10. Спаун объекта класса AActor

1. Создали новый класс-спаунер для акторов BaseGeometry

2. Что нужно для начала получить, в чем содержится функция спауна, какую библиотеку надо подключить, как называется функция

3. Что принимает данная функция, как будут установлены остальные параметры по умолчанию, если передать только класс

4. Создали проперти для нашего класса через специальную функцию, синтаксис

5. Какие еще проперти создали в учебных целях

6. Какой надо подключит ЗФ

7-11. Чем отличаются наши проперти и что дает функция из пункта 4

12. Доработали наш класс BP\_BaseGeometryActor зачем-то

13. Что возвращает функция для спауна и как можно ее шаблонизировать

14. Создали трансформацию для нашего спаунера и передали туда

15. Создаем акторов в цикле и меняем им координату от счетчика

16. Имеем указатель на созданный нами объект – значит может что-то менять – проперти и функция для этого и у какого из классов

17. Как теперь в цикле меняем параметры у генерируемых акторов

18. Что получили в едиторе

19. Забыли одну важную проверку

20. Как работает функцию SpawnActor?

21. Другой способ спауна актора с отложенным вызовом функции BeginPlay - функция

22. Чем отличается от SpawnActor? Что нам позволяет сделать? Как создать объект после всего (функция)?

23. Что получили в едиторе

24. Как в классе хаба будем настраивать GeometryData акторов?

25. Создали массив (специальный тип UE) наших новых структур

26. Как спауним акторов в зависимости от информации из массива

27. Переименовали два блюпринта

28. Кастомизация нашего массива в UE + вынесли в функции всех спаунеров акторов

29. И поправили код – удалили учебные проперти и добавили везде проверки на кое-что

1. В данном уроке мы рассмотрим с вами динамическое создание акторов. Так называемый spawn. Для этого нам потребуется дополнительный actor. Назовем его hub. В нём будет происходить логика создания наших BaseGeometry акторов. Давайте с него и начнем. Добавляем новый C++ класс. Наследуемся от актора. Называем наш новый класс GeometryHubActor. И нажимаем на кнопку create class. Итак, наш класс создан, класс, который будет отвечать за создание BaseGeometry акторов, наш hub.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

2. Давайте приступим к написанию кода. Для начала получим указатель на мир игры - world. В данном объекте находится функция спауна. Мы помним, что каждый автор имеет функцию GetWorld, которая возвращает указатель на глобальный объект мира игры. Проверяем что наш указатель не нулевой. Функция спауна актора называется SpawnActor. С большой буквы:

Изображение выглядит как текст

Автоматически созданное описание

World подчеркивается в Visual Studio красной волнистой линией. Visual Studio таким образом сообщает нам, что необходимо подключить заголовочный файл, где объявлен тип UWorld. Добавляем сверху .cpp файла #include "Engine/World.h" и посмотрим на функцию спауна актора:

Изображение выглядит как текст

Автоматически созданное описание

3. Данная функция перегружена. Имеется несколько вариантов, как мы можем заспаунить актор. Некоторые из данных функций шаблонные, некоторые нет. Но смысл более-менее у всех у них одинаковый. Вначале мы должны указать класс актора, который мы хотим заспаунить. Далее идут параметры трансформации: положение в мире, вращение и, в самом конце, идет специальная структура, которая называется FActorSpawnParameters. В ней указываются дополнительные настройки спауна. Сейчас мы её рассматривать не будем. Оставим по умолчанию. Вернемся к ней в дальнейших разделах. Мы можем вызвать данную функцию всего лишь с одним параметром - классом актора, который нам необходимо заспаунить:

Изображение выглядит как текст

Автоматически созданное описание

Матрица трансформации при этом будут единичной. Актор заспаунится в начале координат с нулевыми вращениями и с единичным скейлом.

4. Создадим property GeometryClass в заголовочном файле, в секции protected. В макросе проперти укажем спецификатор EditAnywhere. Для объявления указателя на класс в unreal существует специальный шаблон, который называется TSubclassOf. Параметром данного шаблона необходимо указать непосредственно класс. Мы будем спаунить BaseGeometryActor, поэтому передаем шаблону данный тип и называем GeometryClass:

Изображение выглядит как текст

Автоматически созданное описание

Мы совсем скоро увидим как-то выглядит в едиторе, в панели details. По сути TSubclassOf отфильтрует все доступные классы движка и в property можно будет установить только непосредственно сам BaseGeometryActor класс или класс, который от него наследуется.

