Close Combat: Swordsman

Документация

(Unreal Engine 4 ассет)

https://www.unrealengine.com/marketplace/close-combat-swordsman

Видео демонстрации проекта

Документация написана автором проекта ZzGERTzZ.

Manual version : 1.0

1. Введение:

Проект **CloseCombat:Swordsman** создан с целью предоставить разработчикам заготовку для игр в жанре Экшен/РПГ, слэшера или файтинга включающие в себя боевую систему, систему перемещения персонажа, систему прокачки уровней персонажа, систему скиллов персонажа.

В пакет входят следующие элементы для геймплея:

Перемещение. Базовое перемещение с процедурными наклонами, чтобы можно было обходится меньшим кол-вом анимации и при этом чувствовалось реакция персонажа на смену направления при движении. Присутствует система прыжков, перемещение шагом и бегом. Так же присутствует боевая система с шагом в стойке на все направления, перебежками, уворотами и прочее..

Компонент здоровья. Данный компонент содержит в себе параметры здоровья персонажа, а так же может быть применен не только к персонажам, но и к любым **actor**'ам, которым может понадобиться здоровье, например турель, которая должна разрушаться от повреждений. Компонент так же содержит в себе систему восстановления здоровья это может пригодится при постоянной регенерации здоровья.

Оружие ближнего боя. Класс оружия ближнего боя содержит в себе визуальный вид оружия, настройки повреждения, типы ударов. В проекте присутствуют 3 различных меча в разных стилистиках:*Футуристичный меч, Меч света, Меч тьмы.*

ИИ. Базовый искусственный интеллект, который может видеть, слышать, преследовать и атаковать в ближнем бою игрока. Умеет уворачиваться от ударов, перемещаться в бою, ставить блоки и прочее. Также есть функционал патрулирования по игровому уровню. Присутствует удобная и простая настройка характера поведения персонажа в бою, которую можно менять динамически.
Так же в проекте приступает турель, которая использует бой на расстоянии.

Компонент начисления опыта. Данный компонент содержит в себе всю необходимую логику для повышения уровня персонажа. Также содержит систему скилов содержащую необходимые элементы для их создания - кулдаун (время восстановления скилла), активный,пассивный переключаемый типы скиллов.

Игровой интерфейс. В проекте присутствует полностью анимированный интерфейс включающий в себя элементы здоровья как игрока так и врагов, показатели уровня и опыта персонажа, элементы визуального представления скилов персонажа.

Обзор в бою. Кинематографичная боевая камера.

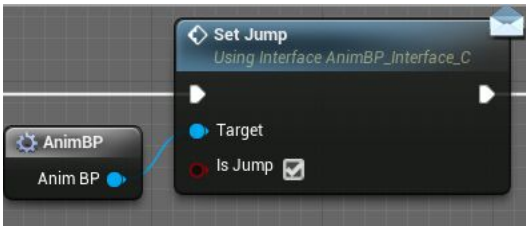
2. Базовое перемещение :

Перемещение персонажа основано на базовом классе перемещения от “*Epic games*” **character movement**. Всвязи с этим все параметры перемещения настраивайте в данном компоненте. Персонаж умеет двигаться в любом направлении, совершать прыжки.

Анимация от пешки в anim blueprint передается посредством интерфейсов

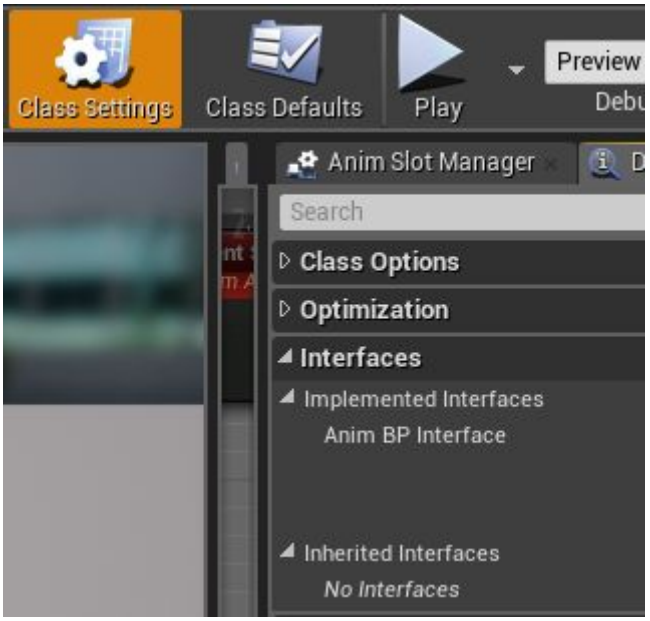
Это сделано ради того чтобы не привязывать пешку персонажа к определенному anim blueprint и впоследствии упрощая интеграцию других персонажей, которые используют свой скелет. После интеграции персонажа вы можете использовать прямую ссылку на anim blueprint посредством “cast to” или же добавляя функции в интерфейс “AnimBP_Interface”.

Разберем как это выглядит на примере прыжка. После нажатия на кнопку прыжка идут различные просчеты (может ли персонаж прыгнуть в текущий момент?) и если персонаж прыгает то мы вызываем сообщение интерфейса под названием Set Jump (это функция которую мы предварительно создали в AnimBP_Interface) для вызова в AnimBlueprint необходимого ивента который активирует анимацию персонажа и переводит его в состояния прыжка.

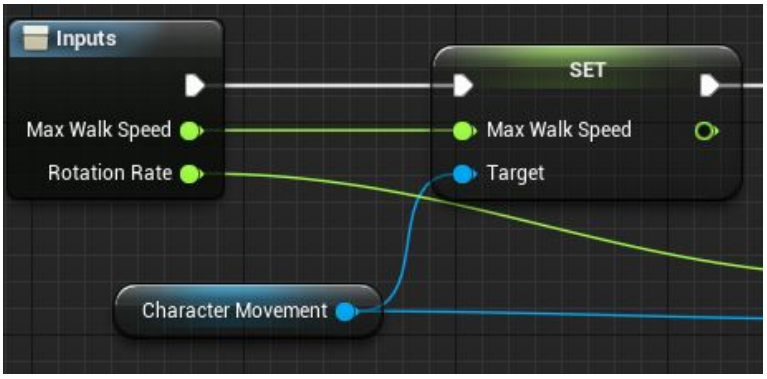


Все ивенты интерфейса AnimBP_Interface в anim blueprint располагаются здесь

Чтобы вызвать ивенты интерфейса вы должны предварительно его подключить в “Class settings”.



Что касается логики перемещения персонажа то оно построено на динамическом изменении параметров класса компонента “character movement” от “epic games”. На примере бега и шага меняется максимальная скорость при нажатии на клавишу переключения с бега на шаг (по умолчанию alt)



В зависимости от скорости персонажа меняется и анимация в anim blueprint.

Таким образом осуществляется связь анимации и перемещения персонажа в пакете “**CloseCombat:Swordsman**”.

3. Компонент здоровья :

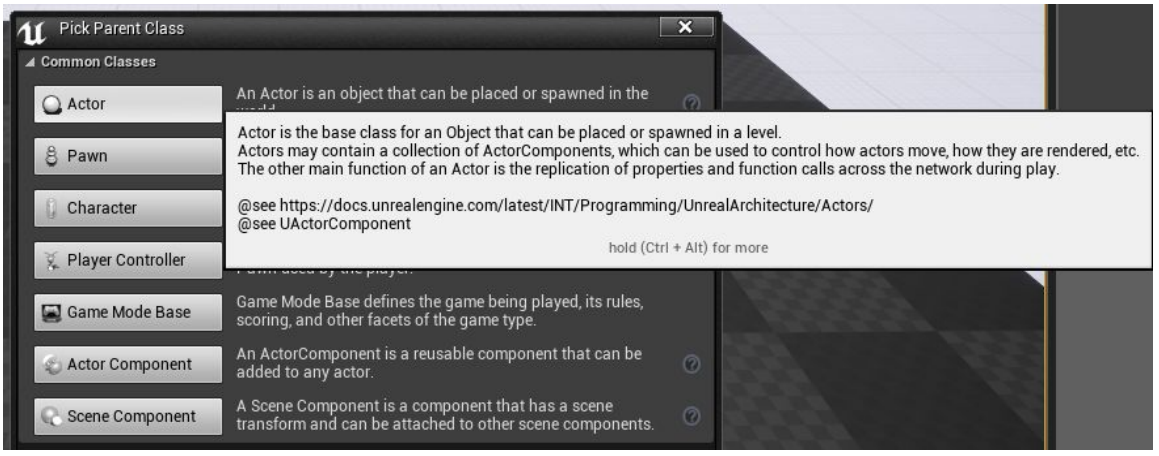
Компонент здоровья служит для унифицированной системы подключения параметров здоровья к любым акторам.

Как это работает:
Подключая компонент к актору вы передаете в этот компонент данные о повреждениях актора (any damage), а компонент в свою очередь считает здоровье, при определенных настройках восстанавливает его и вызывает ивент смерти в случае отсутствия здоровья. Ивент смерти передается по интерфейсу актору к которому относится данный компонент компонент.

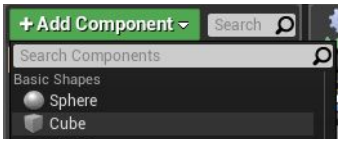
Как использовать компонент здоровья:

Разберем пример использования компонента здоровья.

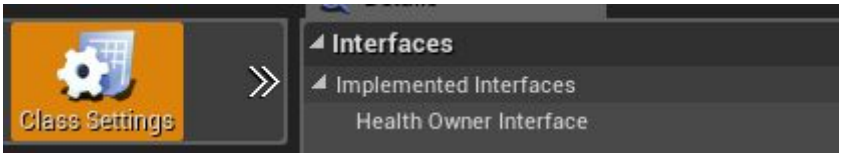
1. Создадим пустой актор. Правой кнопкой мыши в контент браузере=>Blueprint Class



2. Добавим в наш актор для пример куб. Он будет служить визуальном предсталвением нашего актора.

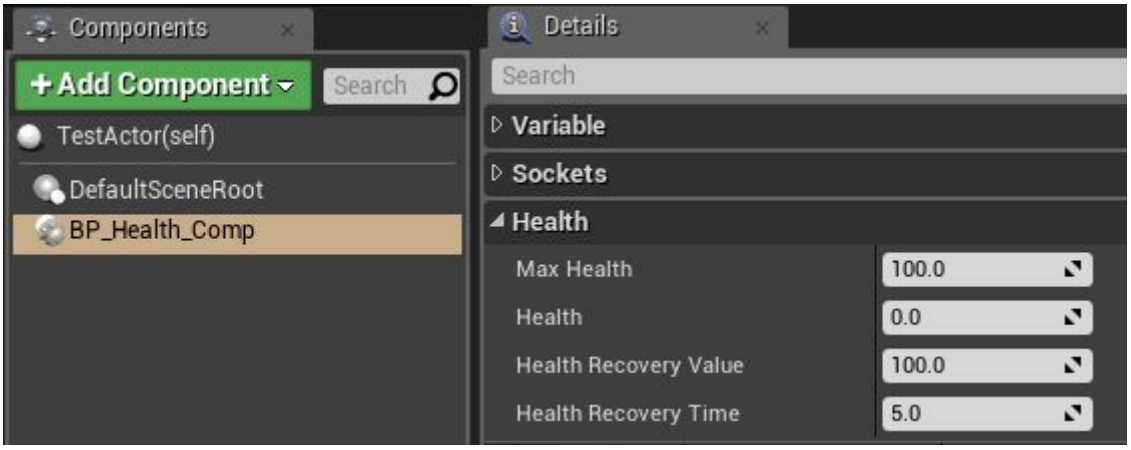


3. Так как компонент здоровья имеет унифицированную систему, то он передает ивенты по интерфейсам. В первую очередь подключаем интерфейс компонента здоровья нашему актору. Для этого переходим в class settings и добавляем интерфейс “**HealthOwner_Interface**”.



4. Добавим компонент здоровья нашему актору.

5. Настроим наш компонент здоровья - для этого выделите компонент здоровья “BP_Health_Comp”, после появятся его настройки в вкладке “Details”



Параметры:

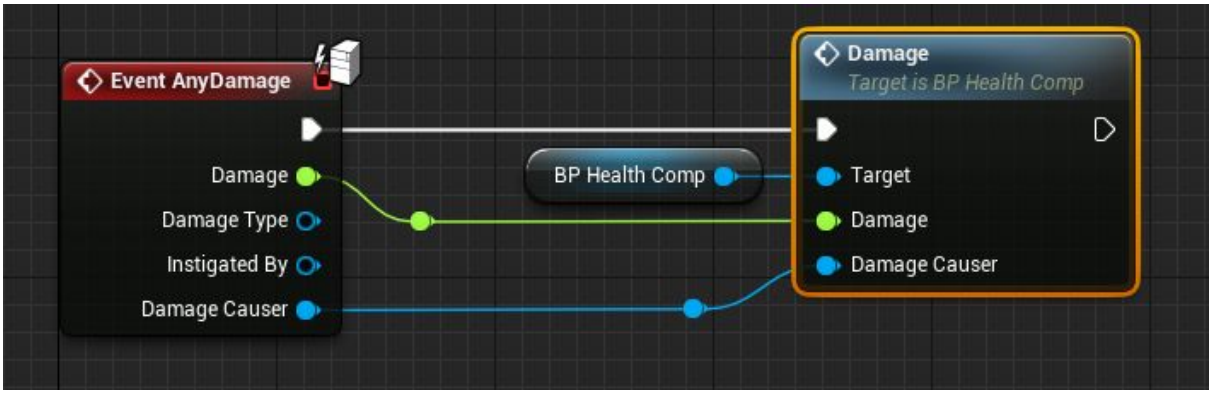
Max Health - максимальное кол-во здоровья.

Health - текущее здоровье

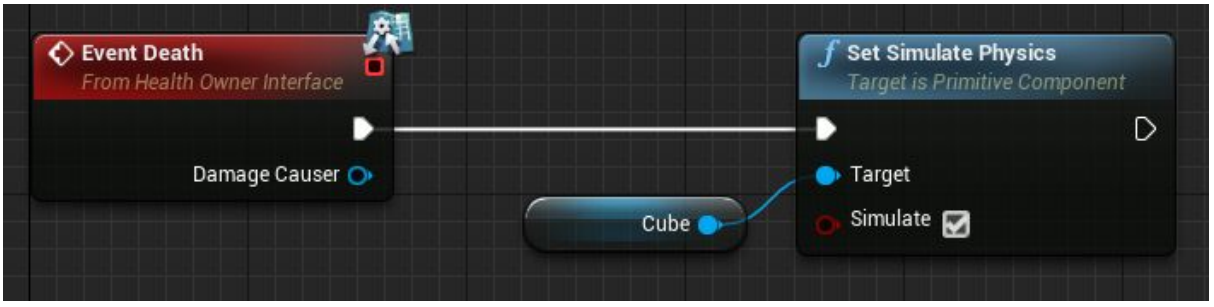
Health_Recovery_Value - до какого кол-ва здоровья будет продолжаться регенерация. При нулевом значении регенерация будет продолжаться до полного восстановления здоровья.

Health_Recovery_Time - за сколько времени восстановится здоровье при нулевом значение регенерации здоровья не будет.

6. Передадим данные о повреждениях компоненту здоровья. Для этого вызываем ивент у компонента Damage при получении повреждения актора.



7. Последним шагом будет вызов ивента смерти. Для этого добавим в ивент граф ивент интерфейса “Death”. На этот ивент указывайте вашу логику смерти для нашего тестового актора я добавил физику нашему кубу.



Теперь вы можете добавить на сцену наш актор куб и после нескольких ударов (зависит от здоровья указанного вами) куб станет физическим и упадет.

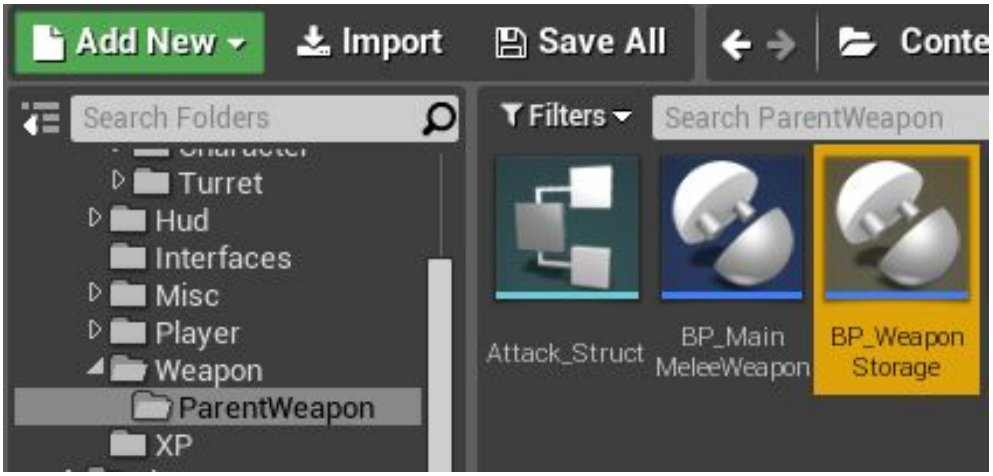
4. Оружие ближнего боя :

Оружие в проекте так же является компонентом и добавляется к персонажу посредством узла add component. В данном моменте содержится информация о модели оружия, список ударов с помощью данного оружия, скорость ударов и нанесения повреждений.

