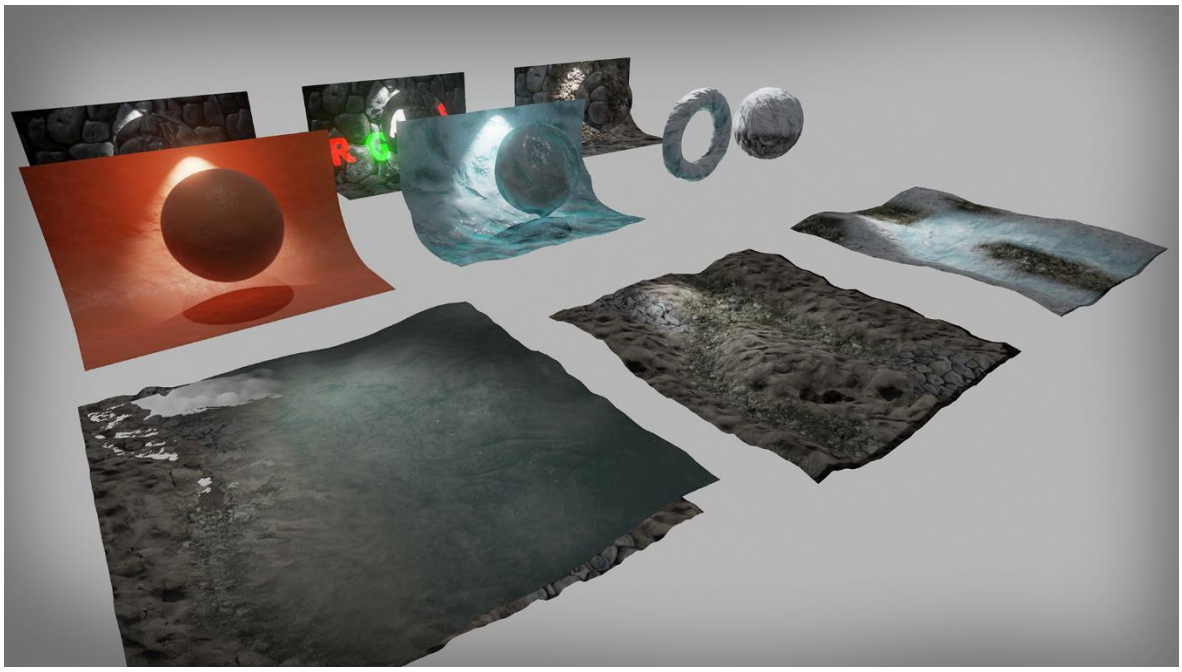
Туториал по Unreal Engine. Часть 3: материалы



Как и в реальном мире, в играх есть множество объектов, каждый со своим внешним видом. В Unreal Engine этот внешний вид зависит от материалов. Какой цвет имеет объект? Прозрачен ли он? Блестит ли? Все эти свойства задаются материалами.

Материалы используются почти для любого визуального элемента в Unreal Engine. Материалы можно наносить на любые объекты, включая меши, частицы и элементы UI.

В этой части туториала вы научитесь следующему:

- Управлять текстурами, изменяя их яркость и цвет
- Использовать экземпляры материалов для быстрого создания вариаций
- Использовать динамические экземпляры материалов для изменения цвета аватара при собирании игроком предметов