5. В учебных целях создадим дополнительное property. С помощью первого мы посмотрим, как установить любой доступный класс Базовый тип класса в unreal называется UClass. Создаем указатель на него и называем property просто Class. А с помощью второго property посмотрим, как передать указатель на существующий объект на сцене. Назовем проперти GeometryObject и сейчас в едиторе увидим, чем они все отличаются:

Изображение выглядит как текст

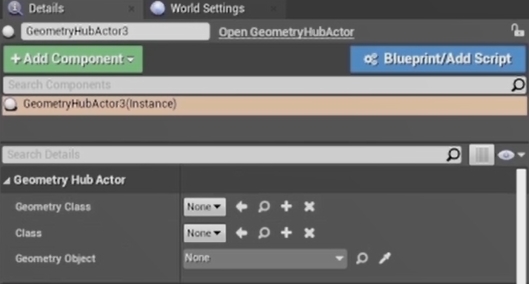
Автоматически созданное описание

6. Также нам необходимо указать заголовочный файл BaseGeometryActor. Давайте это сделаем:

Изображение выглядит как текст

Автоматически созданное описание

7. Компилируем проект. Запустился едитор. Давайте найдем наш класс хаба в директории с C++ классами. Выкидываем его на сцену и переходим в панель details. В панели details видим, что отображаются наши три property: GeometryClass, Class и GeometryObject:



8. Давайте сначала нажмем на проперти класс, на combobox. Открывается список со всеми доступными классами движка. С помощью данного property мы сможем создать объект абсолютно любого класса.

Изображение выглядит как текст

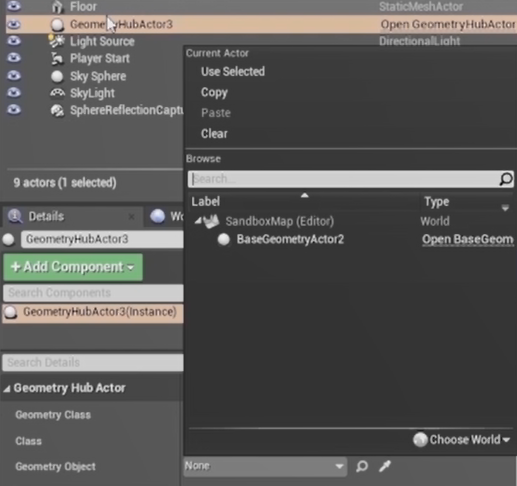
Автоматически созданное описание

9. Далее если мы нажмём на combobox GeometryClass. Мы указали данное property, как TSubclassOf<ABaseGeometryActor> и видим, что все классы отфильтрованы и мы можем выбрать только из основного класса и его блюпринтового наследника.

Изображение выглядит как текст, снимок экрана, электроника

Автоматически созданное описание

10. А третье property позволяет установить указатель на объект класса BaseGeometryActor, который находится на сцене. Видим, что на сцене существует объект BaseGeometryActor2. Мы его можем указать через данное property и в дальнейшем в коде как-то использовать данный объект.



11. Итак, мы разобрались чем отличаются три наших проперти. Основное преимущество шаблона TSubclassOf в том, что он фильтрует все доступные нам классы и мы можем не волноваться, что кто-то установит неправильный класс в данное property.

In addition to this UPROPERTY safety, you get type safety at the C++ level too. If you try to assign incompatible TSubclassOf types to each other you'll get a compilation error. In the case you are trying to assign a generic UClass, it will perform a runtime check to verify that it can do the assignment. If the runtime check fails, the resulting value is nullptr.

UClass\* ClassA = UDamageType::StaticClass();

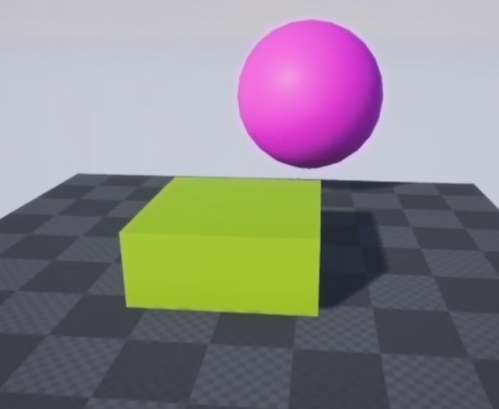
TSubclassOf<UDamageType> ClassB;

ClassB = ClassA; // Performs a runtime check

TSubclassOf<UDamageType\_Lava> ClassC;

