11. Делегаты. Макрос UFUNCTION

1. Что рассматривается в данном уроке

2. Новое значение UPROPERTY для получения доступа на блюпринт графе

3. Записываем геттер для GeometryData (как), а также макрос для доступа в UE, что принимает

4. Как теперь в blueprint’e найти нашу функцию

5. Как получить доступ ко всем полям структуры из GetGeometryData, что отсутствует и в связи с чем у данной структуры в BP

6. Второй способ получения доступа к переменным

7. И третий способ получения доступа к данной структуре

8. Делегаты, что это, каким делегатом мы уже пользовались

9. Макрос для создания делегата + какие мы узнали типы делегатов через F12

10. Первый делегат, через какую функцию создали, когда срабатывает, что передаем параметрами + требования к названию делегатов

11. Второй делегат, через какую функцию создали, когда срабатывает, что передаем параметрами

12. Создаем делегаты уже внутри класса, до этого мы только их описали, какое проперти дописываем какому делегату и зачем

13. В какой функции, когда и как будет вызывать данные делегаты (синтаксис)

14. Как на BP-графе подписаться на делегат

15. Что автоматически создается при подписывании на делегата, когда это вызывается

16. Какие параметры есть у функции, как напечатать на экране цвет и имя актора (функция + какую уже знакомую функцию вызывает), как сконкатенировали строчку цвета и имени

17. Почистили экран от лишних акторов, удалили из массива актор и очистили проперти geometry class, чтобы не заспавнить 10 акторов

18. Подписываемся на делегаты BaseGeometry актора в классе хаба, определяем две функции, какое к ним

19. Добавили лог-категорию хабу, определили поведение наших функций в .cpp файле (какое)

20. Добавляем делегатам (динамическому и обычному) наши функции, как это сделать, синтаксис и аргументы

21. Запускаем едитор, но программа остановилась, почему, как это исправить

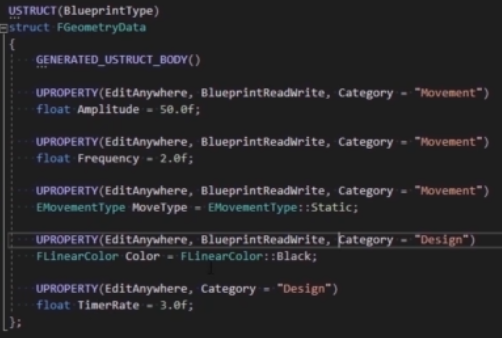
22. Что мы передаем в качестве параметра делегату, хотя у нас там другой тип, что происходит, мы можем поменять сигнатуру, а можем вызвать определенную функцию (какую) для dawn-cast’а

23. Две функции для удаления акторов со сцены (для чего каждая)

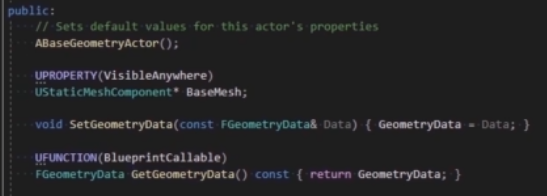
24. Функция, которая вызывается у актора перед тем, как он будет удален со сцены, как переопределить в нашем классе, что надо не забыть вызвать, что добавляем в плане функционала

1. В данной лекции мы рассмотрим с вами делегаты, что это такое, каких они бывают видов. Научим класс BaseGeometryActor отправлять ивенты, а класс хаба подписываться на данные ивенты и в зависимости от типа события выполнять какую-то специфическую логику.

2. Перед тем как начать работу с делегатами, давайте рассмотрим дополнительный спецификатор у UPROPERTY. Называется он BlueprintReadWrite. Данный спецификатор позволяет получить доступ к проперти на блюпринт графе. Добавим данный спецификатор ко всем property структуры GeometryData. К проперти TimerRate добавлять не будем, посмотрим, в чем будет разница.



3. Нам также потребуется доступ к GeometryData в классе хаба. Поэтому давайте создадим функцию getter для данного объекта. Функция будет возвращать копию объекта. Также у нее будет спецификатор const, потому что состояние класса она не меняет. Cделаем данную функцию также доступной в блюпринтах. Для этого в unreal engine существует специальный макрос, который называется UFUNCTION. В параметрах UFUNCTION необходимо указать спецификатор, который называется BlueprintCallable. Благодаря этому, мы сможем вызвать функцию GetGeometryData из блюпринт графа.