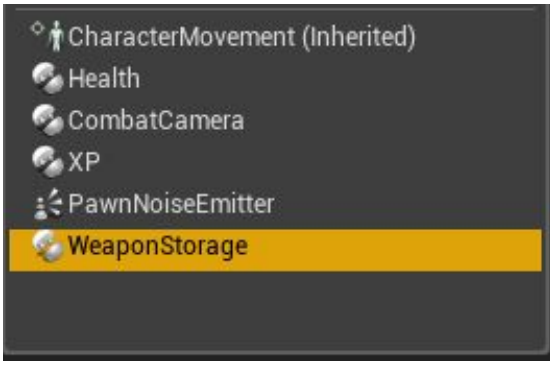
Как это работает: При нажатии на кнопку удара и соблюдения всех условий (персонаж не в воздухе, не мертв и т.д.) персонаж обращается к компоненту и получает от него анимацию удара, скорость проигрывание анимации (скорость атаки), время через которое можно прервать атаку (начать двигаться после взмаха не дожидаясь конца анимации).

Как добавить оружие ближнего боя.

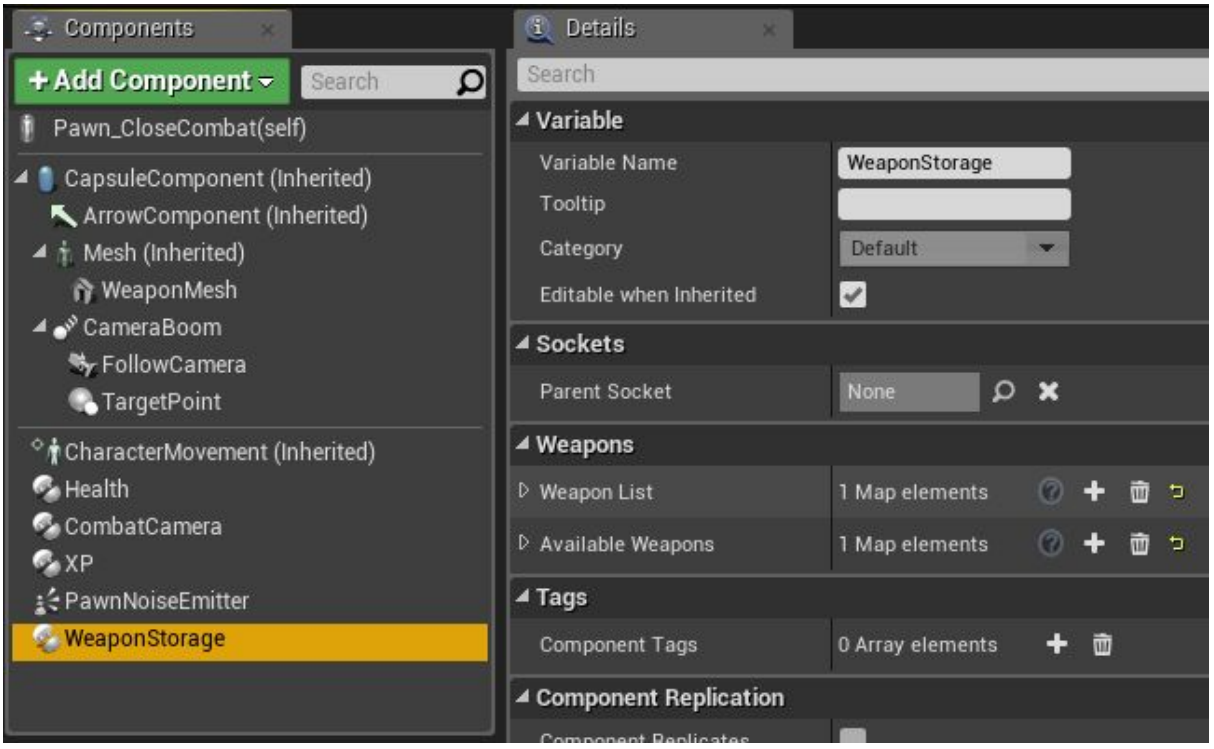
Во втором обновлении в проекте присутствует специальный компонент для создания и хранения оружия - BP_WeaponStorage. Он располагается в папке с классом оружия.



Следующим шагом нужно добавить данный компонент к пешкам (игрока и врага). По умолчанию данные компоненты уже добавлены и переименованы (удален префикс BP_).
По умолчанию в павне он переименован в WeaponStorage.

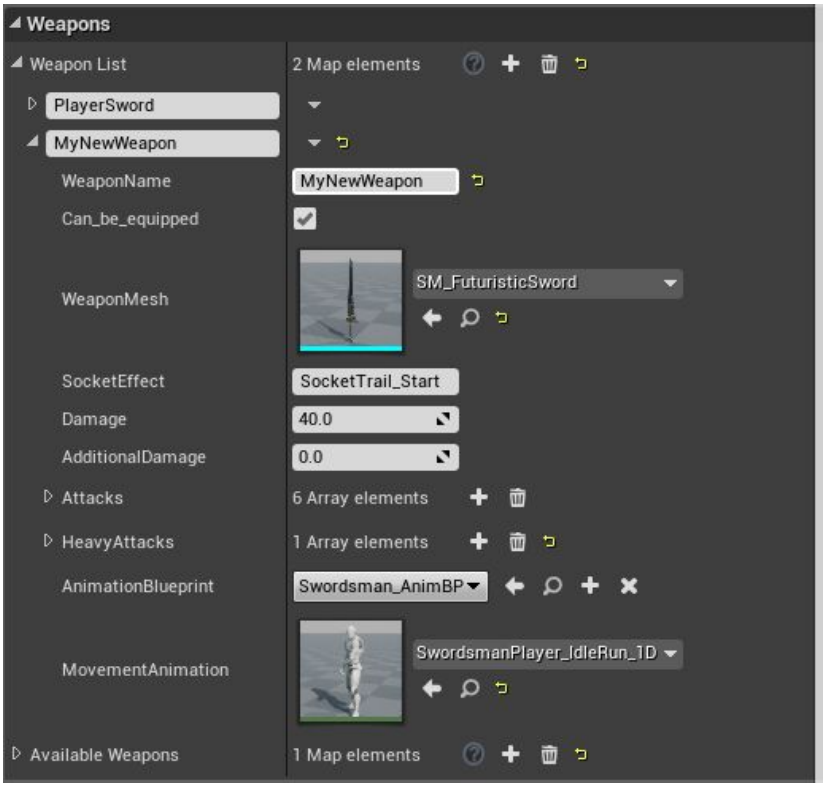


Выделите данный компонент и в вкладе “детали” найдите рубрику “Weapons” в ней вы увидите две map-переменные:

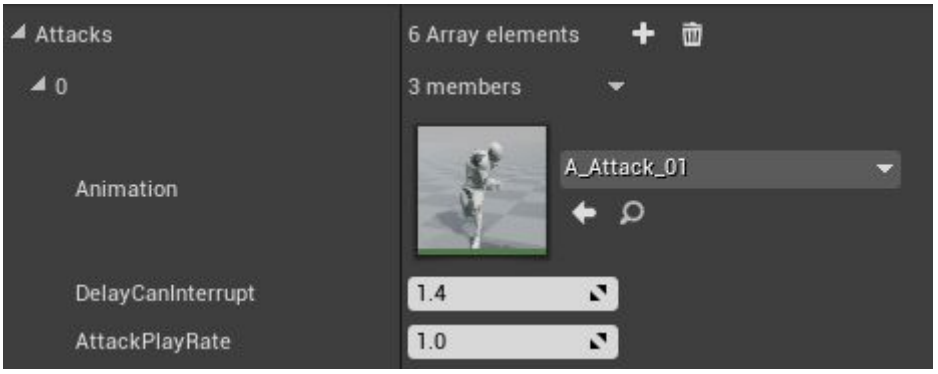


WeaponList - Список всего оружия которое может быть добавлено к персонажу.
AvailableWeapons - Список всего доступного оружия (на подобии оружие в инвентаре). В данном списке можно хранить оружие в котором вы изменили изначальные параметры (апгрейт).

По умолчанию в списке присутствует оружие “PlayerSword” для добавления нового оружия добавьте в map-массив структуру оружия (кнопка с изображением плюса). Укажите название вашего нового оружия (внимание оружие не может содержать одинаковых названий (map-varible).
Рассмотрим параметры оружия:

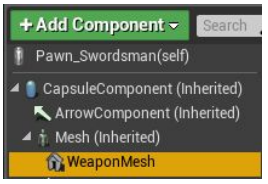


WeaponName - Название оружия, нужно указать такое же как и в массиве, на данном примере MyNewWeapon
Can be equipped - Может ли данное оружие быть убрано за спину (обновление 1 - https://youtu.be/mzYNtDsmEOg). Переменная нужна для оружия которое не имеет модели к примеру рукопашный бой.
Weapon mesh - Визуальный вид оружия,выбор модели.
Socket Effect - Точка из которой будут проигрываться партиклы. (искры)
Damage - Базовое повреждение оружия от удара.
AdditionalDamage - Дополнительное повреждение, лучше добавлять повреждения через дополнительные ивенты (описанные чуть ниже).
Attacks - Список атак от данного оружия, по умолчанию атаки идут одна за другой (вы можете собирать комбо).

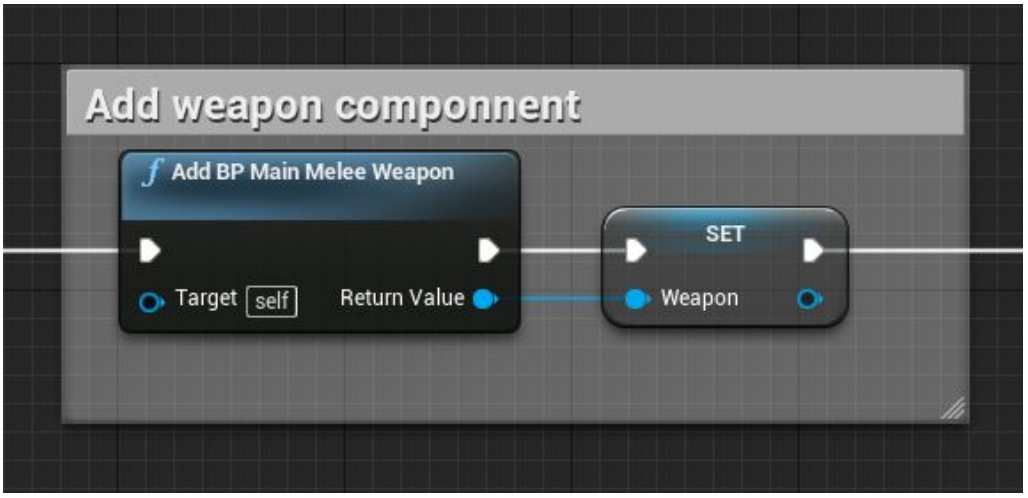


Animation - Анимация удара.
Delay Can Unterrupt - Задержка после которой можно прервать анимация перемещением.
Attack PlayRate - Начальная скорость проигрывания анимации удара (скорость атаки).

Следующим шагом будет добавление оружия к нашему персонажу. Для этого создайте меш и присоедините его к персонажу к сокету оружия.(Socket_Weapon). По умолчанию данный компонент присутствует.

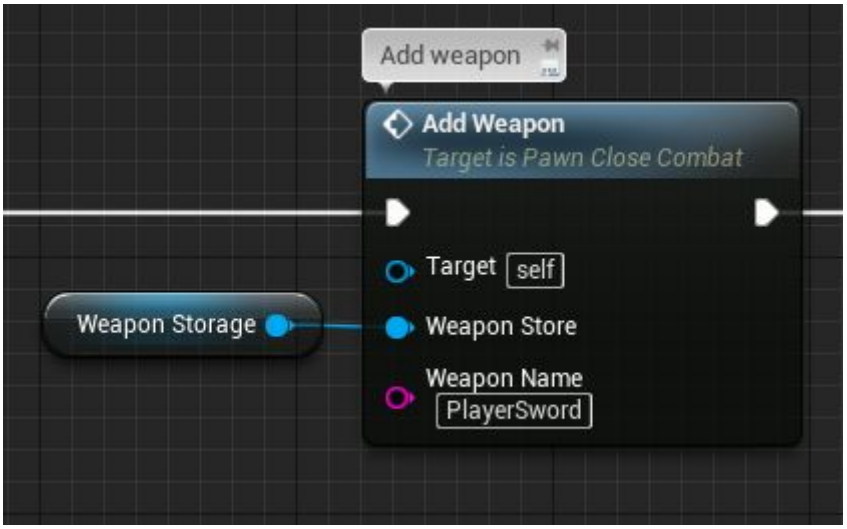


Далее добавим компонент оружия (по умолчанию компонент добавляется на ивент begin play).



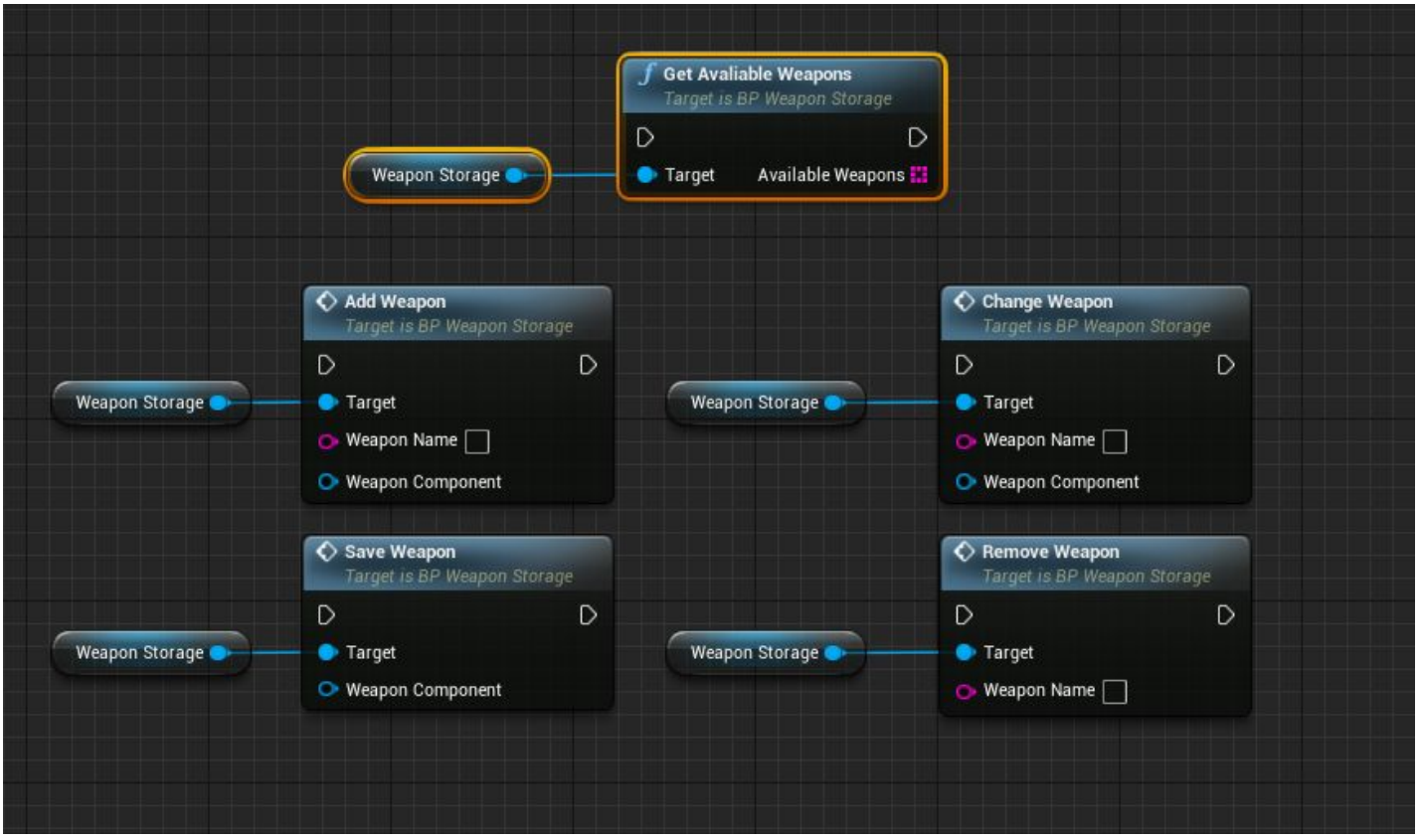
При добавлении компонента стоит указать переменную “**Weapon**” (тип переменной - **BP_MainMeleeWeapon**) нашего оружия чтобы в дальнейшем мы могли ссылаться на наше оружие из любого места. Данный компонент содержит в себе логику ближнего оружия а компонент “WeaponStorage” хранит в себе параметры для этого компонента чтобы перезаписывая их менять оружие.

Следующим шагом укажем компоненту оружия какое оружие использовать. Для этого вызовите функцию “Add Weapon” в павне и напишите названия вашего оружие, создания которого мы рассматривали выше.



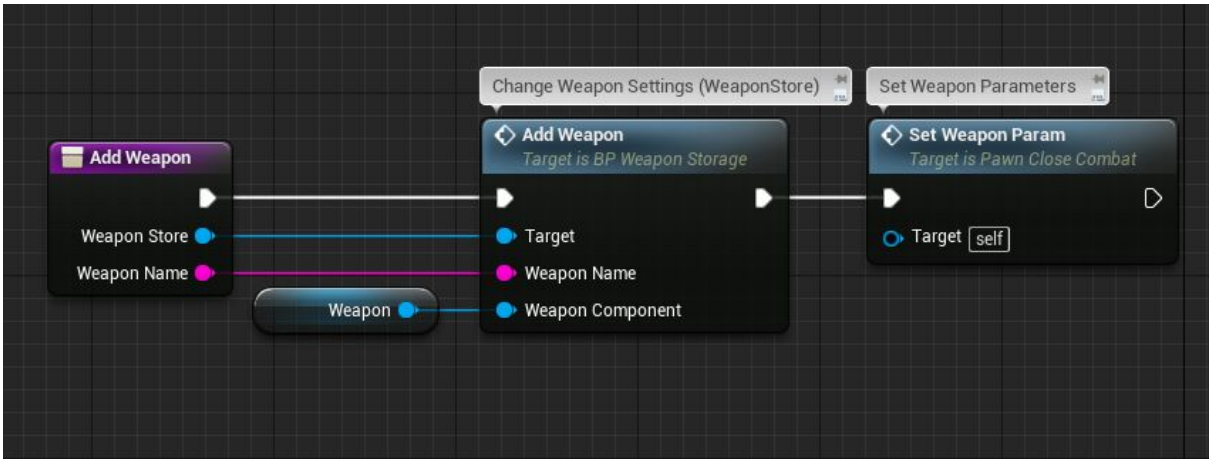
По умолчанию это “PlayerSword”. Данная функция запишет в компонент оружия параметры из списка оружия “WeaponList”.

