1. Создаем enum для управления перемещением (по синусоиде или статично)

2. Создаем переменную данного enum у нашего актора + значение по умолчанию

3. Как реализуем переключение и в какой функции между значениями enum

4. Как сделать так, чтобы наш enum корректно отражался в эдиторе – синтаксис макроса, что принимает, что означает, с чего начинаются все типы enum в UE и какой тип имеют, добавляем UPROPERTY нашей переменной

5. Создание структуры в UE, макрос – что принимает, еще один макрос как у классов (почти)

6. Создаем переменную данной структуры в акторе и заносим туда наши переменные для перемещения

7. Изменение обращений везде из-за введения структуры

8. Как отображается в Editor структура данных

9. Рефакторинг кода – как должны начинаться все переменные и функции в UE

1. В данном уроке мы с вами рассмотрим enum. Cделаем наш enum класс доступным в едиторе. И также поговорим о структурах. Начнем с enum. Создаем enum класс. Назовем наш тип MovementType. Данный enum будет отвечать за тип движения нашего актора. Первый элемент будет движением по синусоиде, как сейчас, у При этом значении актор будет находиться в состоянии покоя:

Изображение выглядит как текст

Автоматически созданное описание

2. Создаем переменную типа MovementType. Назовем ее MoveType и присвоим значение по умолчанию MovementType::Static:

Изображение выглядит как текст

Автоматически созданное описание

3. Переходим в .cpp файл. В функции Tick сделаем switch по нашему enum MovementType. В блок синуса добавим код, который мы написали в прошлом уроке. Убираем его снизу Tick и вставляем в блок case. Блок с MovementType::Static оставляем пустым, потому что мы не хотим чтобы наш актор двигался и default также оставляем пустым:

Изображение выглядит как текст

Автоматически созданное описание

Итак, мы с вами уже создали работающий пример. Если мы сейчас скомпилируем, то у нас все будет работать. Мы можем поменять дефолтное значение на синус и тогда наш актор будет двигаться по синусоиде, либо оставить дефолтное как есть static и тогда actor у нас положение менять не будет.

4. Возвращаемся в .h файл. Cейчас мы сделаем так, чтобы данный тип enum отображался корректно в едиторе. Для этого нам нужно сделать несколько шагов. Первый - это добавление обвеса из макроса UENUM сверху нашего enum класса. Данный макрос сгенерирует дополнительный мета-код, который расширяет функциональность классических enum классов. Указываем аргументу UENUM BlueprintType. Данный параметр означает, что наш класс станет доступным в едиторе, в блюпринтах, и мы напрямую в едиторе сможем менять значения нашей переменной MoveType. Следующий момент. Все типы enum в Unreal Engine должны начинаться с большой буквы E и все enum строго типизированы. Они имеют тип uint8 uint8 это алиас, typedef unsigned char. По сути это означает, что максимальное количество элементов у enum может быть 255:

Изображение выглядит как текст

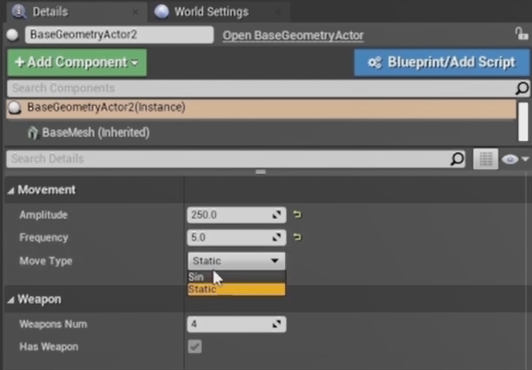
Автоматически созданное описание

Давайте поменяем везде MovementType на EMovementType. В .h файле и также в .cpp файле. Делаем нашу переменную MoveType доступной в едиторе. Для этого копируем property и вставляем над переменной MoveType:

Изображение выглядит как текст

Автоматически созданное описание

Запускаем едитор. Нажимаем play. Наши акторы не двигаются - это нормально, потому что у нас по умолчанию стоит MoveType = static. Нажимаем на actor, идем в details panel, находим наш тип MoveType и мы видим, что enum отображается, как combobox:



Мы можем поменять значение static на sin. Запускаем. Один actor двигается. Другой в состоянии покоя.

5. Переходим с вами к созданию структур в unreal engine. Структуры в классическом C++ практически ничем не отличаются от классов. Единственное различие - это модификатор доступа по умолчанию. К членам и функциям у структур он public, у классов private. В unreal структуры в основном используются для организации данных, то есть для объединения переменных в какие-то связанные объекты. Мы сейчас создадим с вами структуру, которая объединяет переменные из категории Movement нашего актора, то есть амплитуду, частоту и тип движения. Hабираем struct, назовём нашу структуру GeometryData По аналогии с enum нам нужно сверху добавить специальный макрос, который называется USTRUCT. Также декларируем, что наша структура будет доступна в блюпринтах. Указываем параметр макросу BlueprintType. Все типы структур в unreal начинаются с заглавной буквы F. И последний макрос, который нам нужно добавить называется GENERATED\_USTRUCT\_BODY:

Изображение выглядит как текст

Автоматически созданное описание

Данный макрос очень похож на GENERATED\_BODY - макрос актора. Убираем наши переменный из тела BaseGeometryActor и вставляем их в тело нашей структуры GeometryData:

Изображение выглядит как текст

Автоматически созданное описание

6. В акторе создаем переменную типа FGeometryData. Назовем её GeometryData. Делаем переменную доступной в едиторе, для этого добавляем сверху наш макрос с uproperty. Изменим категорию с Movement на GeometryData:

Изображение выглядит как текст

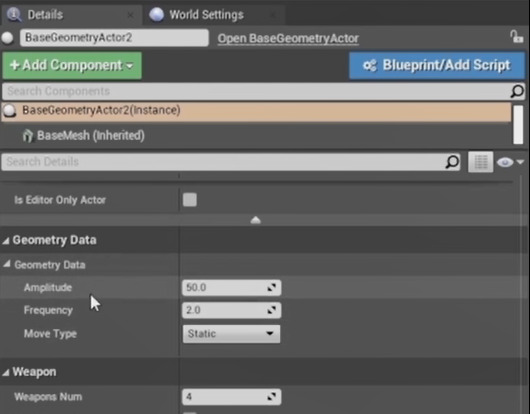
Автоматически созданное описание

7. Мы создали с вами объект типа FGeometryData. Перемещаемся в .cpp файл и в основном коде нам нужно поменять все наши переменные, которые мы переместили в наш объект, на переменные данного объекта:

Изображение выглядит как текст

Автоматически созданное описание

8. Итак, все у нас готово. Запускаем editor. Выбираем любой из наших акторов, находим в details panel категорию GeometryData и видим, каким образом отображается в editor структура данных:



С точки зрения логики в нашем коде ничего не изменилось, а с точки зрения организации данных теперь все наши переменные объединены в структуру. Теперь, например, мы можем передавать все наши переменные не по отдельности, а целым объектом GeometryData.

9. И последний момент - отрефакторил наш код, создал функцию HandleMovement, убрал туда всю логику движения нашего актора, переименовал переменную Time. Теперь она начинается с заглавной буквы:

Изображение выглядит как текст

Автоматически созданное описание

Также с заглавной буквы теперь начинаются все наши функции Print:

Изображение выглядит как текст

Автоматически созданное описание

Это относится к стандартам кодирования. Мы будем использовать стандарт Unreal. Все переменные, все функции будут начинаться с заглавной буквы. Об этом подробнее поговорим в следующих разделах, но с данного момента давайте стараться его соблюдать.