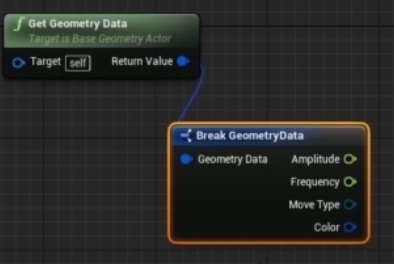


4. Компилируем проект. Давайте посмотрим, как все это выглядит в едиторе. Открываем блюпринт CubeGeometryActor. Давайте проверим, что мы имеем доступ к функции и к нашему объекту GeometryData. Для этого в любом месте графа кликаем правой кнопкой мыши и в поиске вводим GeometryData. В отфильтрованной информации видим, что нам доступна функция GeometryData, в которой мы указали спецификатор BlueprintReadWrite и также проперти GeometryData.



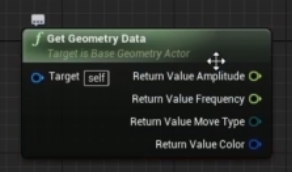
Она находится в секции protected. Данный блюпринт является наследником нашего основного класса, поэтому мы имеем к нему доступ.

5. В правом верхнем углу также можно посмотреть, от какого класса наследуется данный блюпринт. Если вытащить ниточку из пина Return Value, то можно получить доступ ко всем полям структуры. Для этого нужно вызвать ноду Break GeometryData. В данной ноде мы видим, что имеем доступ ко всем полям структуры, кроме поля TimerRate. Для данного проперти мы не указали BlueprintReadWrite, поэтому она недоступна в данной ноде.

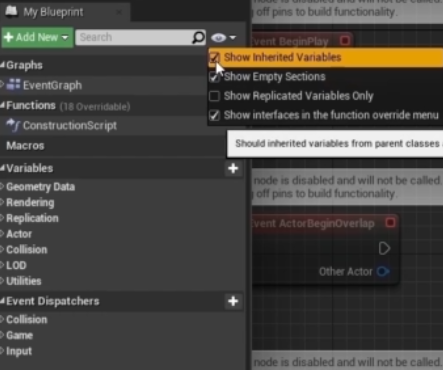


Благодаря данному спецификатору, мы можем читать данные переменные и также менять их значения. Имеется дополнительный спецификатор, который называется BlueprintReadOnly, который разрешает только читать значения переменной в блюпринтах.

6. Можно получить доступ к переменным по-другому. Нажать правой кнопкой на return value и выбрать в меню split struct pin. Так выглядит более лаконично - нам не нужна дополнительная нода break для доступа к переменам структуры.



7. Еще один способ как можно получить доступ к переменным класса это в окне My Blueprint нажать на кнопку глаза и выбрать checkbox: show inherited variables:



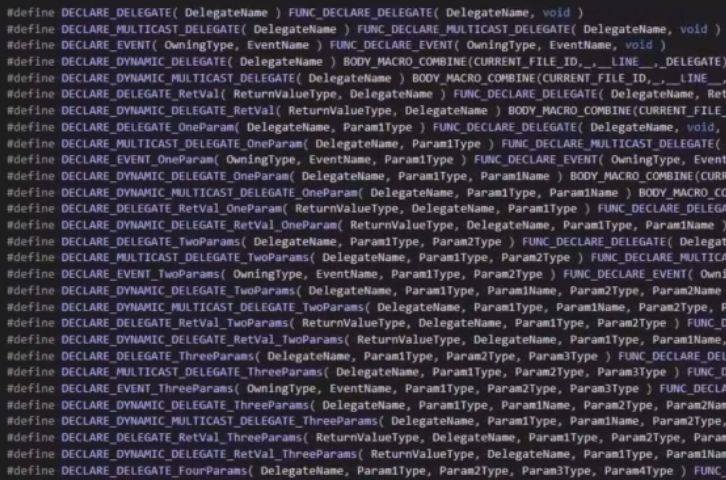
Благодаря этому в секции variables мы сможем найти нашу переменную GeometryData и с зажатой кнопкой мыши её можно напрямую перетащить на блюпринт граф. Открывается мини окошко, в котором можно выбрать либо setter для данной переменной, либо getter.