Рассмотрим все ивенты при работе с компонентом WeaponStorage:

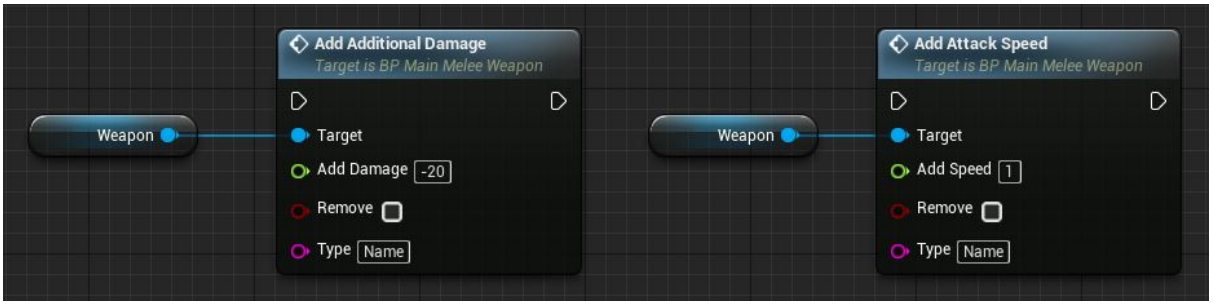


Save Weapon - Сохраняет текущее оружие добавляя его в список “AvailableWeapons”.
Change Weapon - заменяет текущее оружие на указанное из списка доступных оружий (available weapons).
Remove Weapon - удаляет указанное оружие из списка доступных оружий (available weapons).
Get Available Weapons - Выдает список оружия доступного персонажу.

Функции в компоненте “WeaponStorage” addweapon и changeweapon записывают новые параметры в компоненте оружия, но чтобы применить новые параметры к персонажу вам нужно вызвать функцию SetWeaponParam сразу после изменения параметров. Для удобства в пешке созданы функции Addweapon и ChangeWeapon где после применения параметров сразу вызывается функция SetWeaponParam.



Дополнительные ивенты в оружии:



Add Additional Damage - Данный ивент служит для увеличения или уменьшения дополнительного урона от оружия. (К примеру от повышения уровня персонажа или от различных скилов). Используется функция “MultiIncrease”.
Add Attack Speed - Данный ивент служит для увеличения или уменьшения скорости атаки. (К примеру от повышения уровня или от различных скилов). Используется функция “MultiIncrease”.

Нанесения повреждения (“Notify_Damage”)

Нанесения повреждения производятся в классе оружия по ивенту “**CauseDamage**” в классе оружия, но данные для повреждения передаются через анимацию посредством Notify.

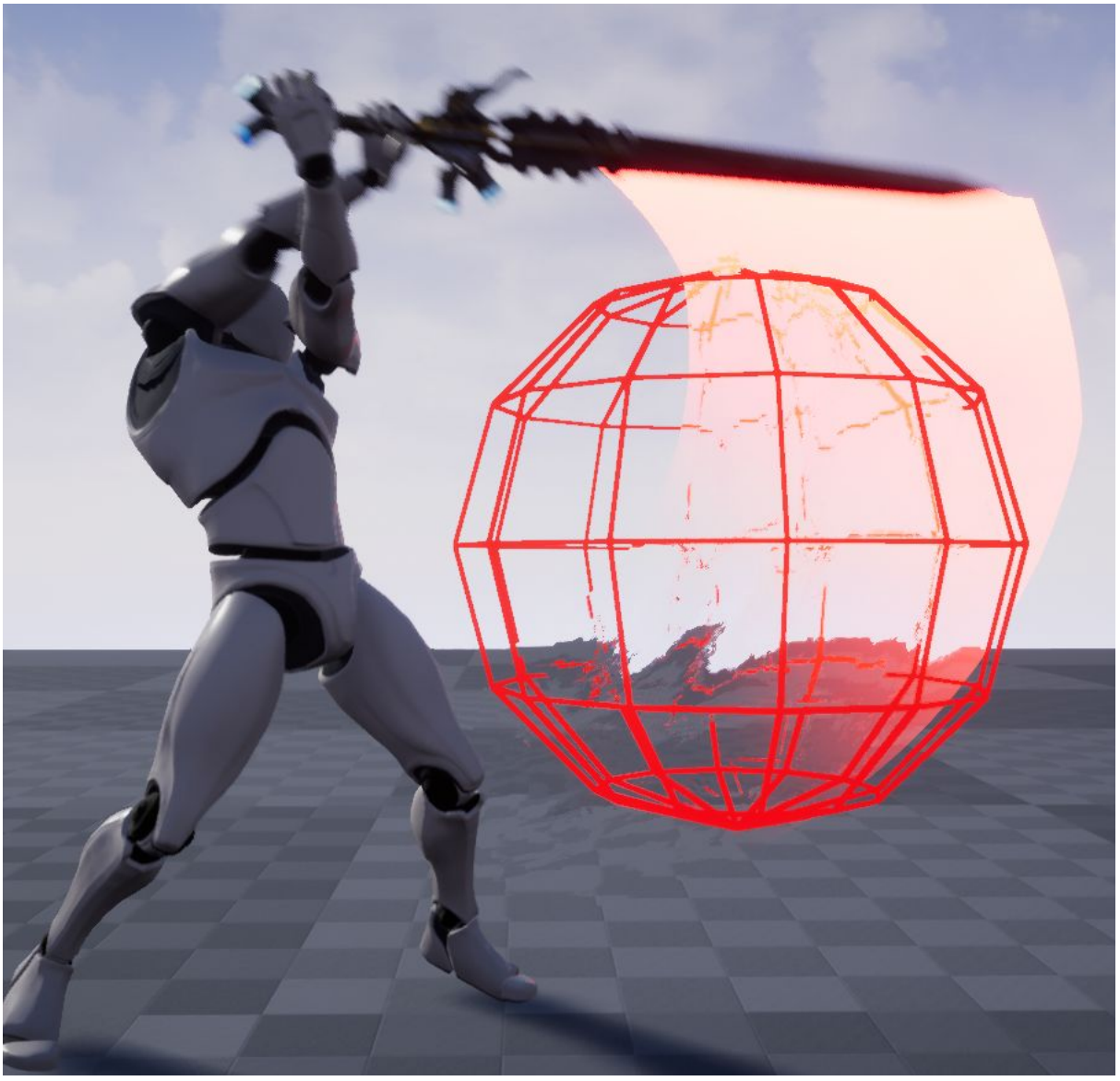
Для примера откройте любую анимацию атаки и на шкале notify вы обнаружите “**Notify_Damage”**

Notify_Damage должен располагаться в кадре нанесения урона, чтобы расположить notify кликните правой кнопкой мыши в нужном кадре и выберите **Notify_Damage**.
Выделив notify вы увидите его настройки во вкладке Details. Раскройте список параметров “Damage Parameters”:

Damage Animation Type - тип анимации повреждения у цели (влево,вправо или назад). Анимация выбирается вручную в ***Enemy_Param*** в настройках **AI** (или заполните таблицу .
DamageMultiply - множитель повреждений от текущего удара.

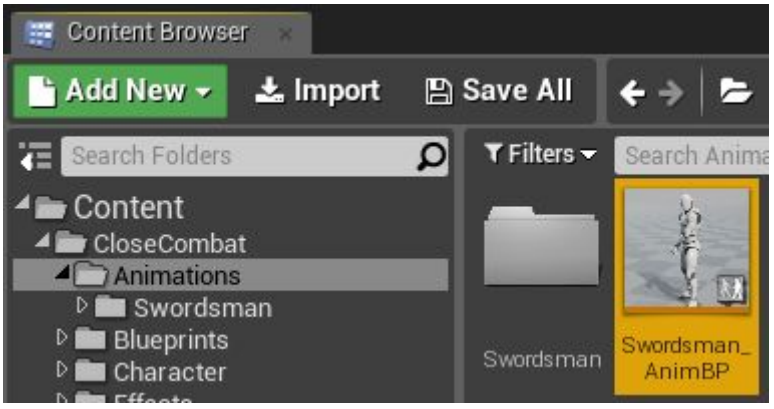
Повреждения по целям проходит по следующему принципу. Создается сфера с заданным радиусом и расположением относительно атакующего которая пересекает акторы и если среди пресечения актов есть теги цели (враждебные группы) то они получают урон. Вы можете отключить проверку цели и урон будет нанесен всем акторам.

Distance - дистанция от персонажа до центра сферы.
DamageRadius - радиус сферы которая пересекает акторы.
DrawDebugDamage - визуальное отображение сферы для удобной настройки.

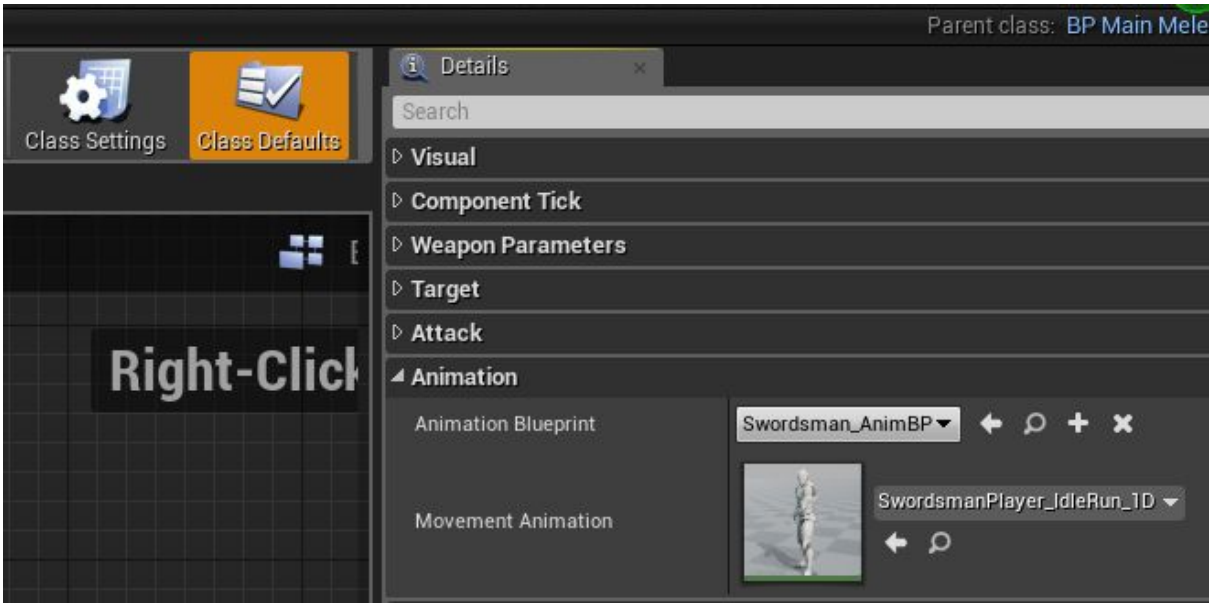


Анимации персонажа с оружием.

Все анимации персонажа (перемещение, поведение в бою, анимация умений..) содержатся в анимацонном блупринте.

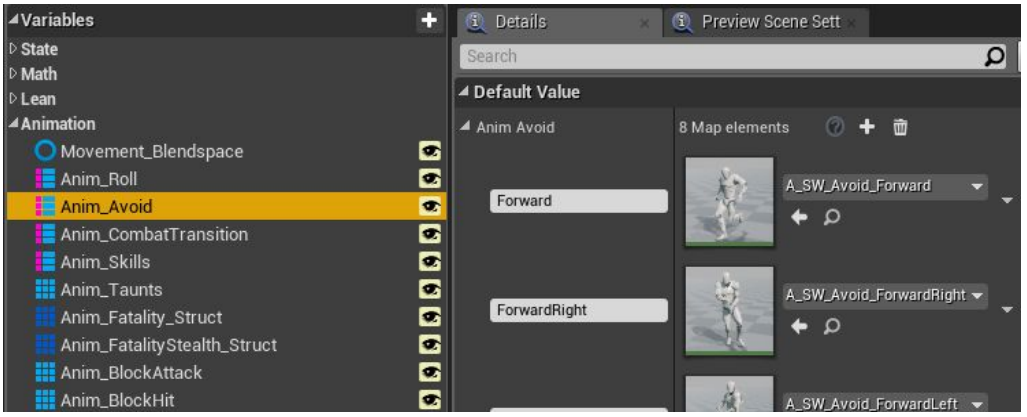


Структура построена таким образом что класс оружия содержит в себе ссылку на анимационный блюпринт меняя полностью все анимации персонажа. Это довольно удобно при добавлении новых видов оружия. К примеру вы задумали создать глефу для этого просто скопируйте анимационный блюпринт указав в нем новые анимации. Далее укажите в классе оружия, какой анимационный блюпринт использовать.



Animation Blueprint - Анимационный блупринт который будет использоваться при подключении данного оружия.
MovementAnimation - Выбор бленд спейса перемещения. (У врагов могут быть свои анимации перемещения, а боевые анимации те же самые).

Все дополнительные анимации указываются в категории “Animation”

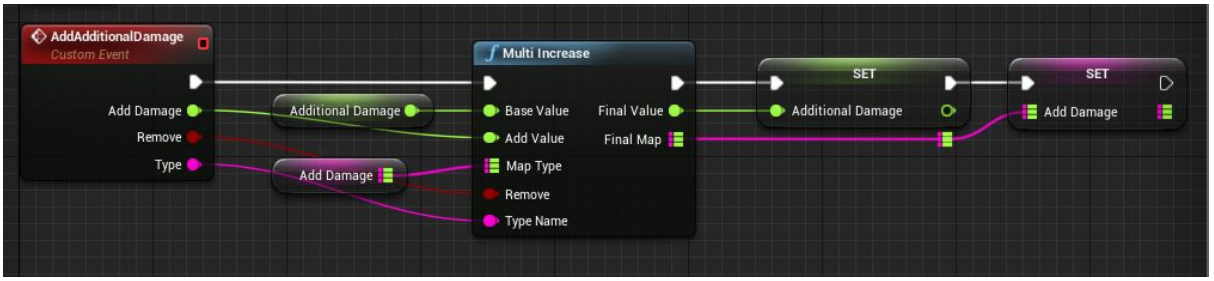


Библиотека функций “CloseCombat_FunctionLibrary”

В данной библиотеке содержатся функции которые могут быть полезны вне одного актора.

Функция “MultiIncrease”

Одна из функций это ***MultiIncrease***. Данная функция служит для изменения значений от разных источников. К примеру повреждения от оружия у вас может повысится от скила и повысится от повышения уровня и так как от скила повышение урона может быть временным, то на помощь приходит функция ***MultiIncrease***.



Функция работает следующим образом. Подается значение (Base Value) к которому надо прибавить или вычесть другое значение (Add Value). Далее данное значение (Add value) записывается в map переменную под определенным ключом (именем). Булевая переменная Remove отвечает за добавления изменений или удаление внесенных изменений на практике это выглядит так.

Вначале к повреждению оружия прибавляется 30 урона под типом “LevelUp”, далее прибавляется 20 урона под типом “Berserker” и через 5 секунд из общего кол-ва дополнительного урона вычитается значения под типом “Berserker” но составляет 20. При булевой переменной Remove=true указывать значение не обязательно так как оно уже в базе.
Для всех компонентов в которых используется функция “MultiIncrease” для удобства сделаны ивенты и начинаются они со слова Add..

Функция “TargetTags”

Данная функция служит для добавление или удалению тегов к актору. Работает с массивом чтобы можно было добавлять или удалять сразу несколько тегов.

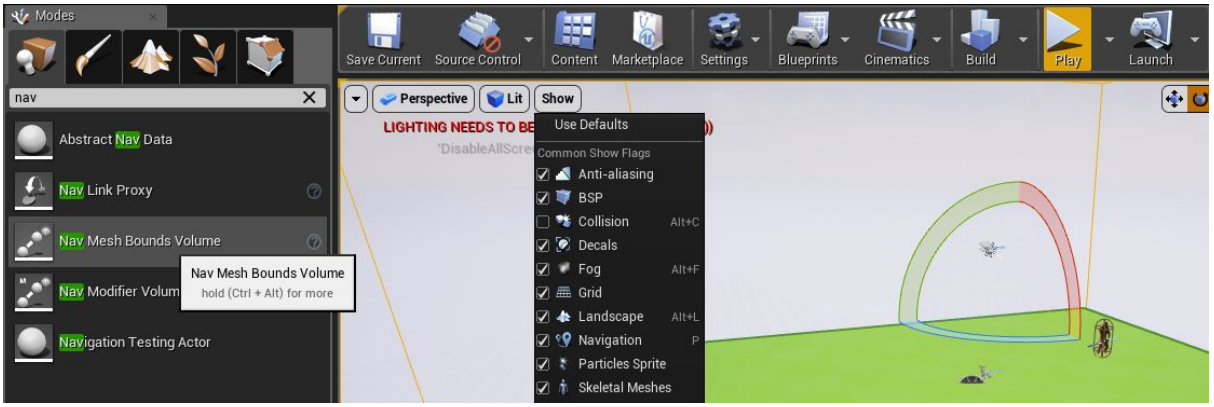
Функция “CheckTargetTag”

Данная функция проверяет есть ли в определенном акторе определенные теги и елси теги присутствуют выдает на выходе булевую переменную цель.