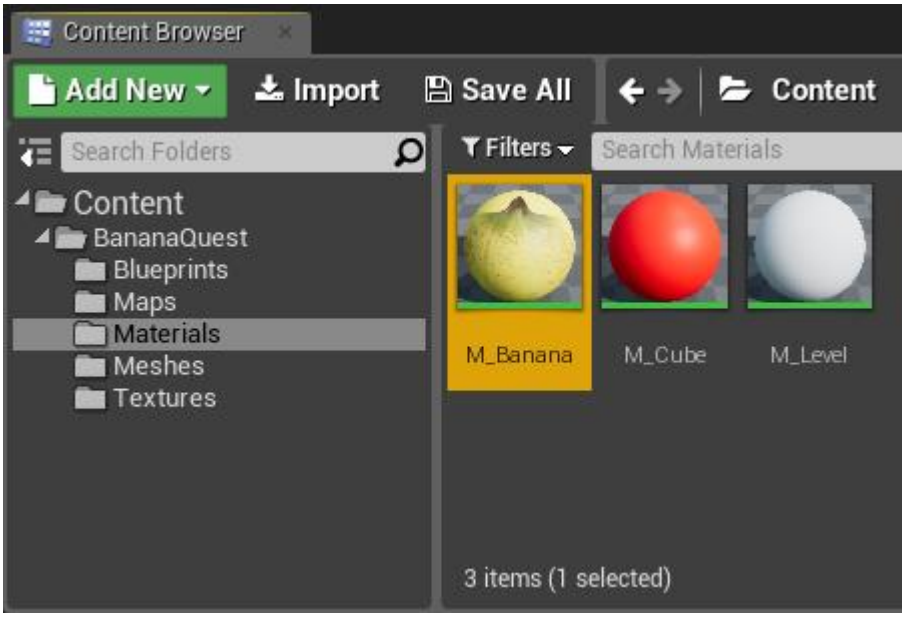
Приступаем к работе

Скачайте https://koenig-media.raywenderlich.com/uploads/2017/07/BananaCollectorStarter.zip и распакуйте её. Чтобы открыть проект, перейдите в папку проекта и откройте *BananaCollector.uproject*.

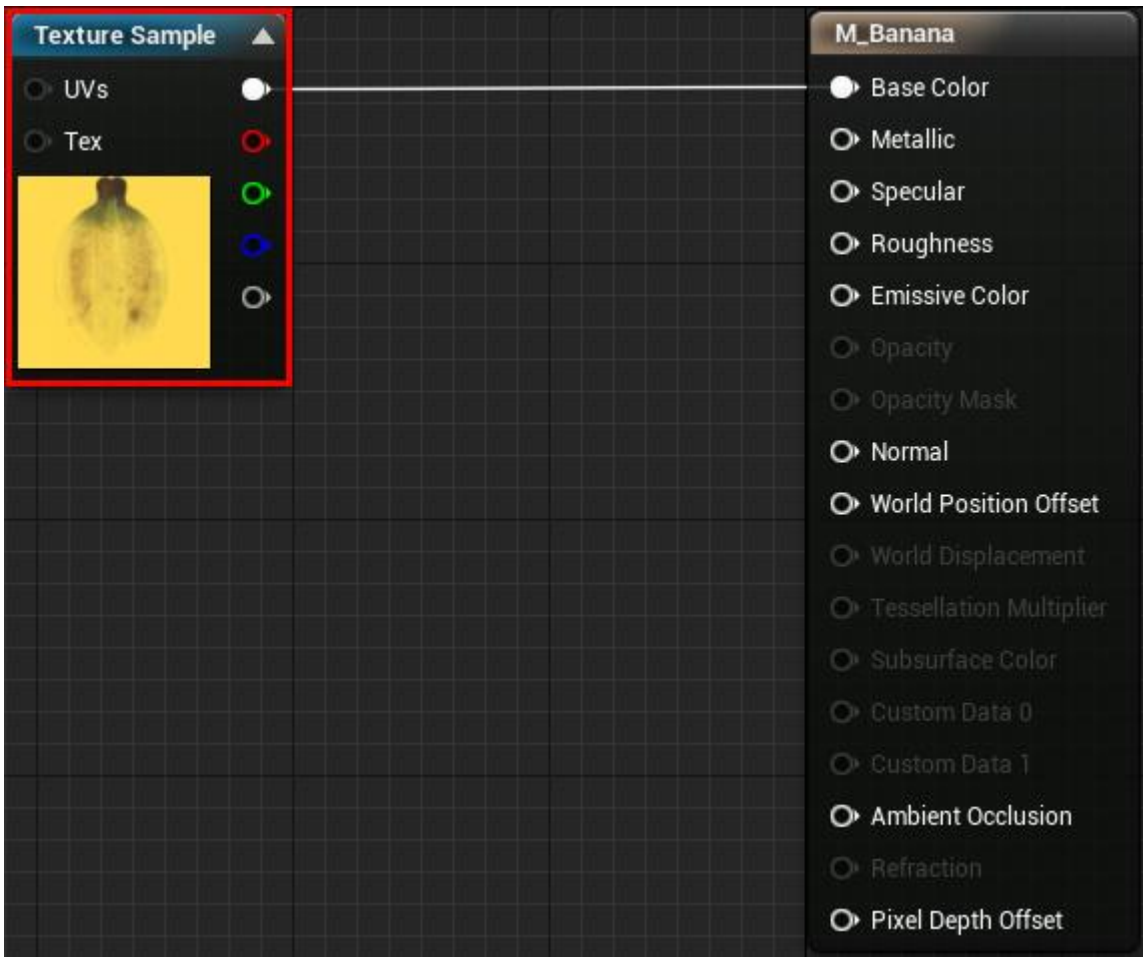
Примечание: если откроется окно, сообщающее, что проект создан в более ранней версии Unreal editor, то всё в порядке (движок часто обновляется). Можно или выбрать опцию создания копии, или опцию преобразования самого проекта.