8. Переходим к делегатам. Делегаты позволяют хранить в себе ссылку на метод с определенной сигнатурой произвольного класса и вызывать этот метод, когда необходимо. Мы с вами использовали делегаты, когда создавали наш таймер. Мы передавали таймер менеджеру функцию OnTimerFired и, каждый раз когда срабатывал таймер, таймер менеджер вызвал нашу функцию.

9. Давайте создадим свой собственный делегат. Для этого идем в заголовочный файл GeometryActor и в самом верху класса объявим наш первый делегат. Делается с помощью макроса DECLARE\_DELEGATE.

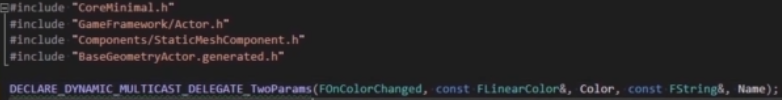


Давайте нажмем на нем F12. У нас открылся файл DeleagateCombinations, где определены все макросы, которые мы можем использовать для объявления наших делегатов.



Cамая простая версия это DECLARE\_DELEGATE - она принимает один параметр DelegateName - это произвольное имя, которое можем задать нашему делегату Данный делегат доступен только в C++, на него может подписаться один клиент, то есть этот самая простая форма делегата. Идем дальше. MULTICAST\_DELEGATE - то же самое, что и первый, только на него могут писаться много клиентов. Следующий делегат - DYNAMIC. Данный делегат отличается от первого тем, что данный тип делегата доступен в блюпринтах. multicast - соответственно в блюпринтах и может писаться несколько клиентов (в C++ он также доступен). Далее идут различные комбинации данных делегатов с параметрами. То есть мы видим \_OneParam, \_TwoParams и так далее вплоть до NineParams.

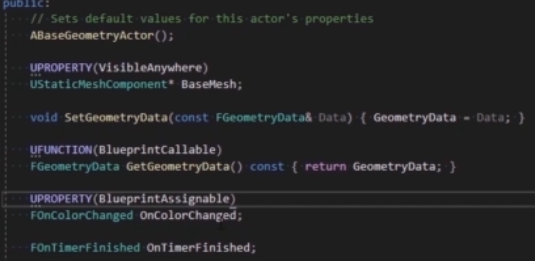
10. Давайте объявим два делегата. 1 делегат у нас будет вызываться, когда поменялся цвет материала нашего актора. У него будет два параметра - этот цвет и имя. Второй делегат будет вызываться, когда таймер смены цвета полностью завершил свою работу. У делегата будет один параметр - это указатель на текущий GeometryActor. Объявляем DYNAMIC\_MULTICAST\_DELEGATE. На него могут подписаться несколько клиентов. Он будет доступен в блюпринтах и так же у него будет два параметра. У делегата у него будет имя FOnColoredChanged Имена всех делегатов в unreal должны начинаться с заглавной буквы F - это требование unreal. Следующий аргумент макроса - это тип первого параметра делегата. Первый параметр у нас будет цвет. Используем тип FLinearColor. Следующий аргумент - это имя первого параметра. Это чисто косметический параметр. Нужен он исключительно для отображения имени параметра в блюпринтах. Следующий аргумент - это тип второго параметра. Мы будем возвращать имя актора. Поэтому воспользуемся типом FString. И последний аргумент - это имя второго параметра в блюпринтах.



11. Объявим второй делегат. Копируем полностью макрос, убираем из имени DYNAMIC. Данный делегат будет доступен только в C++. И так же у него будет один параметр. Поэтому меняем TwoParams на OneParam. Назовем наш делегат FOnTimerFinished. Тип параметра данного делегата будет указатель на актор. Мы будем возвращать указатель на текущий BaseGeometryActor. И так, как данный делегат доступен только в C++, нам не нужно указывать параметр, который отвечает за имя переменной в блюпринтах.



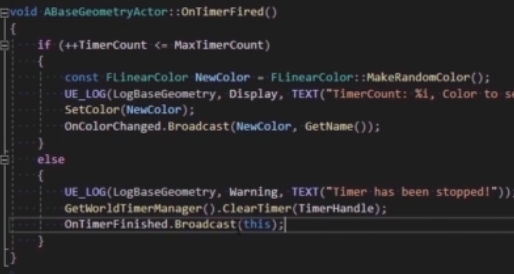
12. Мы задекларировали делегаты, определили как они у нас будут выглядеть. Теперь их надо создать в классе BaseGeometryActor. Создаем делегат OnColorChanged. И так же второй делегат OnTimerFinished. Добавляем их в секцию public в BaseGeometryActor. Чтобы на делегат OnColorChanged можно было подписаться в блюпринтах, его необходимо пометить макросом UPROPERTY. А у UPROPERTY указать спецификатор, который называется BlueprintAssignable. Совсем скоро мы увидим как все это выглядит на блюпринт графе.



