1. Компоненты UE представляют собой объекты, с определенной функциональностью, которую мы можем добавить к актору для расширения его возможностей. С точки зрения объектно-ориентированного программирования данный прием называется композицией.

2. Объявим указатель на наш первый компонент. Указатель будет иметь тип UStaticMeshComponent. Назовем его BaseMesh. данный компонент содержит в себе static mesh, то есть некую геометрию, которую мы с вами совсем скоро установим в редакторе. Добавляем в property VisibleAnywhere, то есть говорим unreal, что мы хотим видеть данный компонент доступным в едиторе. Также нам нужно подключить заголовочный файл нашего компонента. Находится он в Components/StaticMeshComponent:

Изображение выглядит как текст

Автоматически созданное описание

3. Компоненты в unreal бывают нескольких типов. Некоторые из них содержат только логику. У них не имеется визуального представления, не имеется трансформации в сцене. Данный тип компонентов подходит, например, для управления инвентарем персонажа, либо для компонента здоровья. Другой тип компонентов имеет визуальное представление, имеет трансформацию. К ним относится например StaticMesh компонент, компонент по управлению системой частиц и многие другие.

4. Переходим в .cpp файл нашего актора. В конструкторе создадим BaseMesh компонент. Для создания нашего компонента воспользуемся функцией, которая называется CreateDefaultSubobject. Данная функция шаблонная. В качестве параметра шаблона она принимает тип создаваемого подобъекта. В нашем случае это UStaticMeshComponent. И возвращает указатель на созданный подобъект. Также функция принимает два аргумента. Первый аргумент - это имя компонента. Назовем его BaseMesh. Мы видим, что тип первого параметра FName - это другой вид строки в unreal engine. Это более легковесная строка. Она имеет меньший функционал и для строковых манипуляции она не подходит. Данный вид строки используют в основном для задания имен компонентов, имен игрокам, параметров материалов, частей тел в скелете и так далее. Второй аргумент рассматривать не будем, для нас он сейчас не важен и просто оставим его по умолчанию false:

Изображение выглядит как текст

Автоматически созданное описание

5. Последнее, что мы с вами сделаем перед тем как запустить editor – это укажем корневой компонент нашему актору. Для этого воспользуемся функцией SetRootComponent и передадим в неё указатель на наш static mesh компонент BaseMesh:

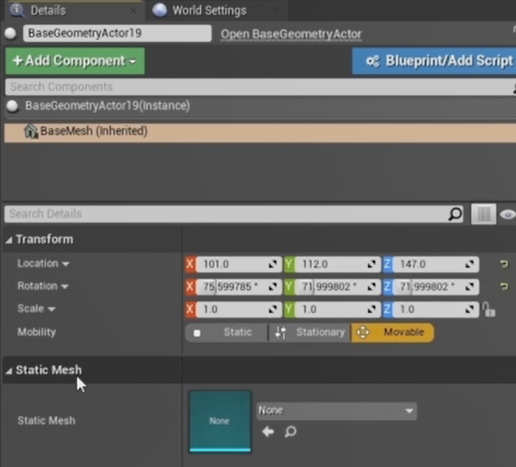
Изображение выглядит как текст

Автоматически созданное описание

Указывая корневой компонент, мы декларируем, что трансформация нашего актора будет совпадать с трансформацией данного компонента.

6. Запустив редактор и добавив актора на сцену мы видим, что у нас появились Gizmo у объекта.

7. При этом в панели Details мы видим, что появился BaseMesh, который мы задали в C++, поля transform и StaticMesh:



8. При нажатии на combobox открывается селектор static mesh. В нашем случае он пустой, потому что при создании проекта мы не добавляли никакой визуальный контент. Это можно исправить в контент браузере. Слева есть большая зелёная кнопка Add New. Выбираем в меню add feature or content pack. В открывшемся окне на вкладке content pack существует кнопка starter content. Здесь находится базовая геометрия, материалы.