Вы увидите небольшую область, в которой расположены бананы. Нажмите *Play*, чтобы управлять красным кубом с помощью клавиш *W*, *A*, *S* и *D*. Собирать бананы можно, касаясь их.

Для начала давайте изменим материал банана, чтобы менять его яркость. Перейдите в папку *Materials* и *дважды нажмите* на *M_Banana*, чтобы открыть его в редакторе материалов.



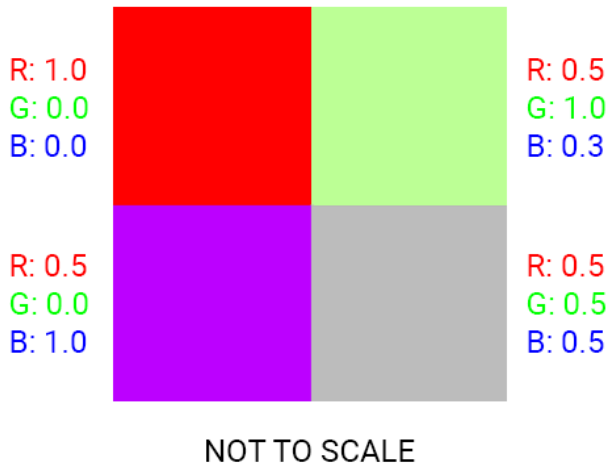
Чтобы отрегулировать яркость банана, нам нужно управлять его текстурой.



Управление текстурами

По своей сущности текстура является изображением, а изображение — это набор пикселей. В цветном изображении цвет пикселя определяется его *красным ®*, *зелёным (G)* и *синим (B)* каналами.

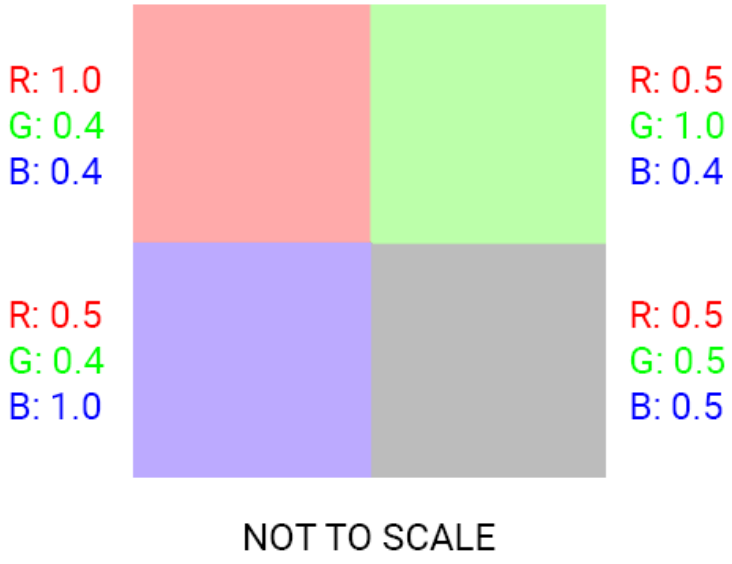
Ниже представлен пример изображения 2×2 с указанными для каждого пикселя значениями RGB.