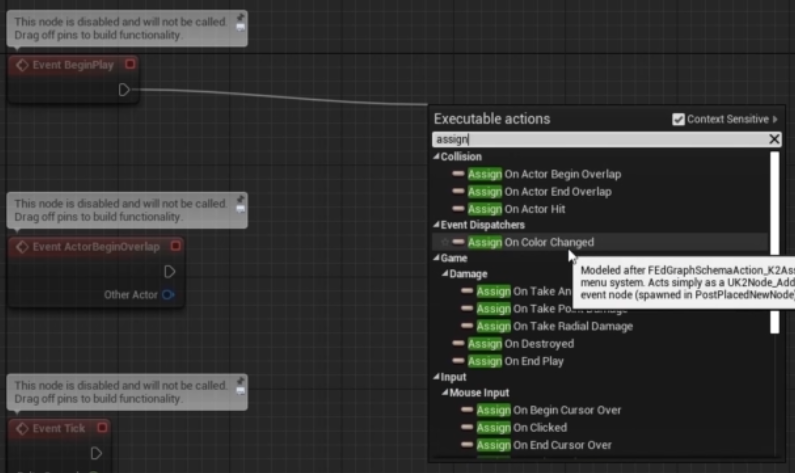
13. Нам осталось сделать последний шаг - это непосредственно вызвать данные делегаты. Делать мы это будем в функции OnTimerFired.

Первый делегат будет срабатывать, когда изменяется цвет материала. Для того, чтобы вызвать все функции, которые подписаны на данный делегат, необходимо вызвать функцию делегата, которая называется Broadcast. Мы видим, что у нее имеются параметры, которые мы определили в сигнатуре делегата. Первый параметр - этот цвет, а второй - это имя актора. В него мы можем передать значение, которое возвращает функция GetName.

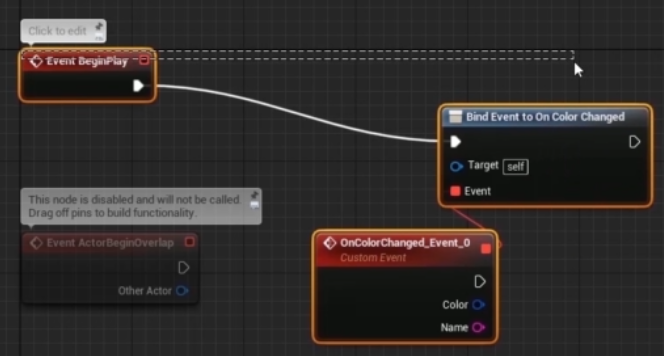
Второй делегат будет срабатывать, когда таймер закончил работу. В качестве параметра будем передавать указатель на текущий актор, для этого воспользуемся ключевым словом this.



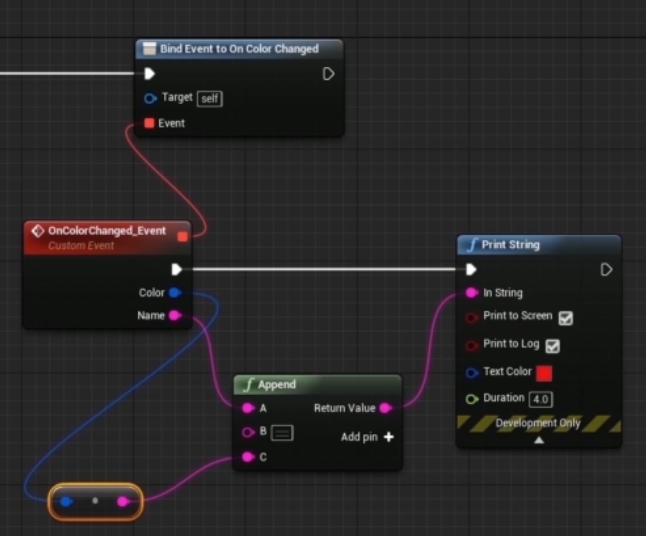
14. Cначала давайте посмотрим как можно подписаться на делегат OnColorChanged в блюпринтах, а потом подпишемся на оба делегата в классе GeometryHub. Компилируем проект. Откроем блюпринт CubeGeometryActor. Если граф не открылся автоматически, то нажимаем на кнопку Open Full Blueprint Editor. Обычно граф не открывается автоматически, если на нем нет никаких изменений. Подпишемся на делегат OnColorChanged функции BeginPlay. Вытягиваем ниточку из спина и в input поле вводим assign. Находим в списке наш делегат OnColorChanged. Выбираем его.