ClassB = ClassC; // Performs a compile time check

12. Давайте доработаем немного наш класс BP\_BaseGeometryActor, который мы сейчас заспауним. Если нажать на кнопку лупа у property, то контент браузер перейдет в директорию, где находится данный класс и подсветит его. Давайте настроим наш блюпринт. В предыдущих лекциях мы перетаскивали actor на сцену и непосредственно у объекта настраивали его property: меш, материал. Мы это можем также сделать непосредственно в блюпринт классе, то есть настроить внешний вид по умолчанию. Для этого давайте установим какой-нибудь статик меш. Если не виден контен, нажимаем view options -> show engine content и установим, например, куб. Также давайте присвоим какой-нибудь материал, установим BaseColor. Запускаем игру. Заспаунился куб. Его трансформация единичная. Находится он в точке (0, 0, 0) нашей сцены:



13. Давайте возвращаться в visual studio. Усовершенствуем наш код. Функция SpawnActor помимо того, что она принимает некоторые параметры, она также возвращает указатель на актор. Мы можем воспользоваться шаблонной версией данной функции. В качестве параметра шаблона передать ABaseGeometryActor. В результате этого создастся функция, которая возвращает указатель на тип ABaseGeometryActor:

Изображение выглядит как текст

Автоматически созданное описание

Благодаря этому, нам не надо будет делать преобразование типов, то есть кастить объект класса AActor к объекту класса ABaseGeometryActor. Мы сразу имеем указатель на объект нашего типа.

14. Давайте создадим объект трансформации и установим данную трансформацию актору при спауне. Класс FTransform имеет много различных конструкторов. Мы воспользуемся конструктором, который в качестве первого параметра принимает объект FRotator, который отвечает за углы вращения. В конструктор FRotator можно передать три 0, а можно воспользоваться константой ZeroRotator, в которой по сути это уже сделано за нас. Во втором параметре конструктора FTransform передаем желаемое положение в мире для актора. Для этого используем объект типа FVector и в конструкторе указываем какую-нибудь координату. Пусть будет (0, 300, 300). Итак, объект трансформации готов и, чтобы актор имел данную трансформацию при спауне, нам необходимо передать данный объект вторым аргументом в функцию SpawnActor:

Изображение выглядит как текст

Автоматически созданное описание

15. Давайте, чтобы наша сцена была повеселее, создадим не один актор, a 10. Для этого воспользуемся циклом for Перемещаем код создания актора в тело цикла и, чтобы все акторы не заспунились в одной точке, будем менять координату по Y в зависимости от индекса цикла:

Изображение выглядит как текст

Автоматически созданное описание

16. Более того, у нас теперь имеется указатель на созданный нами объект и мы можем что-нибудь в данном объекте поменять. Откроем заголовочный файл BaseGeometryActor. Давайте будем менять настройки, которые находятся в объекте GeometryData. Данный член класса находится в секции protected, поэтому нам необходимо сделать публичную функцию setter для данной структуры, чтобы изменять ее значение. Создаём функцию SetGeometryData. Функция будет принимать объект типа FGeometryData. Присваиваем члену класса GeometryData значение параметра Data. Тело функции я оставлю в заголовочном файле, не будем создавать его в .cpp. В целом я допускаю, что небольшие функции, в основном это setter/getter, находятся целиком в .h файле:

Изображение выглядит как текст

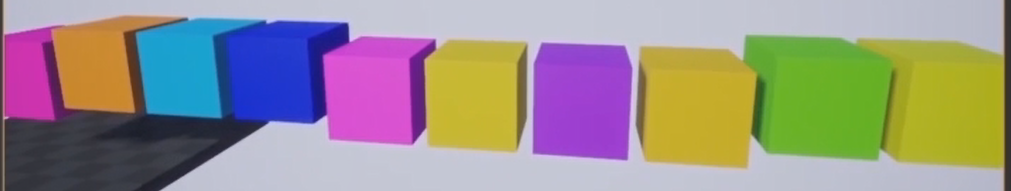
Автоматически созданное описание

17. Теперь мы можем изменять объект GeometryData. Давайте вызовем созданную нами функцию SetGeometryData. Передадим в данную функцию объект Data и давайте его создадим. Давайте, например, поменяем поле MoveType. Тип движения каждый раз будем задавать случайным образом. Для этого можно воспользоваться специальной функцией, которая находится FMath. Она называется RandBool. Данная функция случайным образом генерируют true, либо false. И, например, если сгенерировалось true, давайте установим MovementType::Static Для удобства мы воспользуемся тернарный оператором ? А если функция выдала false, то устанавливаем MovementType::Sin:

Изображение выглядит как текст

Автоматически созданное описание

18. Компилируем проект. Запускаем игру. В соответствии с нашим кодом сгенерировалось 10 BaseGeometry акторов. Некоторые из них двигаются, некоторые нет. Это зависит от функции RandBool:



19. Давайте подправим одну вещь. Мы забыли проверить указатель Geometry на nullptr. Объект может и не создаться, например, если у вас закончилась память:

Изображение выглядит как текст

Автоматически созданное описание

20. Давайте теперь посмотрим, как работает функция SpawnActor. Откроем .cpp файл класса BaseGeometryActor. При создании объекта вызывается конструктор класса BaseGeometryActor и после этого сразу BeginPlay. Как мы помним, в нашем BeginPlay мы инициализируем переменную InitialLocation. При создании актора устанавливается трансформация, которую мы передаем в функцию SpawnActor. И при вызове функции GetActorLocation на BeginPlay, мы получим координату, которую мы передали с объектом FTransform. Цвет по умолчанию мы поменять не можем, поскольку SetGeometryData мы вызываем уже на созданном объекте после вызова функции BeginPlay. То есть по умолчанию у нас цвет будет черный для всех акторов.

21. Имеется другой способ спауна актора с отложенным вызовом функции BeginPlay. Давайте его рассмотрим. Для начала скопируем полностью цикл. И заспауним еще 10 акторов. Единственное, давайте поменяем им координату по Z. Установим 700. Функция, которая позволяет делать отложенный вызов BeginPlay, называется SpawnActorDeffered. Давайте её вызовем:

Изображение выглядит как текст

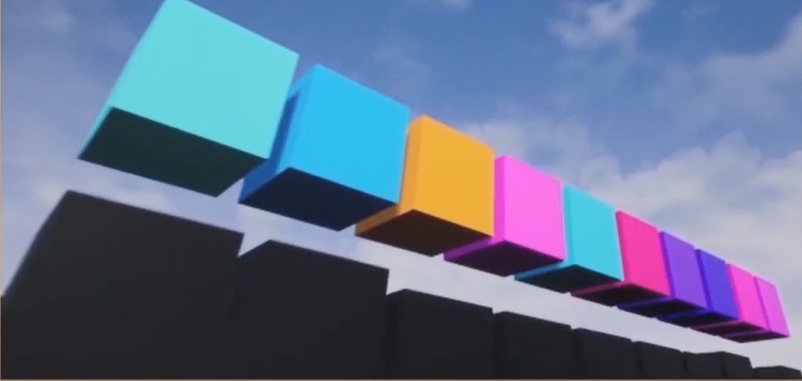
Автоматически созданное описание

22. Данная функция отличается от обычной функции SpawnActor тем, что при вызове создается объект и вызывается только конструктор класса, а функция BeginPlay вызвана не будет. Благодаря этому, у нас появляется возможность настроить некоторые параметры актора и вызвать функцию BeginPlay вручную. Передадим в объекте Data также цвет, который мы хотим, чтобы актор имел по умолчанию. Генерируем случайный цвет, вызывая функцию MakeRandomColor. Итак, мы создали объект, вызвался конструктор класса. Далее мы установили GeometryData нашего класса, и теперь нам необходимо вручную вызвать функцию BeginPlay. Делается это с помощью функции актора, которая называется FinishSpawning. Данной функции также необходимо указать трансформацию. Она у нас меняться не будет, поэтому передаем объект GeometryTransform:

Изображение выглядит как текст

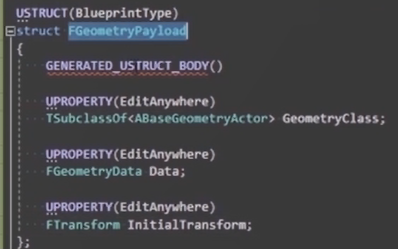
Автоматически созданное описание

23. Компилируем проект. Нажимаем play и видим, что верхний ряд акторов сразу заспаунился с цветами:



Таким образом, благодаря функции SpawnActorDeffered, мы можем отложить вызов функции BeginPlay и изменить все параметры, которые нужно, до вызова функции BeginPlay.