Можно добавить пак и выбрать меши и оттуда, но мы воспользуемся другим способом. Кликаем на combobox статик меша. В правом нижнем углу нажимаем на view options. Cтавим галочку show engine content и в списке показываются static mesh, которые поставляются вместе с едитором.

Найдем в списке более адекватную геометрию, например, введём cube. Наш актор превратился в куб. Его трансформация совпадает с трансформацией корневого компонента нашего статик меша.

Мы можем вытащить еще один актор и в статик меш компонент давайте поставим сферу. Таким образом с использованием лишь одного базового актора и статик меш компонента мы создали два совершенно разных инстанса, с разной геометрией и трансформацией.

9. Давайте получим доступ к трансформации нашего актора из кода. Воспользуемся функцией актора, которая называется GetActorTransform. Данная функция возвращает объект типа FTransform. Создаем переменную Transform и присваиваем ей возвращаемое значение данной функции:

Изображение выглядит как текст

Автоматически созданное описание

Наводим мышкой на FTransform, нажимаем F12 и видим, что в данной структуре содержатся три переменные: rotation, translation и scale:

Изображение выглядит как текст

Автоматически созданное описание

Давайте их оттуда достанем. Для этого воспользуемся getter методами данной структуры. Первый из них GetLocation. Данная функция возвращает объект типа FVector - позиция нашего актора в 3d пространстве:

Изображение выглядит как текст

Автоматически созданное описание

Нажмем F12 на FVector и видим, что данная структура содержит три компонента: x, y и z:

Изображение выглядит как текст

Автоматически созданное описание

Далее создаем переменную типа FRotator. Данный тип отвечает за углы поворота в 3d пространстве. Функция трансформа, которая возвращает данный объект, называется Rotator. И создаем третью переменную Scale. Она имеет также тип FVector. Функция, которая возвращает скейл актора, называется GetScale3D:

Изображение выглядит как текст

Автоматически созданное описание

10. Давайте все эти переменные выведем в output лог. Начнём мы с трансформа. У каждого из данных типов существует вспомогательная функция, которая позволяет перевести внутренние данные в строчку. Функция называется ToString:

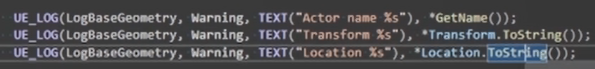


Если мы нажмем F12 на ToString, то увидим, что внутри функции происходит print всех элементов трансформации:

Изображение выглядит как текст

Автоматически созданное описание

Далее дублируем строчку, пишем Location. Location.ToString



Нажимаем F12. Видим, что внутри данной функции собираются в строчку все компоненты вектора, с некоторой точностью.

Изображение выглядит как текст

Автоматически созданное описание

То же самое делаем с Rotation. Rotation.ToString Таким же образом выводим и последнюю переменную Scale:

Изображение выглядит как текст

Автоматически созданное описание

Последнее, что мы с вами сделаем. Давайте выведем Transform немного в другом виде. У трансформа есть дополнительная функция, которая называется ToHumanReadableString. Она более информативная. Напишем Human Transform и поменяем уровень логирования на Error, чтобы было заметнее при выводе в консоль:



Нажимаем F12, заходим во внутренности ToHumanReadableString. И видим, что преобразование в строчку в данной функции более подробное:

Изображение выглядит как текст

Автоматически созданное описание

В UE в логе видим, что на сцене находятся два актора. Для каждого из которых вывелся свой набор параметров: трансформация, локация, rotation, scale и красным напечаталась строка в более подробном виде:

Изображение выглядит как текст

Автоматически созданное описание

Перенес полученную логику в функцию printTransform():

Изображение выглядит как текст

Автоматически созданное описание

11. Далее давайте добавим немного интерактива и научим наш фактор двигаться по закону синуса. Для этого каждый фрейм будем изменять z координату актора по формуле:

Изображение выглядит как текст

Автоматически созданное описание

Нам понадобится два параметра. Для этого идем в .h файл. Создаем там property. Указываем спецификатор EditAnywhere. Добавим данное property в категорию Movement. Первый параметр будет амплитуда, то есть насколько actor будет отклоняться от первоначальной точки. Зададим 50. Нам сейчас не так важно начальное значение, потому что в едиторе мы всегда сможем его поменять. Копируем данный параметр и меняем амплитуду на frequency. Frequency будет отвечать за частоту колебания относительно начального положения актора. Поставим его чуть поменьше. Пусть будет 2.

Изображение выглядит как текст

Автоматически созданное описание

Также нам нужно закэшировать первоначальное положение актора. Для этого создаем дополнительную переменную типа FVector. Создаем ее в private секции и назовем InitialLocation:

Изображение выглядит как текст

Автоматически созданное описание

Возвращаемся в .cpp файл, переходим в BeginPlay и проинициализируем сразу нашу переменную InitialLocation. Присвоим ей возвращаемое значение функции GetActorLocation - это дополнительная функция для получения доступа к локации актора:

Изображение выглядит как текст

Автоматически созданное описание

Мы можем сделать это как делали раньше, то есть через GetActorTransform().GetLocation(), а можно сделать с помощью только одной функции.

Положение актора в пространстве будем менять каждый фрейм, поэтому переходим в функцию Tick и создаем переменную типа FVector, которая называется CurrentLocation. Присваиваем ей также значение функции GetActorLocation. Мы сейчас поменяем координату Z текущего положения актора в соответствии с нашей формулой и установим положение нашего актора в измененную CurrentLocation. Z0 в нашей формуле это InitialLocation.Z Далее у нас идет амплитуда. Воспользуемся нашей переменной amplitube. Нам нужно домножить амплитуду на значение синуса. Для этого воспользуемся специальной структурой в unreal, которая называется FMath. В данной структуре определены различные статические математические функции, в том числе синус, косинус и другие тригонометрические функции. Пишем frequency и умножаем на время:

Изображение выглядит как текст

Автоматически созданное описание

Нам нужно каким-то образом получить время. Можно воспользоваться количеством секунд, которые прошли с момента старта игры. Создаем переменную time и получим данное время с использованием указателя на глобальный объект мира игры. Чтобы получить данный указатель, нужно воспользоваться функцией GetWorld() - данная функция есть у каждого актора. У данного объекта существует метод, который называется GetTimeSeconds. Данный метод возвращает количество секунд, которые прошли с момента старта игры, то есть когда мы нажали кнопку play в едиторе, либо запустили .exe файл игры:

Изображение выглядит как текст

Автоматически созданное описание

Нам осталось обновить позицию нашего актора. Для этого воспользуемся методом, который называется SetActorLocation. Передаем данному методу в качестве аргумента переменную, которую мы только что вычислили - CurrentLocation. Данная функция меняет текущее положение актора в мире, и мы с вами только что реализовали функцию движения по синусоиде:

Изображение выглядит как текст

Автоматически созданное описание

Запускаем editor. Итак, посмотрим, что у нас получилось. Нажимаем кнопку play. Наши акторы совершают синхронные колебания относительно их начального положения. Это нормально, потому что параметры по умолчанию абсолютно одинаковые. Давайте это подправим. Выбираем первого актора. Находим категорию Movement и ставим амплитуду в 100, а частоту в 4. Второму актору проставляем амплитуду в 250, а частоту в 5. Запускаемся. Каждый актор двигается в соответствии со своими параметрами. Нажимаем Esc. Нажимаем на актора. Заходим в категорию Movement и мы видим, что мы изменили параметры и напротив каждого поля появилась справа желтенькая стрелочка. Если мы на нее нажмем, то значение возвращается к дефолтному. Это иногда бывает очень удобно.