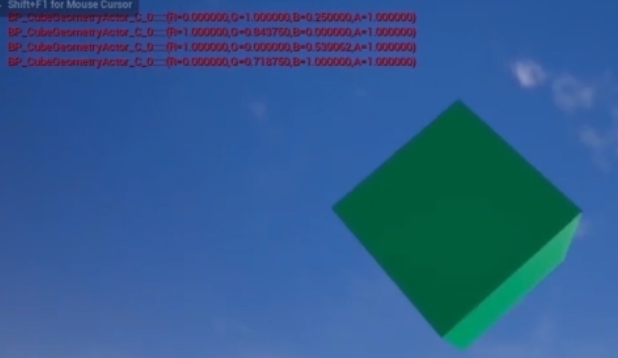
15. Автоматически создалось 2 ноды. Первая из них называется bind, которая добавляет функцию OnColoredChanged Event нашему делегату. То есть мы будем попадать в данный event каждый раз, когда вызывается функция Broadcast делегата.



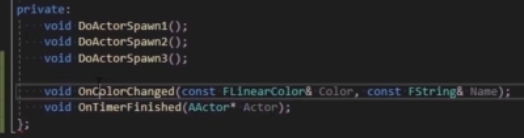
16. Видим, что у функции создались два параметра: color и name. Данные имена мы указали в сигнатуре делегата. Напечатаем на экране цвет и имя актора. Для этого воспользуемся функцией, которая называется PrintString. Данная нода вызывает функцию AddOnScreenDebugMessage, которую мы уже с вами рассматривали. Нам нужно сконкатенировать строчку цвета и имени. Для этого воспользуемся специальной нодой, которая называется Append. В первый пин добавляем имя. Так же можем добавить дополнительный пин для еще одной строчки. И например добавим какой-нибудь разделитель. В третий пин выводим цвет. Автоматически создалась нода преобразования цвета в string. И подключаем финальную строку в PrintString.



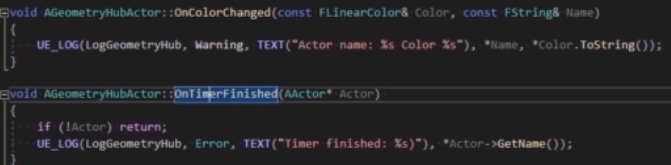
17. Давайте удалим со сцены BaseGeometryActor, который у нас имеется и немного отредактируем hub. Очищаем property GeometryClass. В результате этого не будут заспаунены 10 акторов. И также в GeometryPayloads оставим только один элемент массива. Мы с вами все это делаем, чтобы не опечалилась куча сообщений от разных акторов на экране. У элемента нажимаем на треугольник и выбираем опцию delete. То есть фактически на сцене у нас заспаунится только один geometry actor. Также давайте поменяем TimerRate, чтобы цвет менялся, например, каждую секунду. Cохраняемся и запускаем игру. Наш код работает. На экране печатается имя актора и измененный цвет материала.



18. Теперь давайте подпишемся на делегаты BaseGeometry актора в классе хаба. Для начала в заголовочном файле хаба объявим две функции, которые будут вызываться при срабатывании соответствующих делегатов. Первую функцию назовем OnColoredChanged. Сигнатура функций должна совпадать с сигнатурой делегатов. И вторая функция OnTimerFinished.



19. Создаем тело для каждой функции в .cpp файле. Давайте добавим собственную лог категорию GeometryHub актору. В функции OnColorChanged давайте выведем в консоль цвет и имя актора. И также будем выводить сообщение, когда таймер закончил работу. Копируем log. Изменим Verbosity на Error, чтобы текст в консоли подсвечивался красным цветом. Напечатаем Timer finished. Дополнительно выведем имя актора. Для этого воспользуемся функцией GetName. Также перед выводом проверим, что указатель на actor не является нулевым. Если он нулевой, то мы сразу выходим из функции.