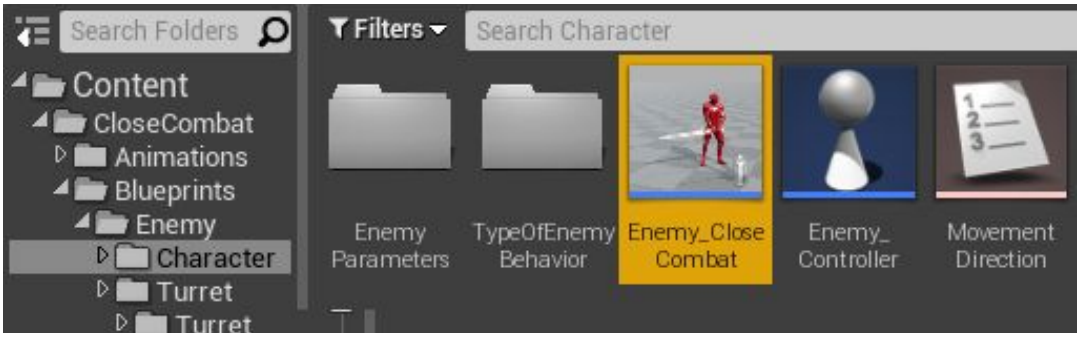
Искусственный интеллект (враги)

В проекте присутствуют противники которые могут патрулировать местность и если обнаруживают цель атаковать её в ближнем бою. Для начала рассмотрим добавление противника на сцену. Это можно сделать несколькими способами мы рассмотрим способ непосредственно добавления противника на сцену и спавна противника на сцене посредством Blueprint.

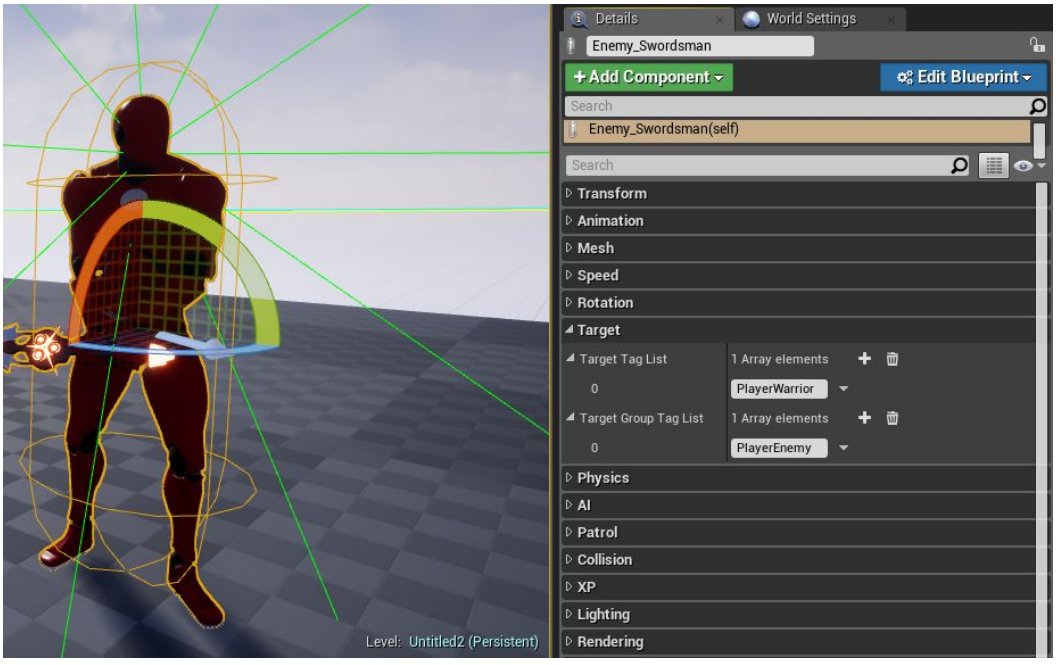
Первым делом посчитаете навигацию на вашем уровне, делается это с помощью добавления волума “*Nav mesh Bounds Volume*” внутри которого должна располагаться ваша сцена где могут перемещаться потенциальные противники.



Следующим шагом разместите вашего противника на сцену посредством “click and drop” из content browser.

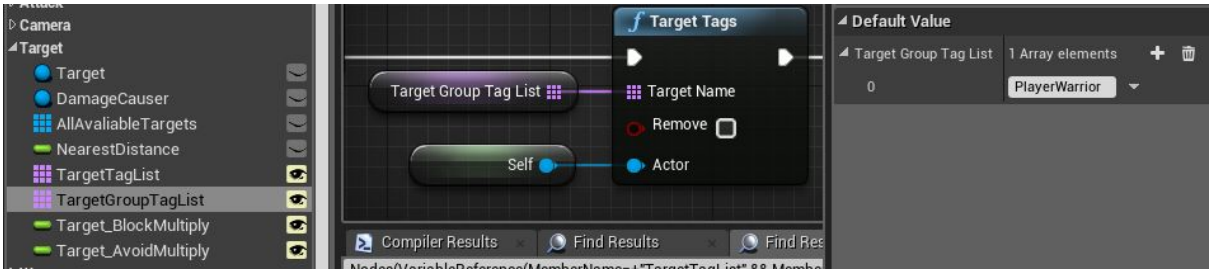


Выделите персонажа и рассмотрим его настройки.

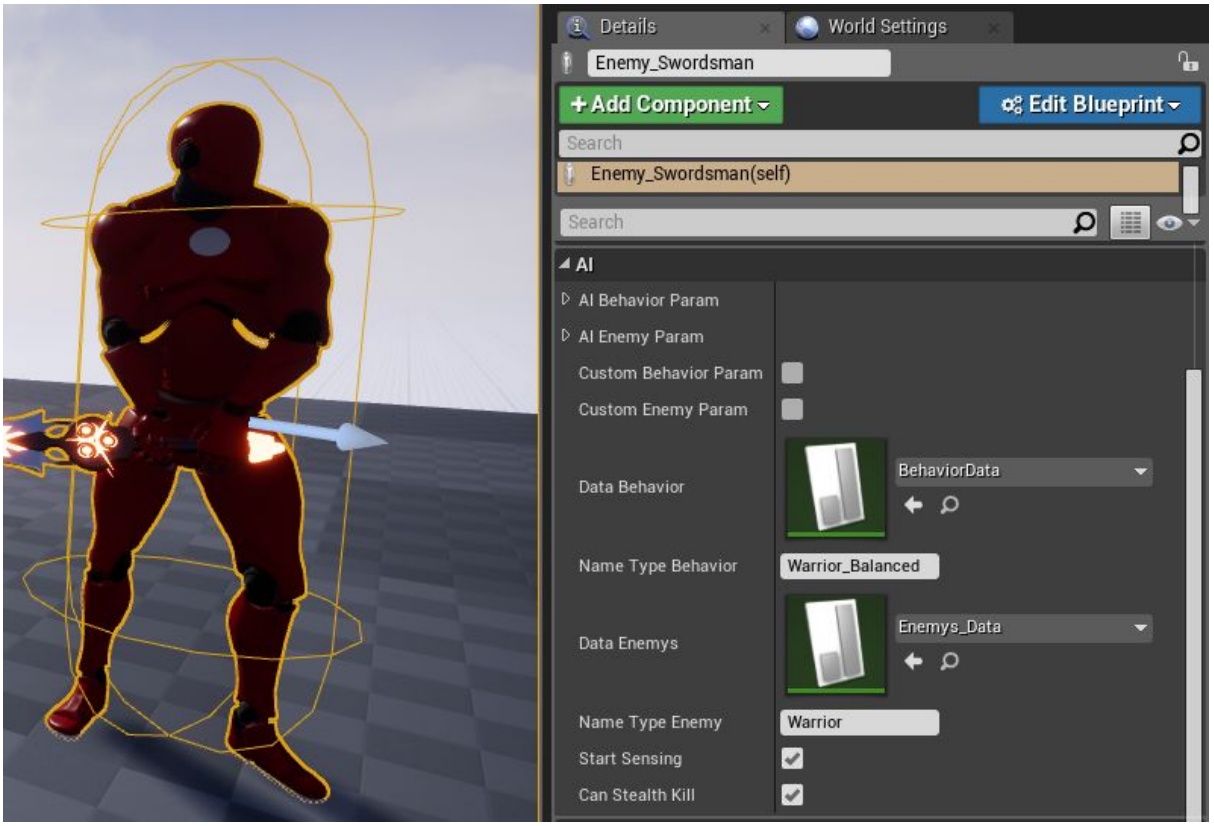


Первым делом стоит указать кто является для персонажа целью и к какой группе он относится. По умолчанию целью для персонажа является группа “**PlayerWarrior**” к которой по умолчанию относится игрок, так же персонаж относится к группе “**PlayerEnemy**” враждебная для игрока группа. Очень важно учитывать данные группы так как наведение на цель у игрока происходит по названию тегов у актора. Теги

назначаются на ивент beginplay посредством выше описанной функции “Target Tags”.



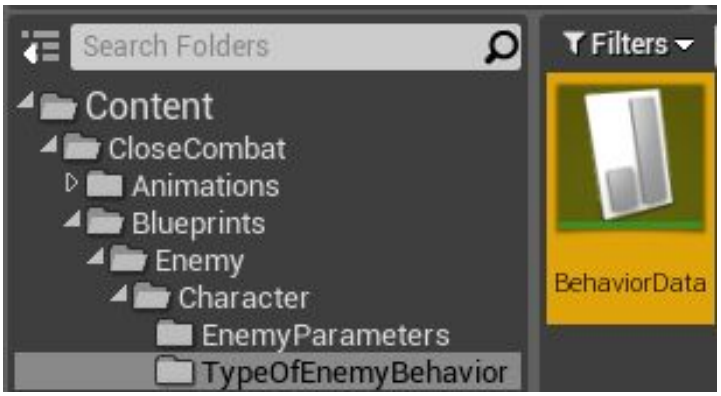
Указывать можно сразу несколько групп тем самым вы можете создавать произвольное количество враждебных или союзных групп.
Следующим шагом рассмотрим настройки “AI”.



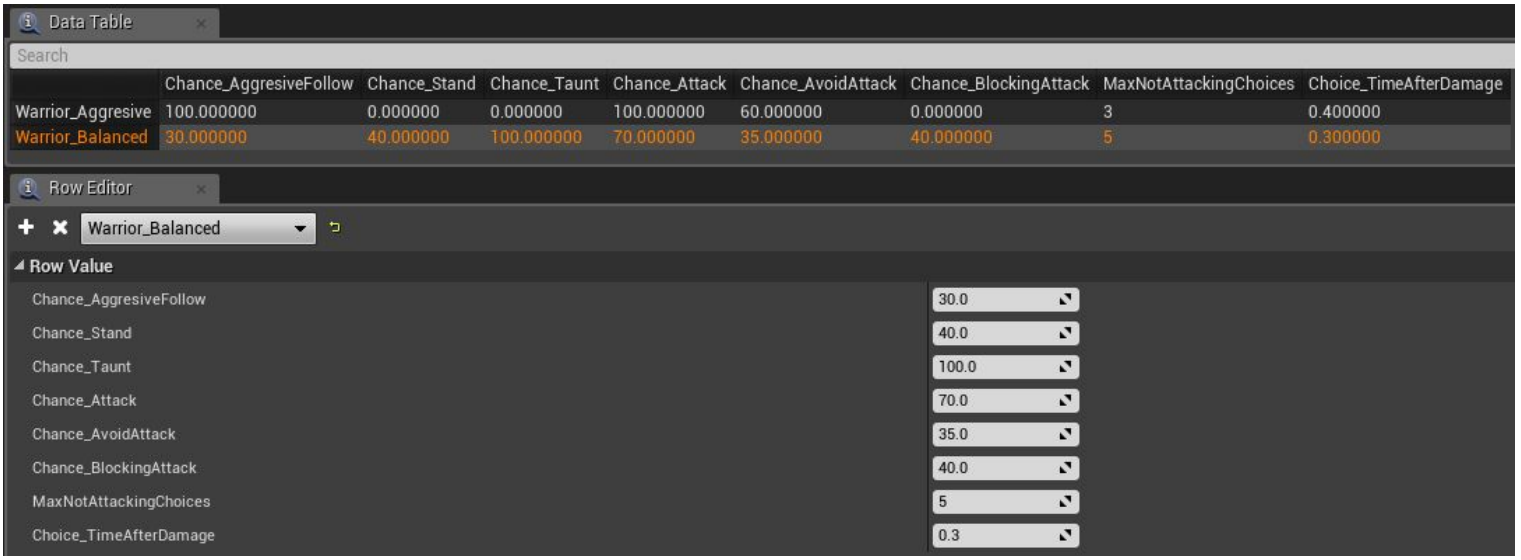
В данной рубрике находятся основные настройки касательно поведения в бою а также параметры персонажа такие как здоровье, количество урона проходящий через блок, количество урона проходящий через уворот.
Start Sensing - Будет ли персонаж изначально видеть и слышать.
Can Stealth Kill - Можно ли убить данного актора при помощи скила “StealthKill”
Custom Behavior Param - Не использовать таблицу а указать параметры вручную. После включения этого параметра раскройте список *AI Behavior Param* и укажите параметры поведения персонажа.
Custom Enemy Param - Не использовать таблицу а указать параметры вручную. После включения этого параметра раскройте список *AI Enemy Param* и укажите параметры персонажа.

Для удобства заполнения и хранения данных о поведении персонажей в бою в проекте используются Data table.
Персонаж при нахождении цели и находясь от нее на дистанции ведения боя (по умолчанию 6 метров) принимает решение каждые 0.3 секунды в процентном (0-100) соотношении от указанных параметров в структуре **AI_BehaviorParam** которой

назначаются данные из таблицы или данные заполняются вручную (исходя из настроек представленных выше).
Откроем таблицу и разберем настройки поведения поведения в бою.



По умолчанию в проекте есть 2 заготовленных типа поведения.
Warrior_Balanced - сбалансированный характер по обороне и атаке.
Warrior_Aggresive - агрессивное поведение настроенное на постоянную атаку
У вас может быть любое кол-во заготовленных таблиц которые вы можете указывать как изначально так и менять динамически при помощи ивентов в персонаже **ChangeBehavior** для поведения и **ChangeEnemyParam** для параметров.



Описание параметров поведения:

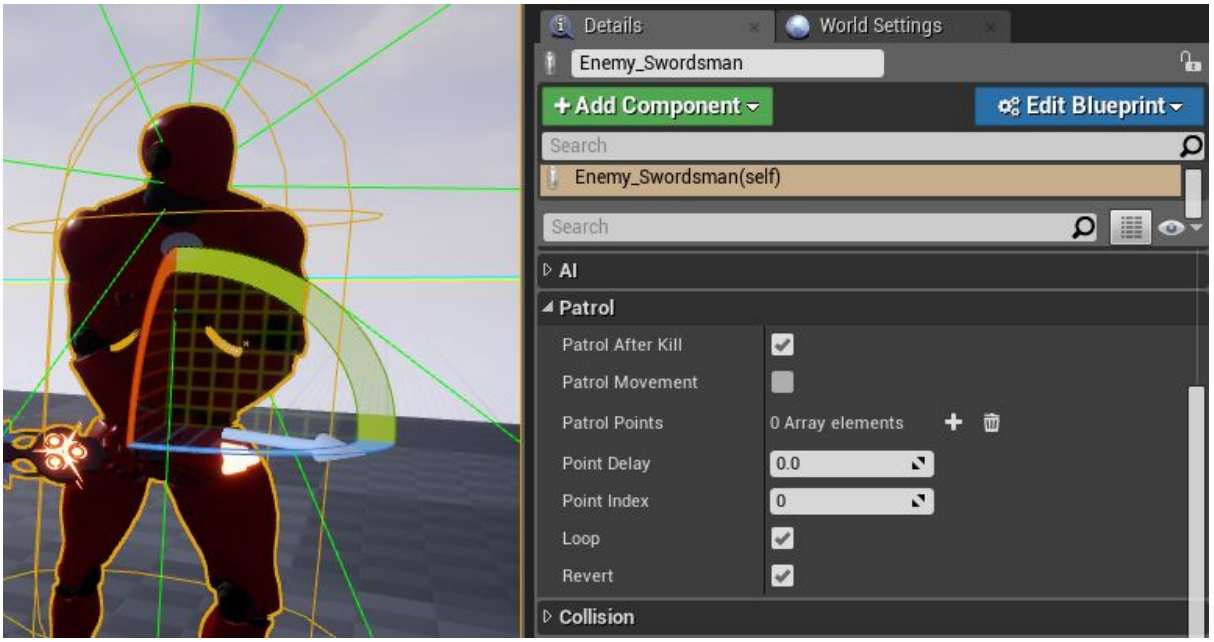
Все параметры задаются в сто процентном соотношении. На примере первого параметра агрессивного направления - данный параметр составляет 30% шанса принятия решения на сближение с целью (30/100) решение принимается каждые 0.3 секунды.
Chance_AggresiveFollow - шанс агрессивного шага в сторону цели с последующей атакой.
Chance_Stand - шанс остановится.
Chance_Taunt - шанс дразнить цель после остановки.
Chance_Attack - шанс начать атаку если до цели 3 метра, если шанс не срабатывает персонаж начинает короткую перебежку.

Chance_AvoidAttack - шанс увернуться от атаки используя кувырок.
Chance_BlockingAttack - шанс использования блока.
MaxNotAttackingChoices - максимальное кол-во принятия решений не связанных с атакой. По истечению принятых решений не связанных с атакой персонаж начнет сближение с целью и попытается атаковать после атаки кол-во решений сбрасывается.
Choice_TimeAfterDamage - после получения повреждения через данное время персонаж снова начнет принимать решения.

Исходя из данных параметров персонаж принимает решения в бою. Для более детального разбора ознакомьтесь с комментариями в blueprint - каждая часть кода имеет подробные комментарии.

