://www.crashbandicoot.com/, свободный.
5. Little Nighmare [Электронный ресурс] – Электрон. дан. – Режим доступа: https://ru.wikipedia.org/wiki/Little_Nightmares, свободный
6. Unreal Engine 4 [Электронный ресурс] – Электрон. дан. – Режим доступа: https://ru.wikipedia.org/wiki/Unreal_Engine, свободный
7. Документация по Unreal Engine 4 [Электронный ресурс] – Электрон. дан. – Режим доступа: https://docs.unrealengine.com/4.26/en-US/, свободный.
8. Visual Studio 2019 [Электронный ресурс] – Электрон. дан. – Режим доступа: https://visualstudio.microsoft.com/ru/vs/, свободный.
9. Василюк Ф. Е. Пережить горе // О человеческом в человеке / под ред. И. Т. Фролова. — М.: Политиздат, 1991. — С. 230—247. — URL: http://psychlib.ru/inc/absid.php?absid=158548.
10. Blueprints [Электронный ресурс] – Электрон. дан. – Режим доступа: https://it-cube48.ru/archives/18002, свободный.
11. Behavior Tree [Электронный ресурс] – Электрон. дан. – Режим доступа: https://it-cube48.ru/archives/18002, свободный

АНТИПЛАГИАТ
Обнаружение заимствований

Отчет о проверке на заимствования №1

Автор: 645105@bk.ru / ID: 9257968
Проверяющий:

Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 6
Начало загрузки: 05.06.2022 12:48:09
Длительность загрузки: 00:00:01
Имя исходного файла: ВКР_ЛевкевичВИ.pdf
Название документа: ВКР_ЛевкевичВИ
Размер текста: 41 кб
Символов в тексте: 42360
Слов в тексте: 5087
Число предложений: 502

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 05.06.2022 12:48:11
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Интернет Free

ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
1.79% ↓	0%	0%	98.21%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты, общеупотребительные выражения, фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска
[01]	1.61%	не указано http://rhsace.susu.ru	08 Ноя 2018	Интернет Free
[02]	0.18%	Программа графического отображения данных от мультиметра http://library.elteh.ru	19 Сен 2019	Интернет Free
[03]	0%	не указано https://bspu.ru	22 Окт 2019	Интернет Free

С результатом ознакомлен. /Трисчупа А.В./

Еще источников: 7
Еще заимствований: 0%