24. Далее сделаем следующее. В классе хаба будем полностью настраивать GeometryData для акторов, которые мы хотим заспаунить. Для этого нам понадобится дополнительная структура в классе GeometryHubActor. Копируем код структуры, переносим заголовочный файл GeometryHubActor. Назовем структуру GeometryPayload. В структуре будет три поля. Первое - это класс, который мы хотим заспаунить. Второе будет отвечать за GeometryData. Третье будет отвечать за начальную трансформацию. Назовем проперти InitialTransform:



25. Cоздадим проперти типа FGeometryPayload в классе GeometryHubActor. Называем проперти GeometryPayloads. Давайте сделаем проперти массивом объектов. Д ля этого воспользуемся типом unreal engine, который называется TArray. Это класс-контейнер. В качестве параметра шаблона он принимает тип хранимых данных. Мы передаем тип FGeometryPayload:

Изображение выглядит как текст

Автоматически созданное описание

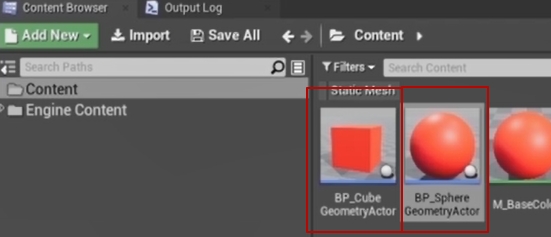
26. Давайте заспауним акторов, в зависимости от информации, которая находится в массиве GeometryPayloads. Будем использовать range версию цикла for. Пройдемся по всем элементам массива и заспауним акторы. Будем спаунить акторы с помощью вызова функции SpawnActorDeffered. Можем скопировать строчку из предыдущего цикла. Параметры функции поменяем на переменные нашего объекта Payload. Второй параметр Payload.InitialTransform Проверяем объект Geometry на nullptr Если объект не нулевой, то вызываем функцию SetGeometryData. В качестве параметра передаем переменную Data объекта Payload. Завершаем создание актора, вызывая функцию FinishSpawning. Передаем в нее InitialTransform:

Изображение выглядит как текст

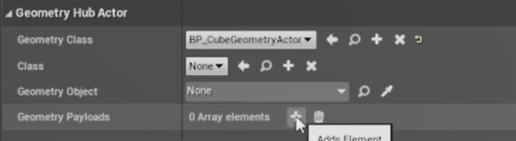
Автоматически созданное описание

Итак, наш код готов. Давайте запускать editor.

27. Первое, что мы с вами сделаем - переименуем блюпринт BaseGeometryActor в CubeGeometryActor, потому что мы установили меш по умолчанию для данного блюпринта - куб. Нажимаем Ctrl+W, копируем блюпринт. Назовем его SphereGeometryActor. В качестве меша в данном классе установим сферу. Компилируем, сохраняем, закрываем:



28. Переходим в GeometryHubActor. Видим, что создалось проперти GeometryPayloads, которое является массивом. У проперти массива в едиторе имеется кнопка плюсик, которая позволяет добавлять элементы данному массиву:



Элементом нашего массива является GeometryPayload. Мы видим все отображаемые проперти данной структуры. В качестве класса установим BP\_Cube. Также давайте поменяем GeometryData. Установим какой-нибудь зеленый цвет. Таймеру, который меняет цвет, установим какое-нибудь большое значение. Пусть будет 20. И также давайте зададим локацию в мире, где мы хотим, чтобы наш актор заспаунился. Также можно поменять вращение нашего актора. Пусть будет 45 градусов по всем осям:

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Нажимаем play. Наш новый актор заспаунился в мире, все его параметры соответствуют тому, что мы указали. Давайте добавим еще один элемент в наш массив. Нажимаем кнопку плюс. Установи м в GeometryClass сферу. Пусть она по умолчанию движется по синусу. Также поменяем цвет. Пусть будет белый. TimerRate сделаем большим - 50 и поменяем начальное положение. Запускаем игру. Заспаунилась наша сфера и она двигается. Для каждого из спаунов, которые мы делали в данном уроке, я создал дополнительную функцию DoActorSpawn 1, 2 и 3:

Изображение выглядит как текст

Автоматически созданное описание

29. Давайте также в GeometryHubActor удалим два проперти: Class и GeometryObject. Они нам не потребуются. Мы создавали их чисто в учебных целях, а в классе BaseGeometryActor давайте подправим все места, где мы не проверяем указатели на nullptr. Первое – это функция HandleMovement. Проверим существует ли объект мира игры или нет. В целом если указатель на глобальный объект игры нулевой, то что-то явно идет не так в нашей игре. И в целом она продолжаться уже не может, но лучше соблюдать единый стиль кодирования. То же самое делаем с объектом GEngine. И последнее место. Мы с вами не проверяли указатель на BaseMesh. Если указатель на него нулевой, то сразу же выходим из функции.