Патрулирование

Персонаж может патрулировать различные области уровня. Патрулирование строится на указанных ключевых точках. Все параметры патрулирования задаются в соответствующей рубрике в деталях персонажа.



Patrol After Kill - после убийства цели персонаж продолжит или нет патрулирование.
Patrol Movement - будет ли данный персонаж патрулировать.
Patrol Points - Ключевые точки патрулирования содержат координаты, первой точкой указывайте текущее положение персонажа, кол-во точек может быть произвольным.
Point Delay - Сколько времени персонаж будет находится в ключевых точках.
Point index - Текущая точка в которую пойдет персонаж (индекс массива точек).
Loop - Персонаж по достижении последней точки вернется в первую и продолжит патрулирование.
Revert - По достижении последней точки персонаж начнет возвращаться к первой точке через предыдущие точки.
Параметр XP содержит кол-во опыта за смерть данного персонажа.

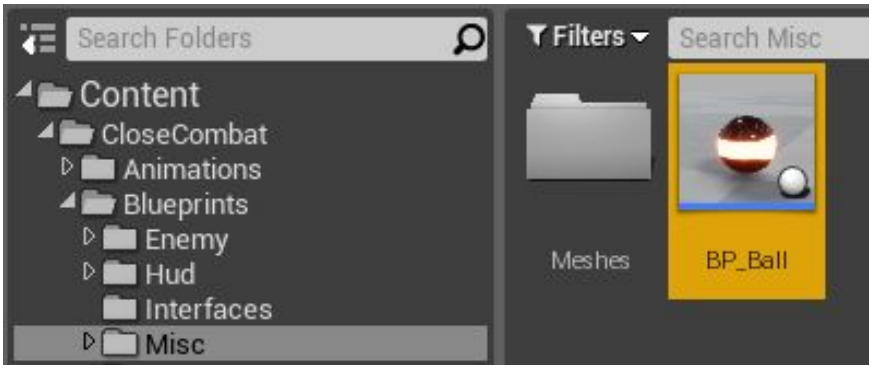
Спавн врагов через blueprint.

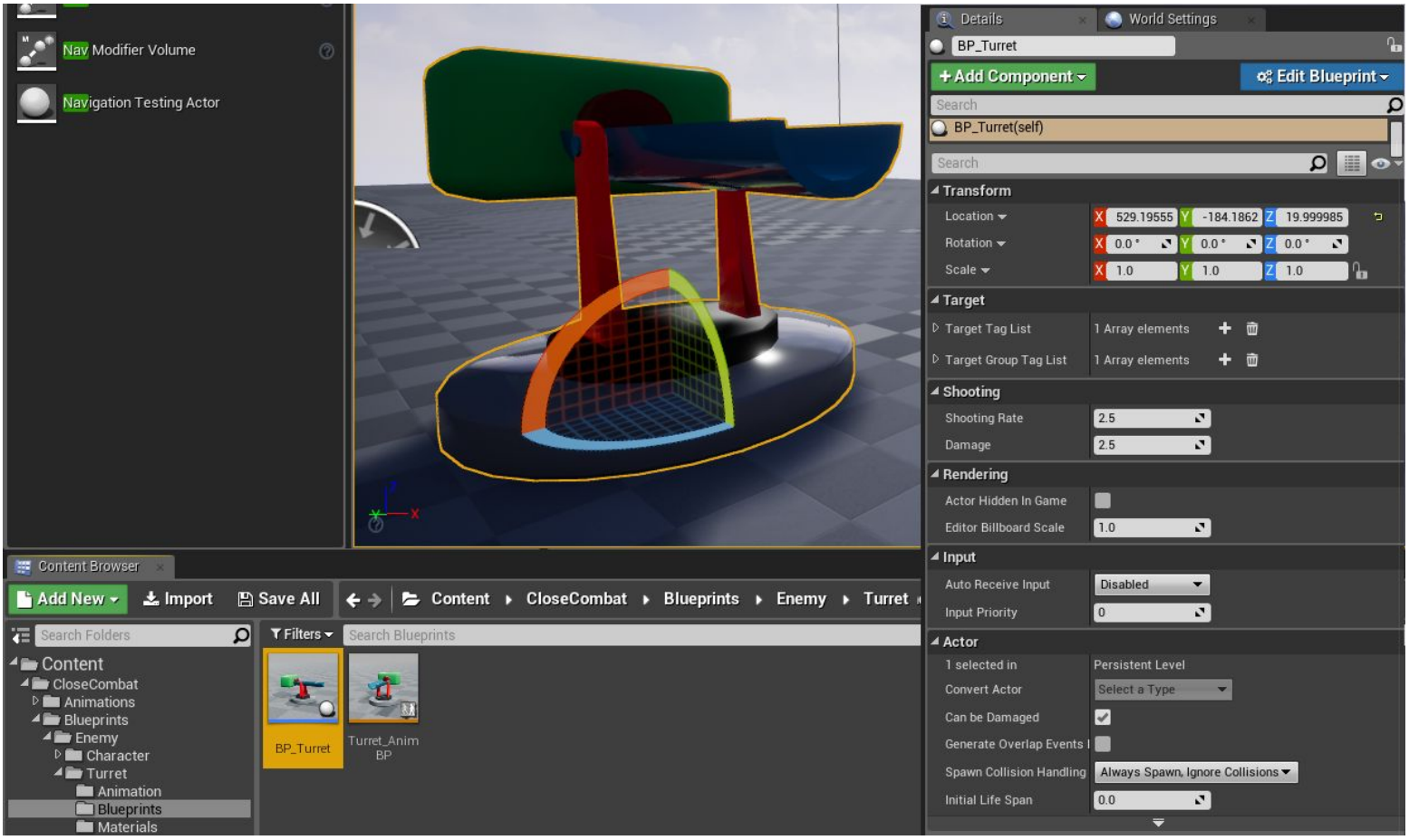
Просто используете узел “**Spawn actor from class**” и при выборе класса “**Enemy_Swordsman**” появятся все выше описанные настройки. Обязательно укажите **Spawn Transform** - место на сцене где появится противник.

Так вы можете настроить параметры атаки в категории “**Attack**”, такие как:
AttackDelayMin - минимальная задержка между атаками.
AttackDelayMax - максимальная задержка между атаками.
AttackTimeRotation - Скорость поворота к цели при атаке.
AttackTimeLookAtTarget - Через какое время после поворота персонаж перестанет поворачиваться к цели. (Может быть полезно для захода в спину..)

Турель

Выше описанное касается и размещения на уровне турели. Турель стреляет шариками с параметрами снаряда.



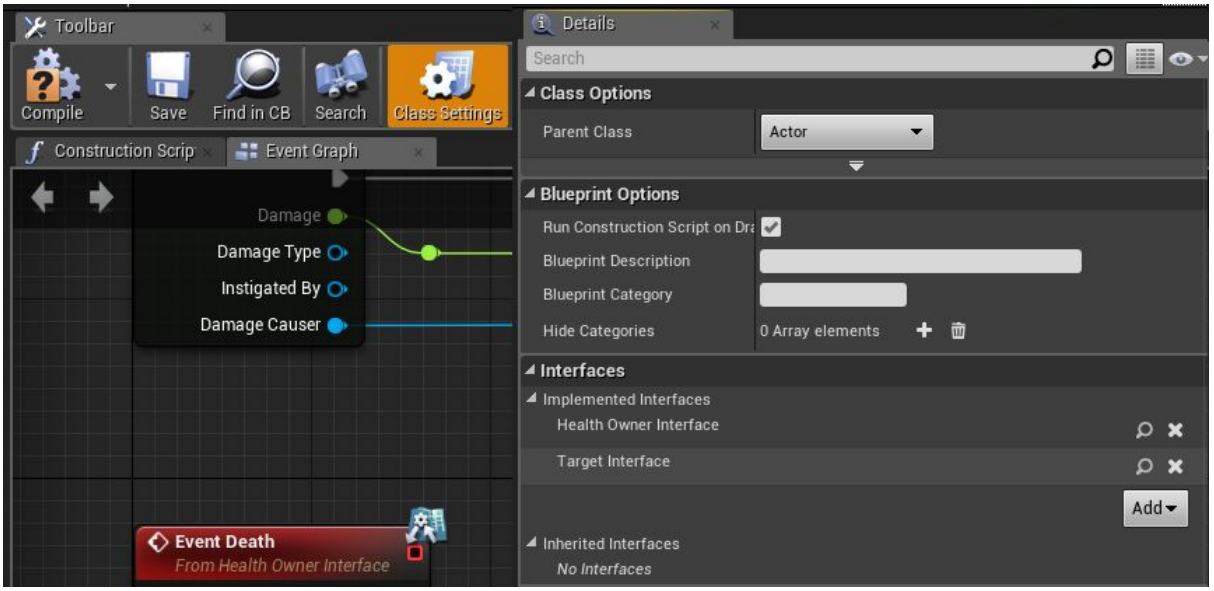


Параметры **Target:**
Target Tag List - Имена групп целей для турели
Target Group Tag List - Имена групп к которой относится турель

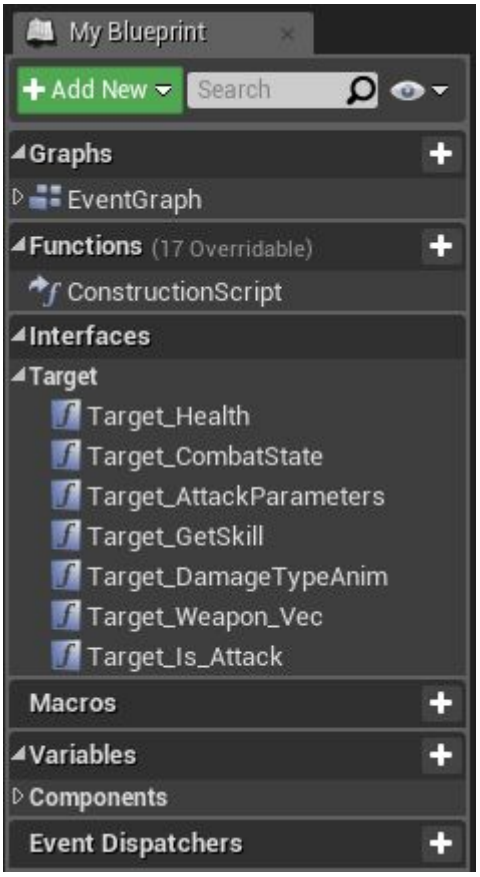
Параметры **Shooting:**
Shooting Rate - Скорость стрельбы
Damage - Урон от стрельбы

Создание нового врага.

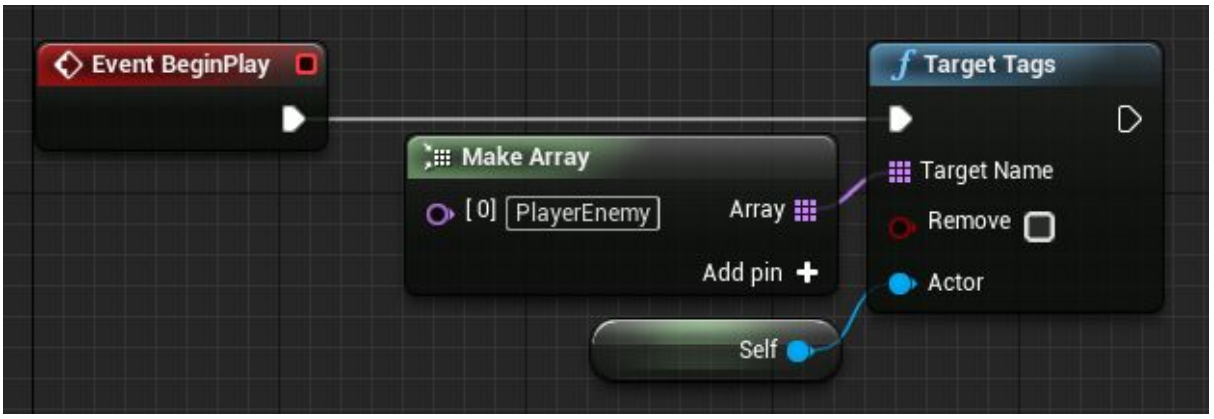
Разберем пример создания нового противника для ближнего боя. Возьмем актор куб который мы сделали в пункте “***Как использовать компонент здоровья***” или создадим его по инструкции из этого пункта.
Взаимодействие между целями также происходит при помощи интерфейсов, поэтому первым делом подключим к нашему новому противнику интерфейс “**Target_Interface**”.



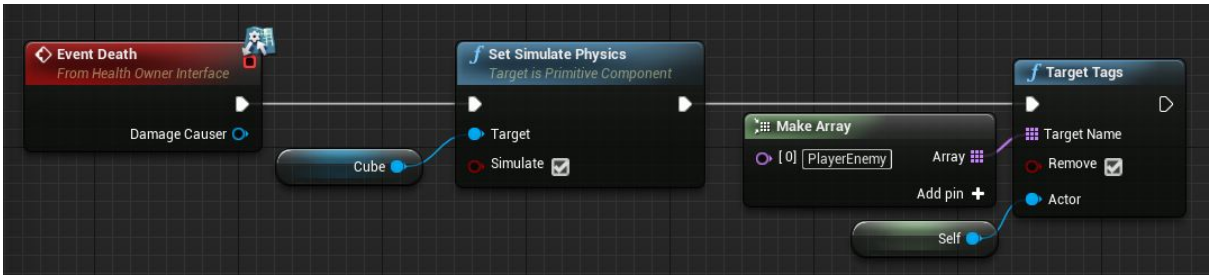
После добавления интерфейса у вас появится категория “Target” во вкладке interfaces.



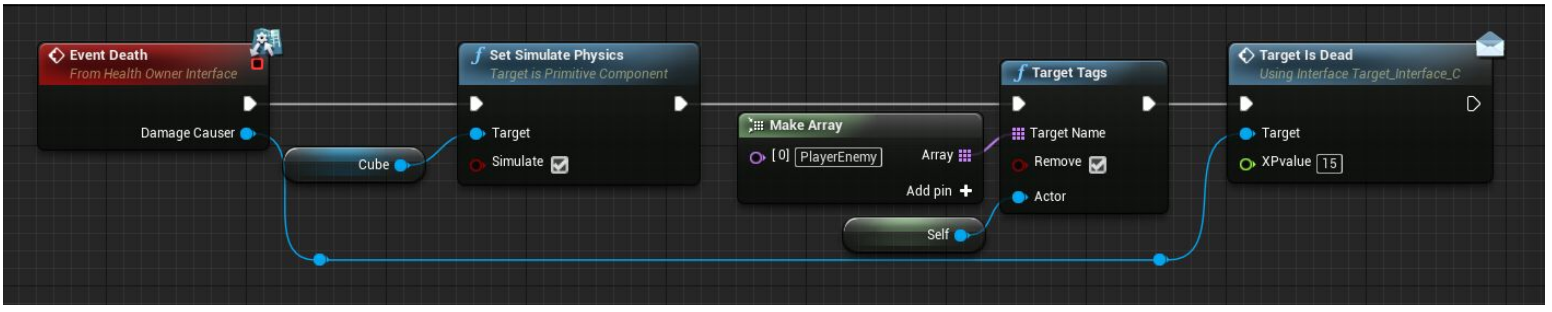
Теперь укажем нашему актору группу к которой он относится, для этого используя функцию “**TargetTags**” укажем группу PlayerEnemy (по умолчанию враждебная для игрока группа).



Следующим шагом будет передача параметров здоровья через интерфейс (чтобы атакующий знал о здоровье цели). Откройте функцию “Target_Health” в категории интерфейсов “Target” и укажите параметры нашего компонента здоровья. Теперь игрок уже сможет наводится на цель и видеть её здоровье.
Далее нужно очистить имена групп при смерти актора, чтобы после смерти цели игрок не мог на нее наводится. Для это копируйте узлы с добавлением группы, но укажите параметр Remove (убрать).



Следующим шагом будет передача информации о смерти актора. Для этого просто передайте интерфейс атакующему “Target is dead” на ивент смерти.

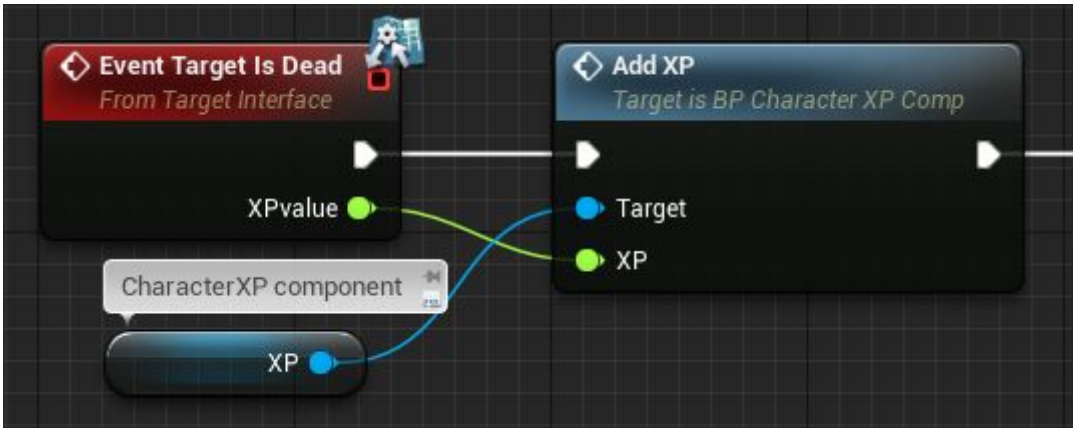


Так же можете указать получаемый опыт за убийство данного актора. На этом все наш простой актор готов на который персонажи могут наводится, убивать и получать за это опыт. Для более сложных и продвинутых противников читайте комментарии в коде у актора врага “**Enemy_Swordsman**” для ближнего боя и “**BP_Turret**” для дистанционного боя.