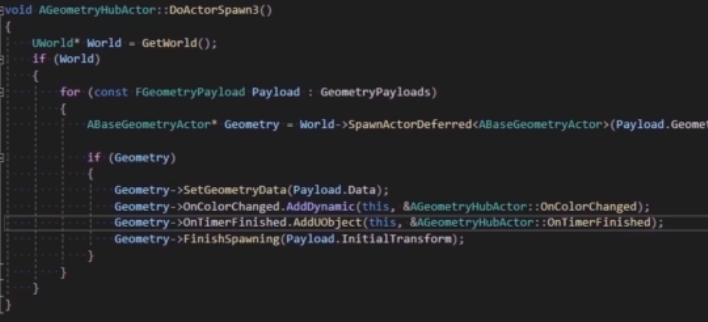


20. И последнее, что нам нужно сделать это забиндить данные функции на соответствующие делегаты. На BeginPlay можно закомментировать первый и второй спаун - они нам не понадобятся.

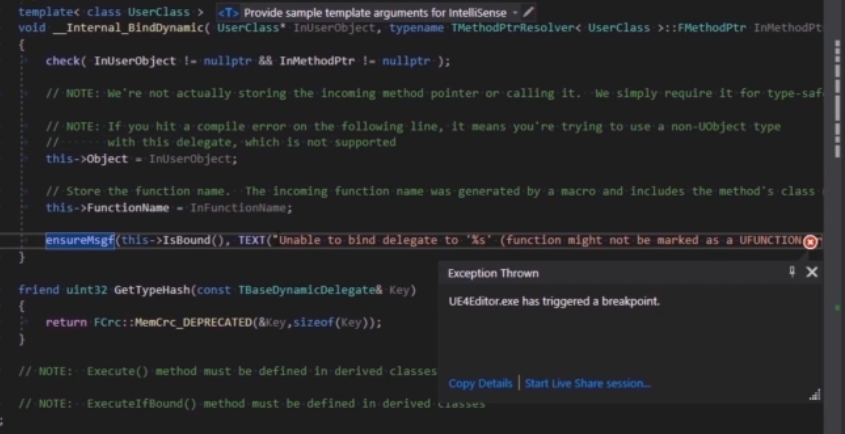
Итак, переходим функцию третьего спауна. Мы создаем actor, изменяем его GeometryData и завершаем spawn. Перед тем как мы завершаем spawn мы можем сделать бинд наших функций.

Первый делегата OnColorChanged является динамическим. Для того чтобы добавить функцию динамическому делегату, необходимо воспользоваться специальным макросом, который называется AddDynamic. Макрос принимает 2 параметра. Первый - это указатель на объект, а второй - это ссылка на функцию данного объекта, которую мы будем вызывать при срабатывании делегата. Первым параметром идёт this - указатель на текущий объект класса GeometryHubActor. А вторым параметром - ссылка на функцию OnColorChanged.

И также нам нужно добавить вторую функцию. Делегат OnTimerFinished не является динамическим. Для добавления функций к такому виду делегата используется функция AddUObject. Параметры у нее такие же: указатель на объект и ссылка на функцию.

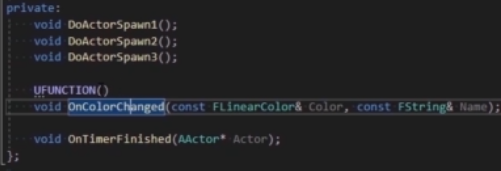


21. Итак, мы с вами сделали бинд для двух функций. Запускаем editor. Посмотрим как все это выглядит. Нажимаем кнопочку play и программа остановилась. Давайте посмотрим, что случилось. Код остановился из-за assertion. Давайте посмотрим, что написано в тексте сообщения.



В сообщении говорится, что не получается сделать бинд для делегата, поскольку функция callback не помечена макросом UFUNCTION. Данное сообщение относится к динамическому делегату. Давайте исправим ошибку. Переходим заголовочный файл GeometryHubActor. И над функции OnColorChanged добавляем UFUNCTION.

Зачем это нужно поговорим чуть попозже в конце данного раздела, а сейчас будем иметь в виду, что функция, которую мы добавляем динамическим делегатам необходимо пометить макросами UFUNCTION. В данном случае макросы UFUNCTION не имеет отношения к тому, хотим ли мы вызвать данную функцию в блюпринтах или нет. Это имеет отношение к модели памяти движка.



