**4. Apply damage**

1. Создаем новый класс для «наносителя» урона в папочке (+ исправления в одном файле надо сделать из-за этого)

2. Удалить нужно у персонажа кое-что, теперь создаем компонент специальный в наносителе урона и визуализируем сферу – какой ЗФ, функция и что она принимает (5 параметров, остальные – по умолчанию)?

3. Создаем два проперти для сферы

4. Как сделать так, чтобы актор получал урон, когда попадает в нашу сферу? Семейство функций, чем отличаются, почему мы не используем первые, какую используем, что принимает (10 параметров)?

5. Создали еще два проперти для функции урона

6. Как устроена функция нанесения урона? Над чем является оберткой, и как эта функция (над которой обертка) работает: два основных момента

7. Добавляем категориюю логирования и логируем урон, который получает персонаж – в чем отличие для двух сфер с разным девятым параметром? Если добавить еще персонажей на сцену – будут ли получать урон?

1. Создаем актора, который будет наносить ущерб нашему персонажу. Актор будет наносить RadialDamage, данная логика будет имитировать взрыв гранаты рядом с нашим персонажем.

Создаем новый класс, наследуемся от Actor, в папке Dev, назовем STUDevDamageActor.

Добавляем строчку в Build.cs для нашей новой папки.

2. В Tick() функции персонажа удалили урон, наносимый самому себе.

В DevDamage актора добавляем SceneComponent.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Первое, что надо сделать – визуализацию сферы, имитирующей взрыв. Для этого подключаем ЗФ DrawDebugHelpers.h – он позволяет рисовать различные геометрические примитивы.

Рисовать будем каждый фрейм с помощью функции DrawDebugSphere:

Изображение выглядит как текст

Автоматически созданное описание

Первый параметр – указатель на World.

Второй – центр сферы.

Третий – радиус сферы.

Четвертый – количество сегментов сферы (поставили тут любое число, главное, чтобы визуально было понятно, что это сфера).

Пятый – цвет сферы.

Остальные – по умолчанию.

3. Создаем проперты для радиуса и цвета:

Изображение выглядит как текст

Автоматически созданное описание

4. Теперь нам надо сделать так, чтобы, когда актор попадал внутрь сферического объема, количество ее жизней уменьшалось.

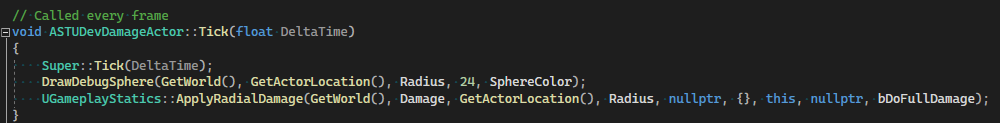
Для начала надо подключить ЗФ Kismet/GameplayStatics.h в нашем DamageActor’е.

Наносить ущерб будем в функции Tick(), в классе GameplayStatics есть семейство методов для нанесения урона:

Изображение выглядит как текст

Автоматически созданное описание

В первых двух функциях нужно знать указатель на актора, которому мы хотим нанести урон, но так как мы коллизии пока не проходили – логику вычисления актора, который находится в сфере, мы пока реализовать не можем, поэтому будем пользоваться функция ApplyRadialDamage() – внутри нее уже реализована данная логика и урон будет наносится всем акторам, которые в модельной сфере.



Первый аргумент функции – WorldContentObject – ну, как обычно.

Второй – урон.

Третий – центр сферы урона.

Четвертый – радиус сферы.

Пятый – тип урона.

Шестой – массив акторов, которым мы не хотим наносить ущерб данным уроном.

Седьмой – актор, который наносит ущерб.

Восьмой – контроллер (передаем ничего, потому что наш актор не паун и не персонаж – он никем не контролируется).

Девятый – DoFullDamage – булевская переменная, отвечает за то, как будет наноситься урон данным актором урона, если она true – в любой точке наносится один и тот же урон, если false – урон увеличивается к центру сферы.

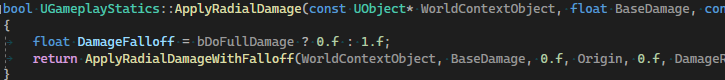
Последний, десятый аргумент, относится к коллизиям, мы оставим его по умолчанию. Он позволяет контроллировать, каким образом будет наноситься ущерб, если между нашим персонажем и эпицентром взрыва находятся другие предметы – мы можем сделать так, чтобы при данных обстоятельствах урон игнорировался.

5. Создали проперти для урона и последней переменной:

Изображение выглядит как текст

Автоматически созданное описание

6. Если изучить устройство функции ApplyRadialDamage, то мы увидим, что это обертка над функцией ApplyRadialDamageWithFalloff.



Два важных момента:

Во-первых, находятся все акторы, которые перекрываются сферой с нашими параметрами – за это отвечает функция OverlapMultiByObjectType().

Изображение выглядит как текст

Автоматически созданное описание

Во-вторых, происходит итерация по всем найденным акторам и на каждом из них вызывается функция TakeDamage(), которую мы разбирали ранее.

Изображение выглядит как текст

Автоматически созданное описание

7. Также залогируем урон, который приходит нашему персонажу в HealthComponent:



Изображение выглядит как текст

Автоматически созданное описание

В одном случае урон получаем везде одинаковый:

Изображение выглядит как текст

Автоматически созданное описание

А в другом – в зависимости от близости:

Изображение выглядит как текст

Автоматически созданное описание

Если добавить еще персонажей на сцену в сферу – они тоже будут получать урон.