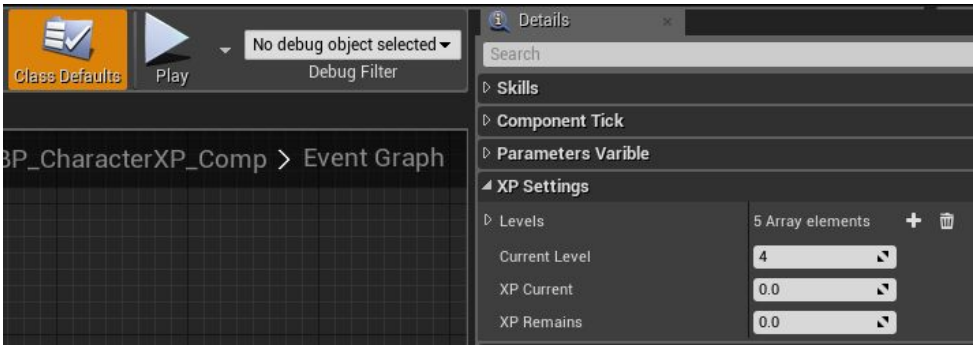
Компонент начисления опыта.

Данный компонент служит для начисления опыта и повышения уровня персонажа. Добавляется точно так же как и компонент здоровья. Осуществляет связь между компонентом и персонажем через интерфейс “**CharacterXP_Interface**”.

Как это работает. Компонент имеет структуру уровня персонажа, которая меняется по индексам (уровням) в массиве всех уровней по набиранию определенного кол-ва опыта. По умолчанию опыт начисляется при убийстве противников, начисляется он посредством ивента “Add_XP”.

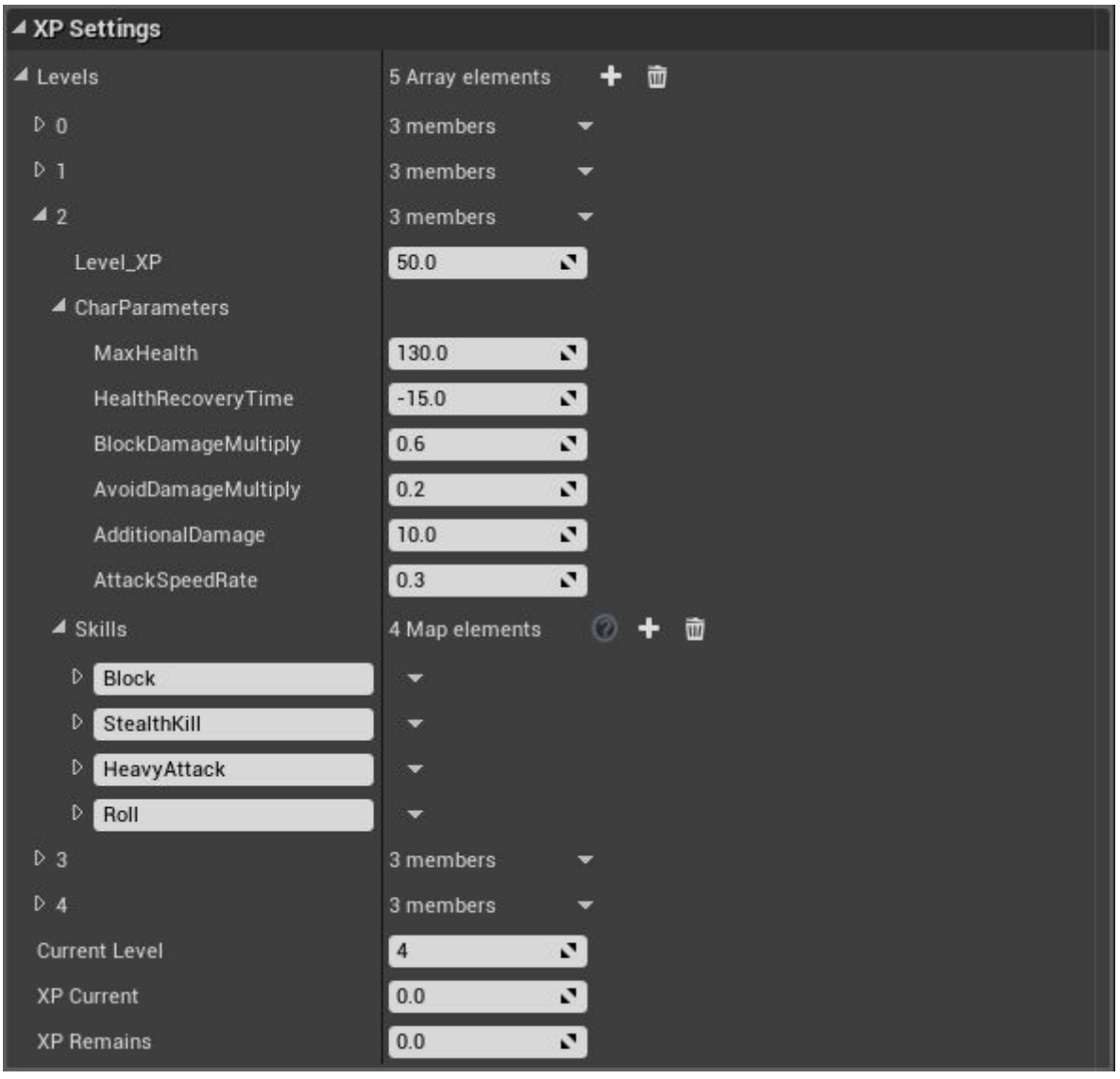


Как на примере созданного нами актора врага при смерти он посылает сообщение интерфейса атакующему о смерти и начислению опыта в свою очередь атакующий на данный ивент передает данные о опыте в компонент “**CharacterXP_Interface**”.
Компонент подсчитывает опыт и если его достаточно для перехода на следующий уровень то происходит повышение уровня персонажа обновляя его параметры из структуры и обновляя данные на игровом интерфейсе (Hud).
Рассмотрим структуру уровня для этого откройте компонент “**CharacterXP_Interface**” и перейдите в “**Class Defaults**”, нас интересует только категория “**XP Settings**”.

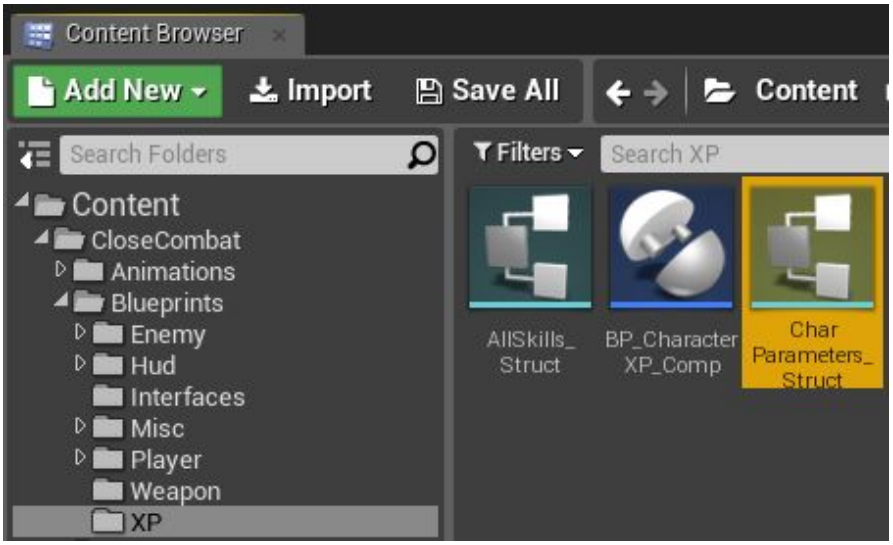


Levels - Раскрывающейся список наборов структур на каждый уровень персонажа.
Current Level - текущий уровень персонажа. Укажите 1 чтобы начать с первого уровня.
XP Current и XP Remains оставляйте в нолях. Это переменные для расчета опыта

Структура уровня:

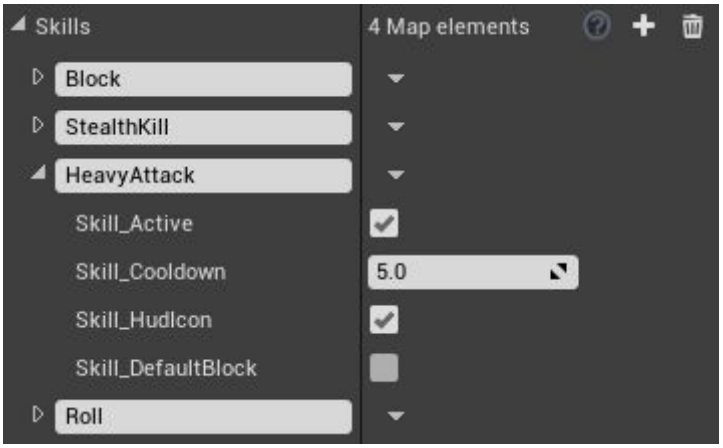


На данном примере разберем 2ой уровень персонажа, вы можете создавать любое кол-во уровней которое вам необходимо.
Level XP - опыт который необходим чтобы перейти на 3ий уровень.
CharParameters - структура параметров персонажа на данном (2ом) уровне тут хранится информация сколько должно быть здоровья у персонажа, на сколько меньше ему нужно времени, чтобы восстановить полностью здоровье, сколько поглощает урона блок, увеличения повреждения от оружия и прочее. У вас могут быть дополнительные параметры просто расширяйте структуру “**CharParameters**”.



Данная структура и остальные параметры уровня передается при повышении уровня в ивенте “**Add_XP**” персонажу посредством интерфейсов. (читайте комментарии в коде).

Skills - Массив map структур скилов персонажа на данном уровне.



На примере скилла HeavyAttack:

Skill_Active - активный ли скил изначально.

Skill Cooldown - время восстановления скилла.

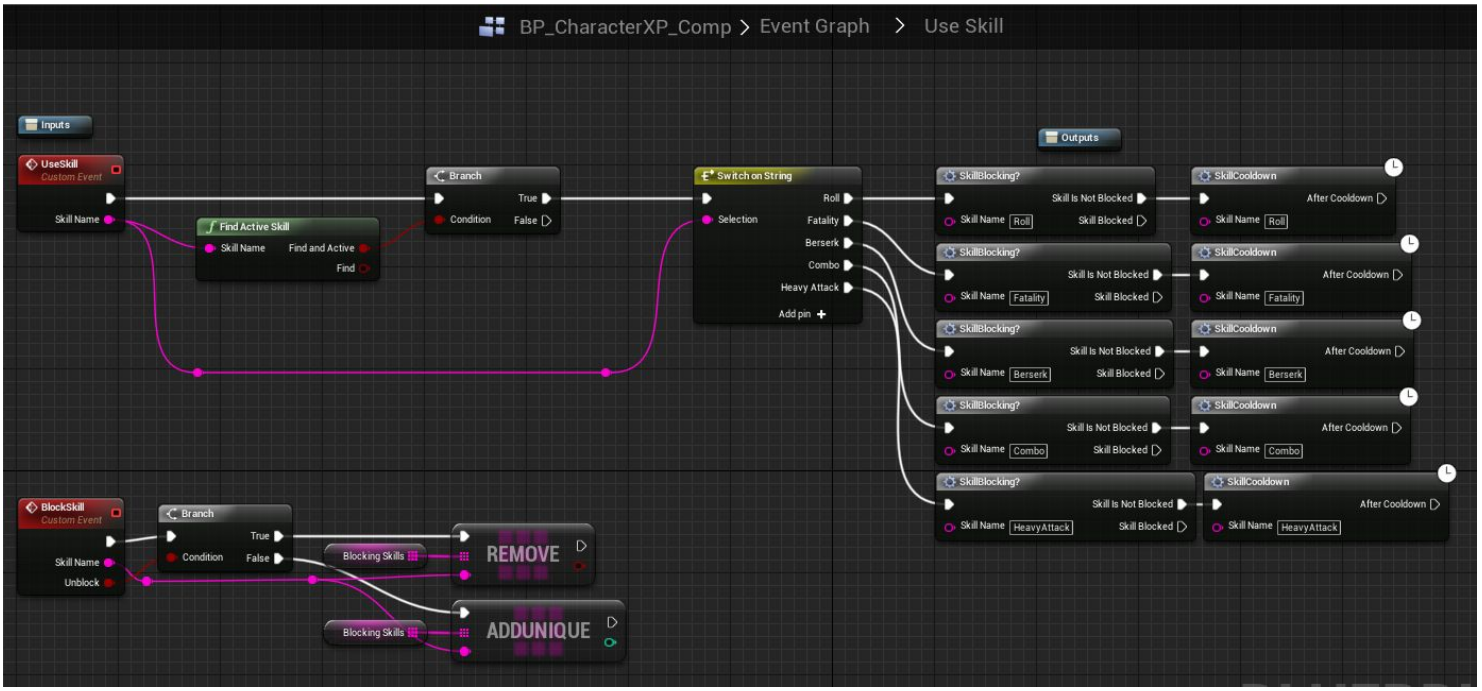
Skill_HudIcon - имеет ли данный скил визуальное представление на игровом интерфейсе.

Skill_DefaultBlock - заблокирован ли изначально данный скил. Этот параметр служит для переключаемых скилов на подобии фаталити. Фаталити срабатывает только тогда когда вы его активировали и нанесли последний удар цели. Поэтому фаталити изначально заблокирован и разблокируется с кнопки скила (по умолчанию 2).

По умолчанию это все параметры которые передаются по уровням персонажа. Добавив дополнительные параметры которые вам необходимы, заполните данный массив на все уровни персонажа.

Скиллы персонажа.

Вы можете использовать различные умения персонажа и для этого в первую очередь нужно их указать в компоненте начисления опыта персонажа “**CharacterXP**”. По умолчанию скиллы добавляются в ивенте “**Use Skill**”.

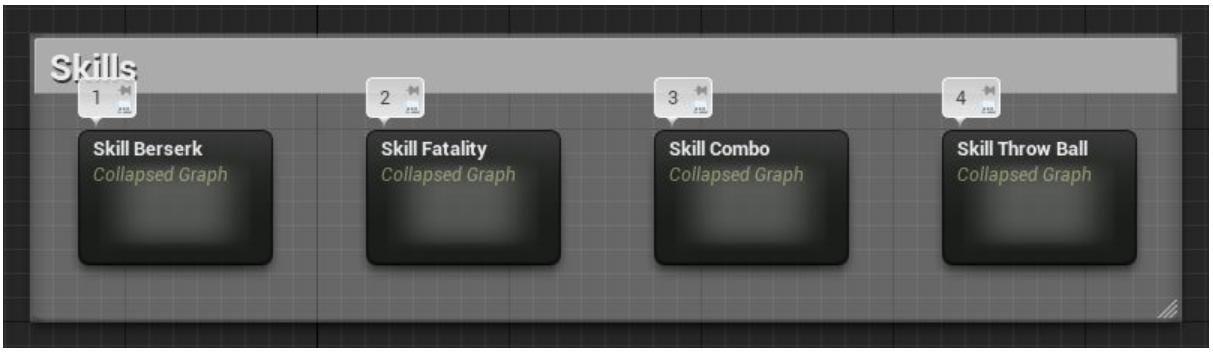


Ваш новый добавленный скил должен иметь неизменное имя - добавляйте необходимые скиллы по аналогии присутствующих скиллов.

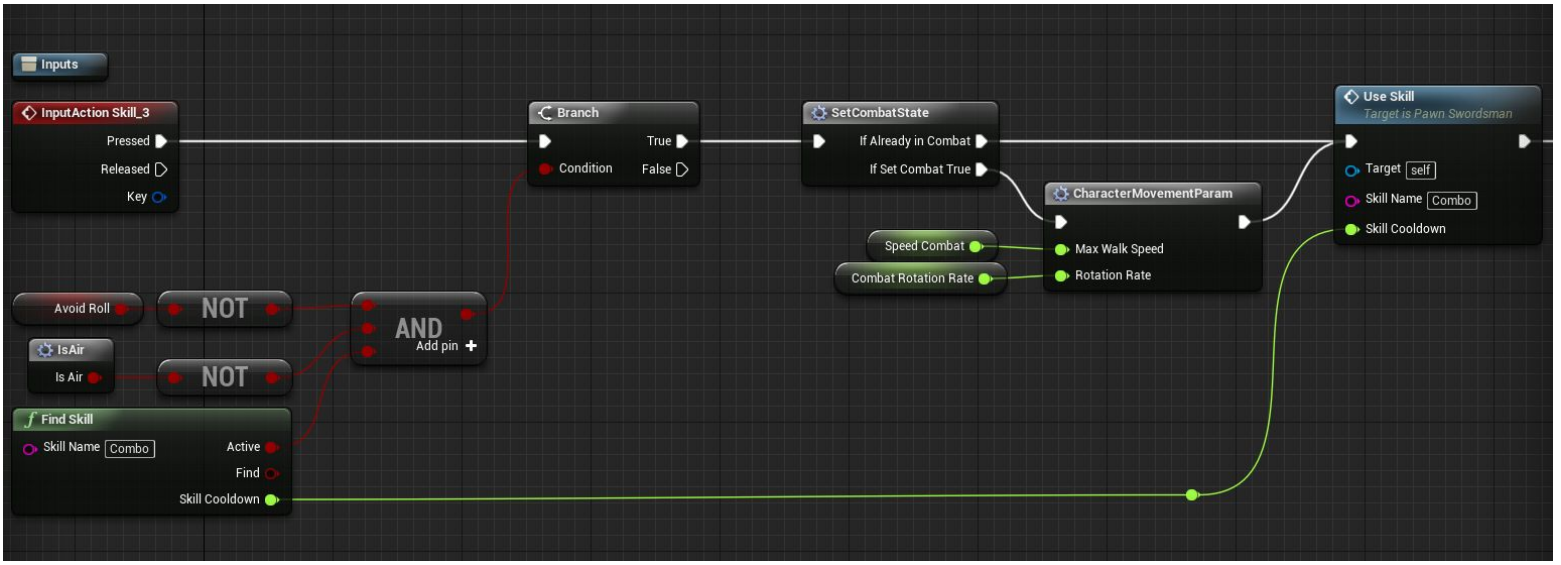
Типы скиллов

По умолчанию скиллы могут быть 3ех типов:
Активные - скилы которые срабатывают по нажатию кнопки.
Пассивные - скиллы которые для запуска просто должны присутствовать в структуре уровня.
Переключаемые - скиллы которые срабатывают только в том случае если их предварительно активировали переключателем.