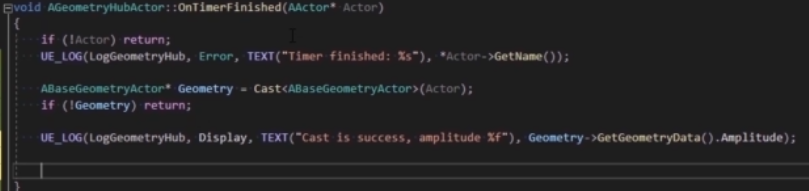
Останавливаем игру и перекомпилируем проект в едиторе. Откроем лог. Очистим его. И запустим заново наш проект. Функции GeometryHub вызываются. Печатается log с именем актора и новым цветом материала. И в самом конце лог GeometryHub сообщает, что таймер закончил работу.

22. Давайте сейчас дополнительно рассмотрим еще пару вещей. Когда мы вызываем функцию Broadcast нашего делегата, то в качестве параметра передаем указатель на текущий actor, то есть фактически указатель на BaseGeometryActor, но в сигнатуре делегата мы указали параметр, как указатель на базовый класс актора - AActor.

Фактически, при вызове функции Broadcast происходит автоматический up-cast - преобразование указателя от ABaseGeometryActor к указателю на просто AActor. Мы с вами, конечно же, можем поменять сигнатуру делегата и указать явно сигнатуре ABaseGeometryActor.

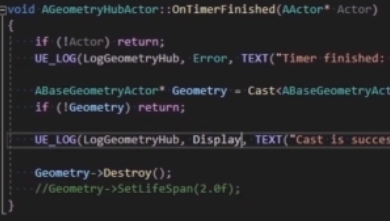
Мы сделаем по-другому. И сами преобразуем указатели на AActor к указателю на ABaseGeometryActor. Для этого в Unreal существует специальная функция, которая называется Cast. По сути она делает dawn-cast и преобразуют указатель на базовый класс к указателю на класс наследника. Функция шаблонная. В качестве параметра шаблона она принимает тип, к которому нужно преобразовать указатель.

Проверим преобразованный указатель на nullptr, если получился нулевой указатель, то сразу выходим из функции, а если нет, то давайте напечатаем какое-нибудь сообщение. И выведем, например, значение амплитуды из объекта GeometryData. Для этого мы можем воспользоваться функцией, которую мы создали в начале данного урока - GetGeometryData. И обращаемся к полю Amplitude.



23. Давайте посмотрим как мы мы можем удалить актор со сцены. Для этого у акторов существу специальная функция, которая называется Destroy.

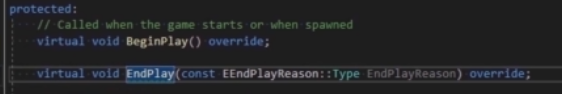
Либо можно сделать это с помощью функции, которая называется SetLifeSpan. Данная функция принимает один параметр - это количество секунд через сколько актор будет удален. То есть фактически данная функция запускает таймер, и при срабатывании данного таймера вызывается функция Destroy. Второй вызов давайте закомментируем.



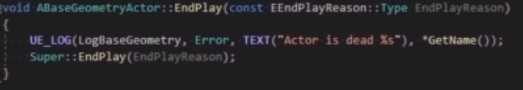
24. Рассмотрим еще одну функцию акторов, которая называются EndPlay. Давайте нажмем F12 на классе актора. В поиске вводим EndPlay. Данная функция вызывается у актора перед тем как он будет удален со сцены.



Давайте переопределим её в классе BaseGeometryActor. Объявляем её в секции protected. И добавим ключевое слово override.



Создаем тело функции в .cpp файле. Также нам необходимо вызвать функцию базового класса через ключевое слово Super, чтобы не потерять никакую функциональность. И давайте в данной функции просто напечатаем сообщение, что актор уничтожен. Установим уровень логирования Error.



Компилируем проект, запускаем editor. Открываем лог, очищаем и запускаем игру. Ждем некоторое время. Итак, смотрим последние логи. Вывелось сообщение, что таймер остановлен, далее - таймер завершён. После этого преобразование указателя произошло успешно. Далее мы вызываем функцию Destroy актора. И напечаталось сообщение функции EndPlay. Актор со сцены удалился. Давайте посмотрим ещё раз. Наводим камеру на куб. И всё - куб удалился.