Рассмотрим все 3 типа на примере готовых скиллов.
Активный скил. Серия из 5ти последовательных ударов “**Combo**” является активным скилом и располагается на клавише 3.



При нажатии на данную клавишу происходит поиск данного скила (вместе с дополнительными условиями) в компоненте “**Chatacter_XP**” у игрока он переименован в короткое название **XP**. Для поиска скилла написана функция “**Find Skill**”.



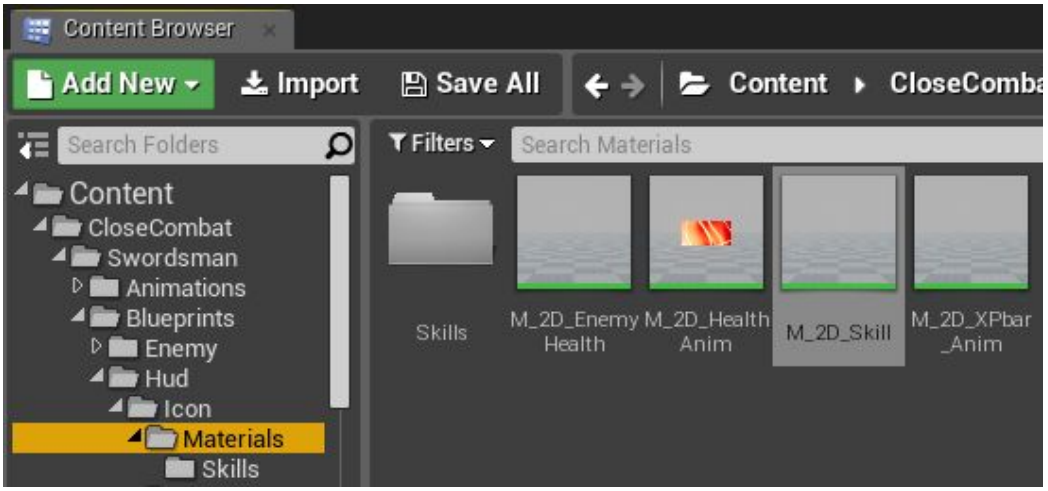
Если данный скилл найден и активен (не находится в данный момент в состоянии восстановления) то происходит его запуск посредством функции “**Use Skill**”. Данная функция запускает состояние восстановления скила в компоненте опыта и также запускает анимацию на игровом интерфейсе. Сама логика скила идет сразу после функции “**Use Skill**” это запуск комбо анимации и различные эффекты.

Пассивный скил. Скрытое убийство со спины не нуждается в кулдауне и поэтому является пассивным скилом. При наведении на цель и сокращению с нею дистанции идет поиск скилла “**StealthKill**” и если таковой присутствует в структуре уровня то появляется возможность убить противника со спины.

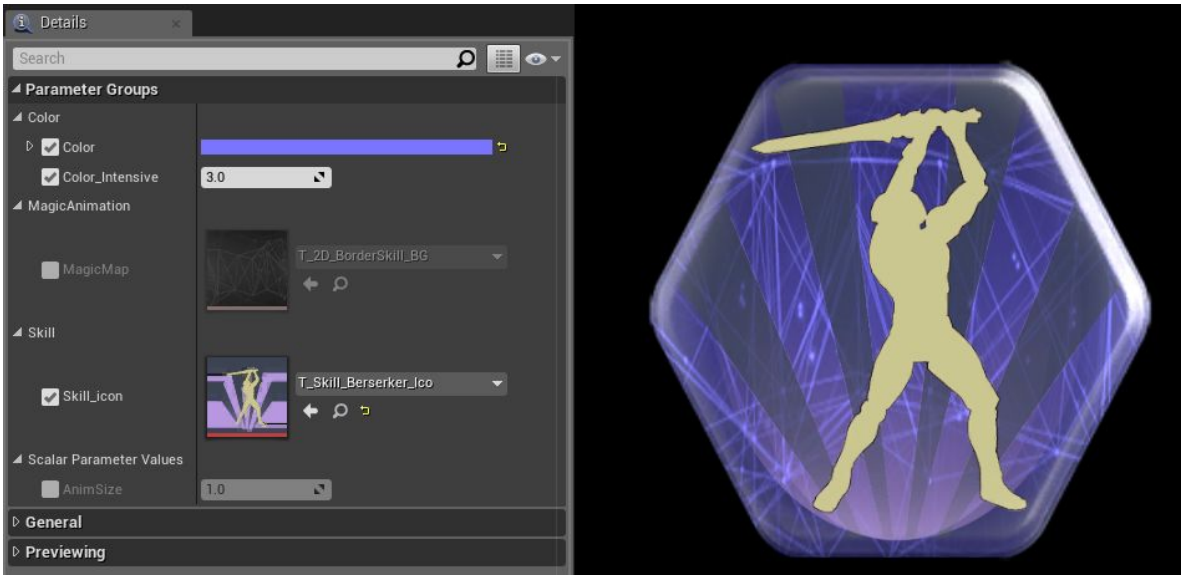
Переключаемый скил. Фаталити имеет и кулдаун и включенное/выключенное состояние. Для этого используется блокировка скила. Фаталити работает по следующему принципу - если данный скил имеется в структуре уровня и он не блокирован, а так же активен (не в состоянии восстановления) то при последнем ударе по противнику активируется добивание. По умолчанию блокирование/разблокирование происходит по нажатию клавиши 2.

Визуальный вид скиллов.

Для всех скилов присутствует унифицированная визуальная схема в виде материала “**M_2D_Skill**”.

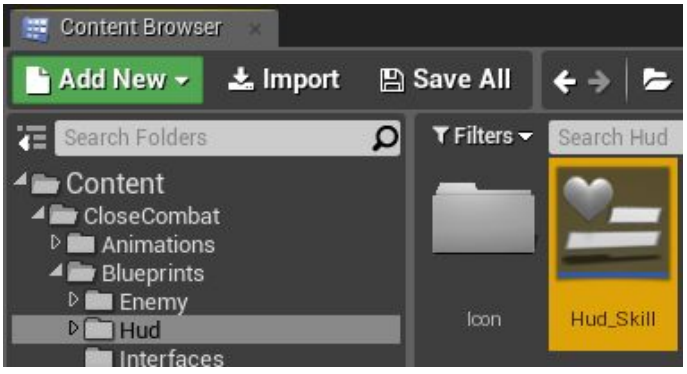


В папки Skills лежат инстансы для каждого скилла. Чтобы создать новую иконку для ваших скилов нажмите правой кнопкой мыши на материал “**M_2D_Skill**” далее Create Material instance. Вы создадите инстанс в котором присутствуют настройки визуального отображения скила.



Color - Цвет фона материала.
Color Intensive - интенсивность цвета (яркость).
MagicMap - Текстура для фоновой анимации.
Skill_Icon - Иконка умения на переднем фоне.
AnimSize - Параметр служит для заполнения фона при восстановлении умения.
Демонстарция материала.

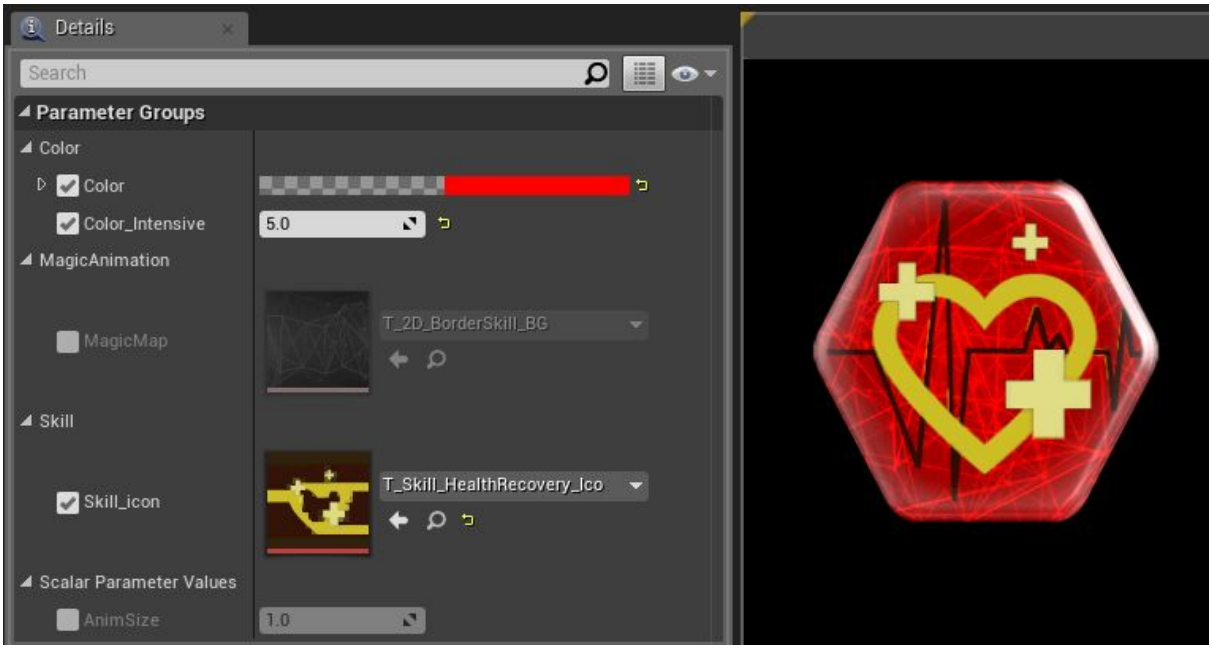
После настройки внешнего вида вашего скилла его нужно внести в базу внешнего вида скилов в виджете “**Hud_Skill**”.



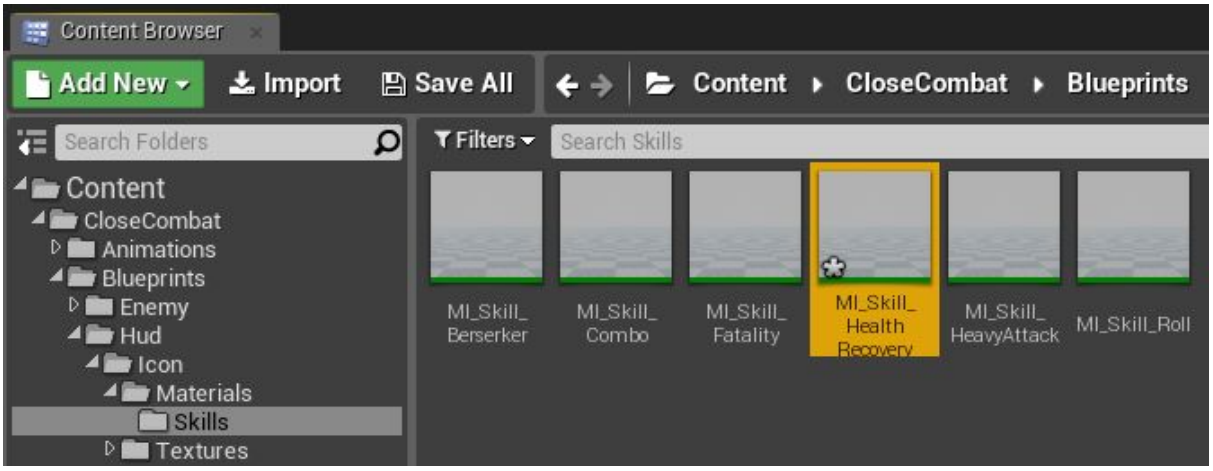
Откройте “**Hud_Skill**” выделите map переменную “**Skill_Visual**” и добавьте новый индекс. Слева нужно указать имя скилла (точно такое же имя которое будет указано в компоненте начисления опыта), а справа укажите ваш созданный материал. Теперь ваша новая иконка в базе и при добавлении скила с одноименным названием на игровом интерфейсе будет отображаться ваше умение.

Создание нового скила.

Рассмотрим создание нового умения. Создадим скилл, который полностью восстанавливает здоровье и восстанавливается через 20 секунд после использования. Первым делом создадим его визуальный вид (material_instance) как описано чуть выше. У меня получилось вот так (иконка присутствует в проекте):

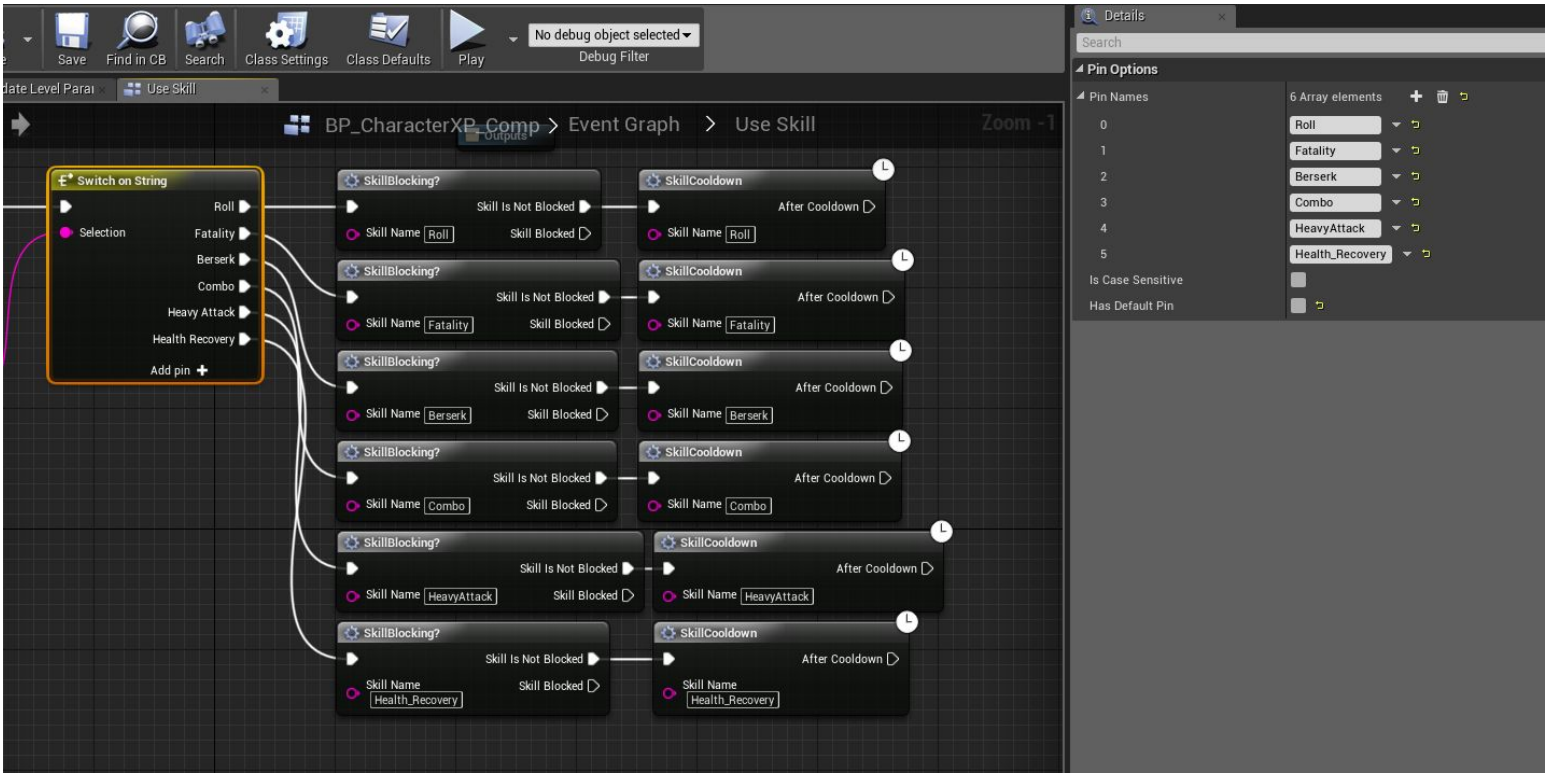


Для единой структуры поместите ваш материал рядом с остальными материалами умений.



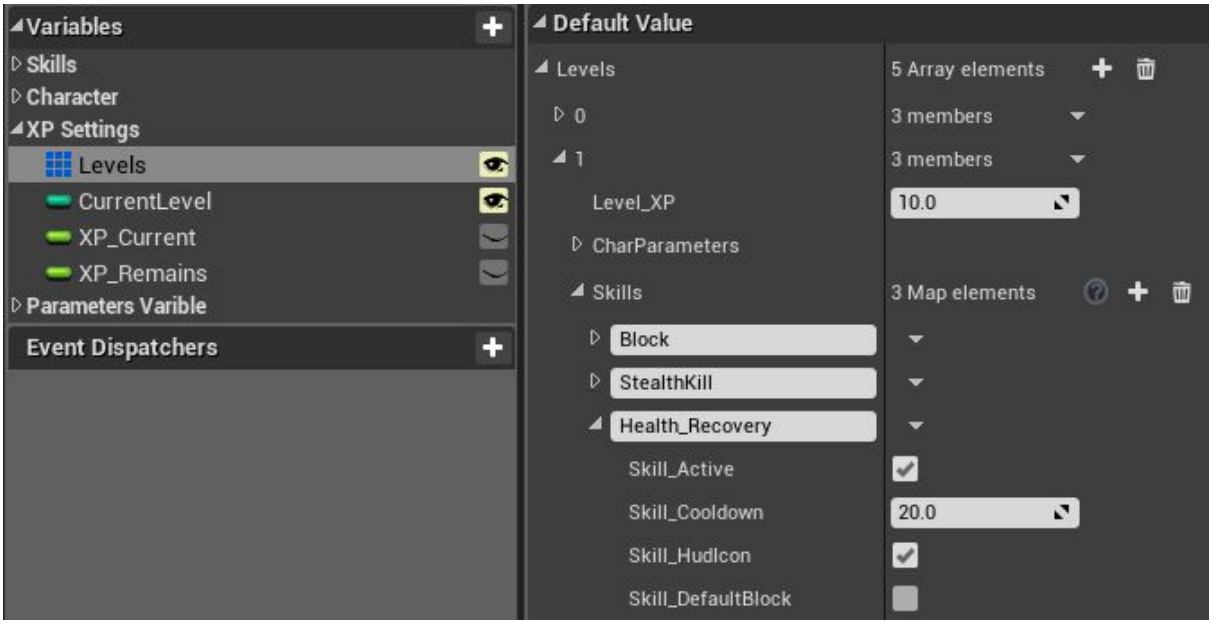
Далее добавим наш материал в список внешнего вида скилов в виджете “**Hud_Skill**” как описано выше. Название скилла укажите “**Health_Recovery**”.

Далее добавим наш скил в компонент “**CharacterXP**”. Перейдите в ивент “**Use_Skill**” и по аналогии с остальными умениями добавьте “**Health_Recovery**”



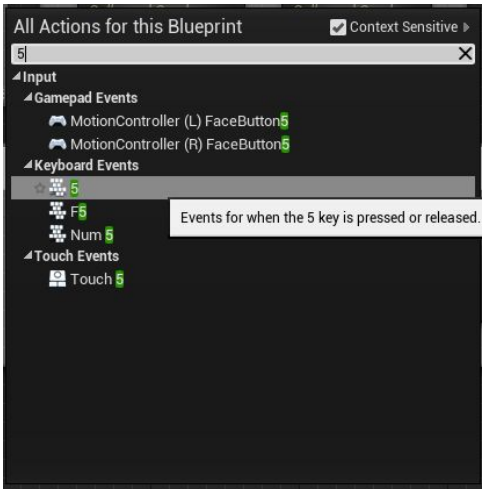
У вас должно получится как на картинке выше. В самом низу после “**HeavyAttack**”.

Далее нужно указать на каком уровне у нас появится данный скил и какие его настройки, для этого перейдите в **Class defaults** или выделите переменную “**Levels**”. перейдите в 1ый индекс (первый уровень), далее раскройте список скиллов и добавьте туда наш “**Health_Recovery**”. Должно получится вот так:

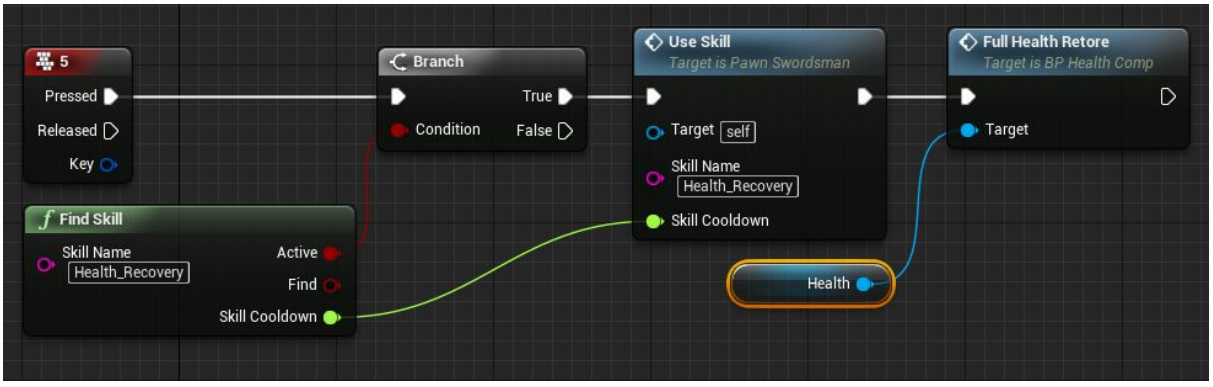


Настройки нашего умения:
Skill_Active - включен, так как скилл должен быть активным изначально.
Skill_Cooldown - 20 секунд на восстановление скилла как и задумывалось.
Skill_HudIcon - включен, так как скилл активный и имеет иконку.
Skill_DefaultBlock - выключен, так как скилл не нуждается в блокировке.

Теперь напишем саму логику умения восстановления здоровья. Переходим в персонажа “**Pawn_Swordsman**” и добавляем инпут ивент клавиши 5.



На данный ивент пишем следующую логику



Поэтапно происходит следующее:
Ищем скил с название “**Health_Recovery**” и если он найден и активен (переменная Active будет выдаваться только в том случае если скилл найден, дополнительно проверять на Find не нужно) то используется скилл при помощи функции “Use Skill” (данная функция запускает восстановление скила и анимации игрового интерфейса), далее за функцией Use skill идет непосредственно восстановление здоровья. Ивент “**Full Health Restore**” в компоненте здоровья полностью восстанавливает здоровье.

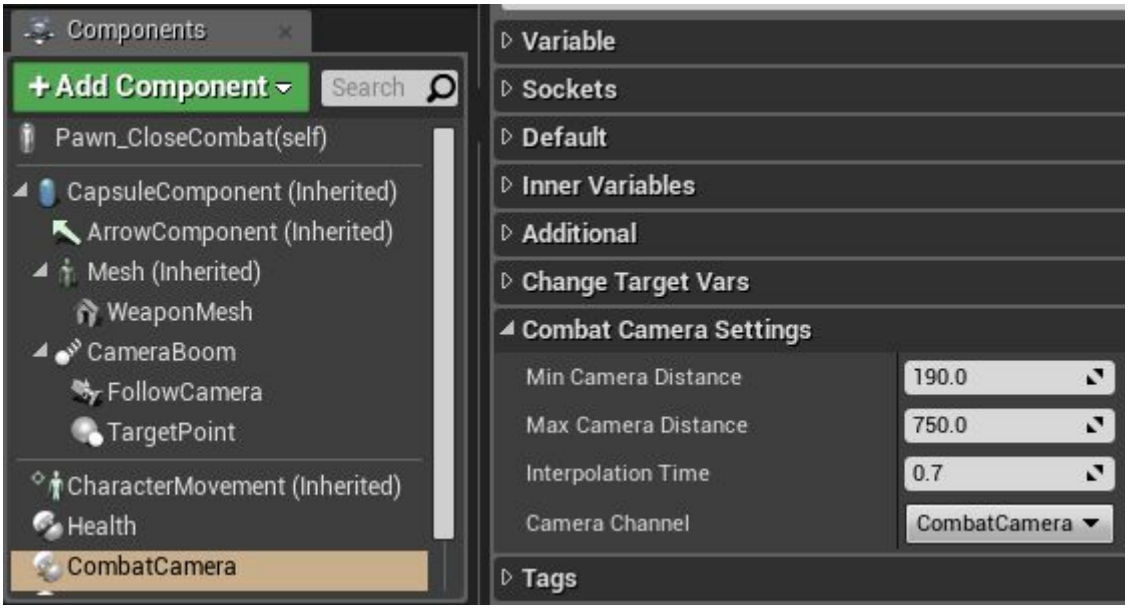
Откомпильте “**Pawn_Swordsman**” запустите игру и вы увидите новый активный скилл восстановления здоровья который активируется на клавишу 5. Все остальные скилы работают по такому же принципу.

Боевая камера.

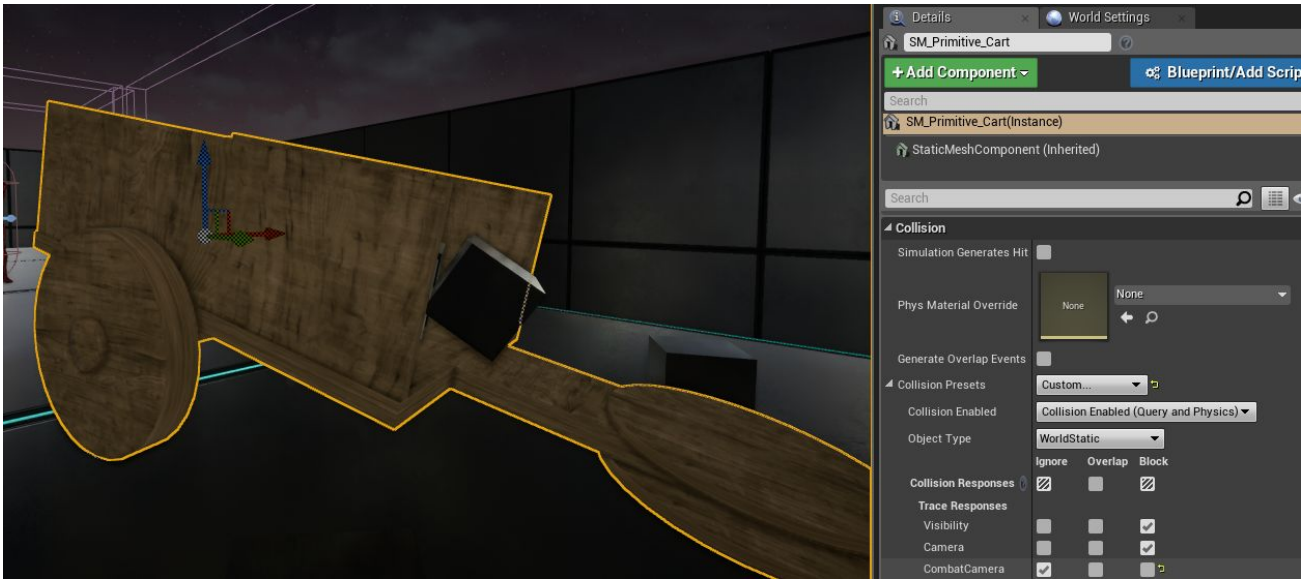
Боевая камера служит для дополнительного обзора между двумя целями и по умолчанию может использоваться если персонаж наведен на цель. Данная камера захватывает две цели на экране. Демонстрация.

Настройки камеры.

Рассмотрим настройки камеры. Выделите компонент боевой камеры и во вкладке “**Details**” нужна только категория “**Combat Camera Settings**”:

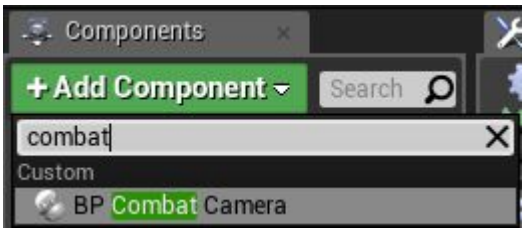


Min Camera Distance - Расстояние камеры до центра между целями их минимальном расстоянии.
Max Camera Distance - Расстояние камеры до центра между целями их максимальном расстоянии. (Максимальное расстояние 7.5 метров)
Interpolation Time - Время плавного перехода камеры из боевого в базовый режимы.
Camera Channel - Канал для камеры. По умолчанию создан дополнительный канал для боевой камеры, он необходим для игнорирования некоторых объектов. Разберем такую ситуацию - у вас на уровне присутствует ящик (столб, некий пропс..) и в бою камера не должна сталкиваться с данным объектом и портить обзор, но вне боя камера должна сталкиваться с данным объектом. Для этого в объекте в каналах коллизии укажите каналу “Combat Camera” игнорировать данный объект.

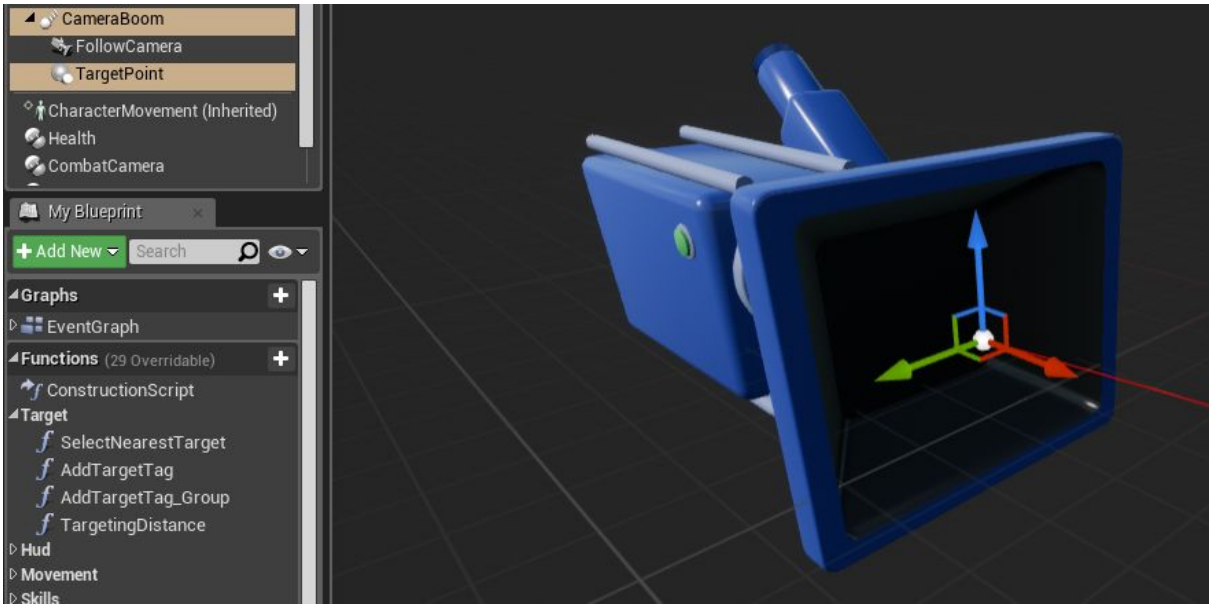


Добавление боевой камеры

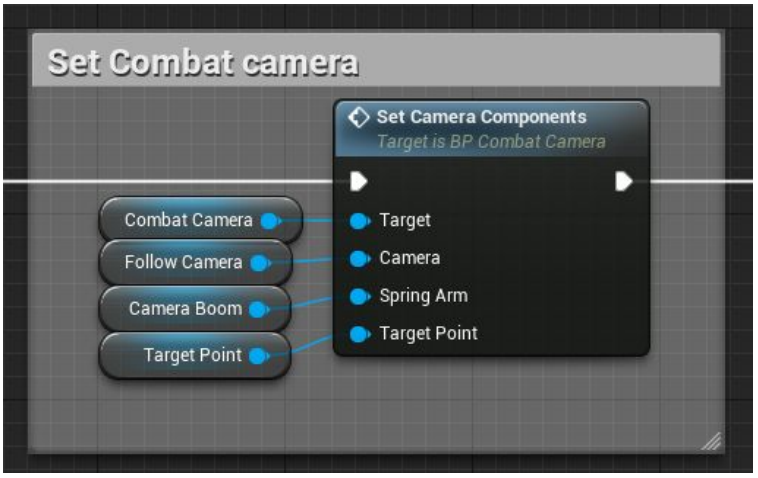
Боевая камера является компонентом и может быть добавлена к любому актору, который использует спринг арм и камеру. Чтобы добавить боевую камеру камеру выберете ее в списке добавляемых компонентов.



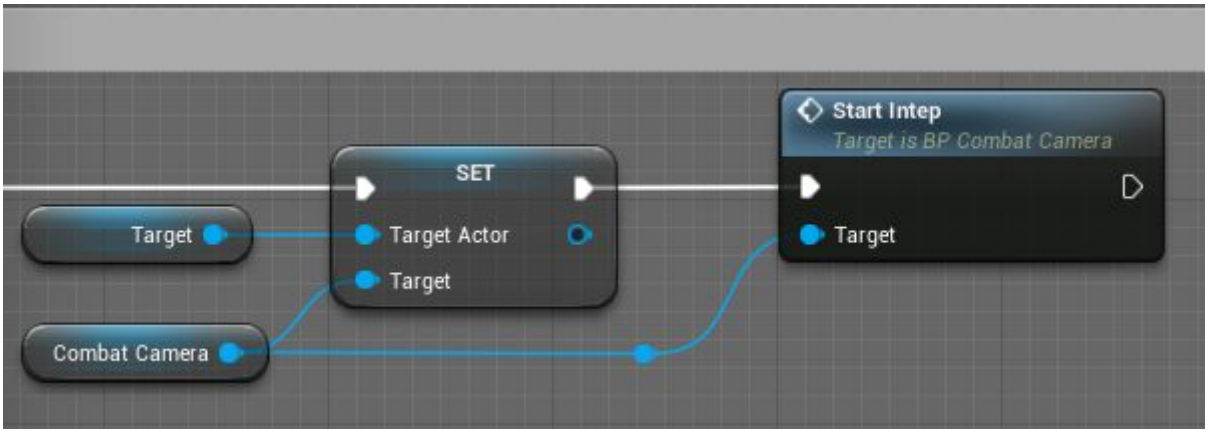
Далее нужно создать дочерний компонент “Scene component” для спринг арма, в проекте данный компонент переименован в “TargetPoint”. Данный компонент будет служить локацией конечной точки спринг арма - это нужно для возврата камеры в базовое положение. Сам компонент должен располагаться в тех же координатах, что и камера.



Последним шагом будет передача информации о компонентах в компонент боевой камеры (функционалу боевой камеры нужно знать с какими компонентами работать). Сделать это лучше по beginplay. Вызовите функцию “Set Camera Components”у компонента боевой камеры и укажите компоненты.



Чтобы боевая камера работала корректно ей нужно передавать ссылку на цель. Сама камера активируется по ивенту “Start Interp”. По умолчанию на кнопку “V”.



Для более детального разбора кода читайте комментарии внутри blueprint классов.
Также при наведении курсором мыши на параметры читайте всплывающие подсказки.

Спасибо за внимание.
С наилучшими пожеланиями ZzGERTzZ.