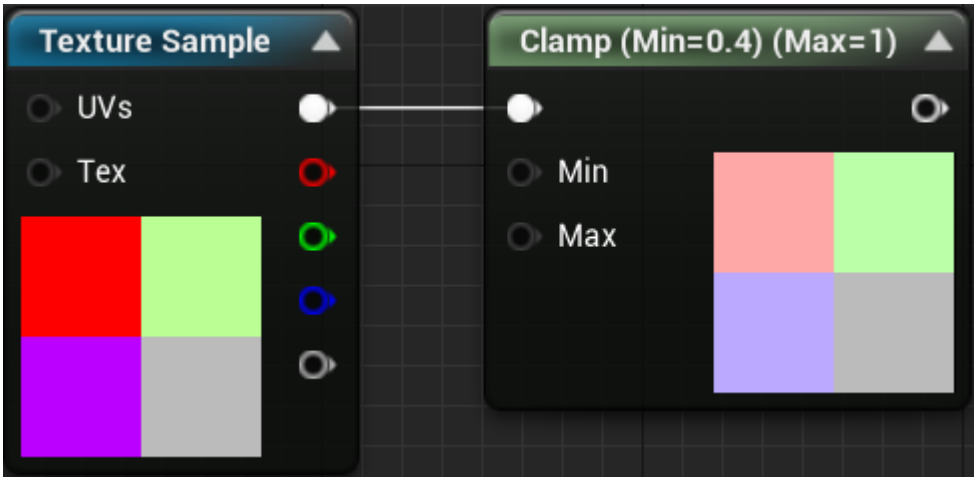
Примечание: в Unreal Engine каналы RGB имеют интервал значений от 0.0 до 1.0. Однако в большинстве других приложений каналы RGB имеют интервал от 0 до 255. Это просто разные способы отображения одинаковой информации и они не означают, что интервал цветов Unreal Engine меньше.

Управление текстурой осуществляется выполнением операций над каждым пикселем текстуры. Операции могут быть даже простыми, например, добавлением значения к каналам.

Ниже представлен пример ограничения каждого канала в интервале от 0.4 до 1.0. Благодаря этому увеличивается минимальное значение каждого канала, что делает каждый цвет ярче.



Вот как это можно сделать в редакторе материалов:

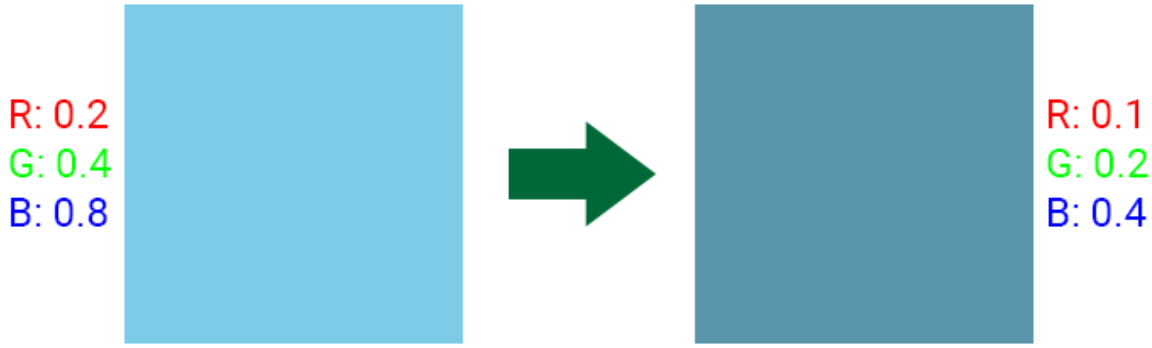


Теперь можно использовать нод *Multiply* для регулировки яркости текстуры.

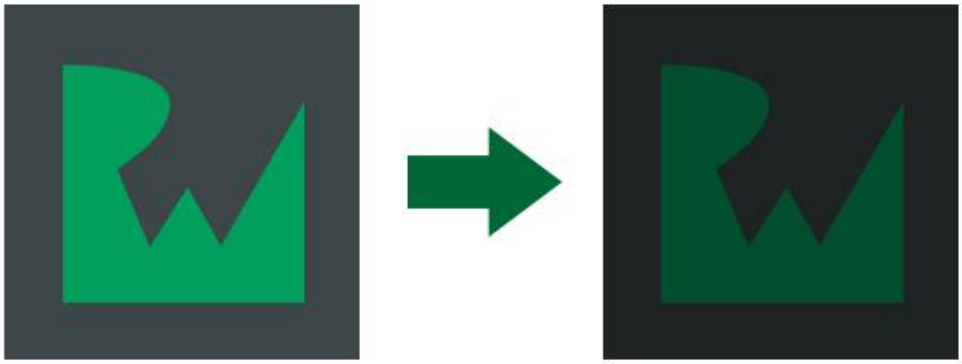
Нод Multiply

Нод Multiply делает именно то, что следует из его названия: он умножает один вход на другой вход.

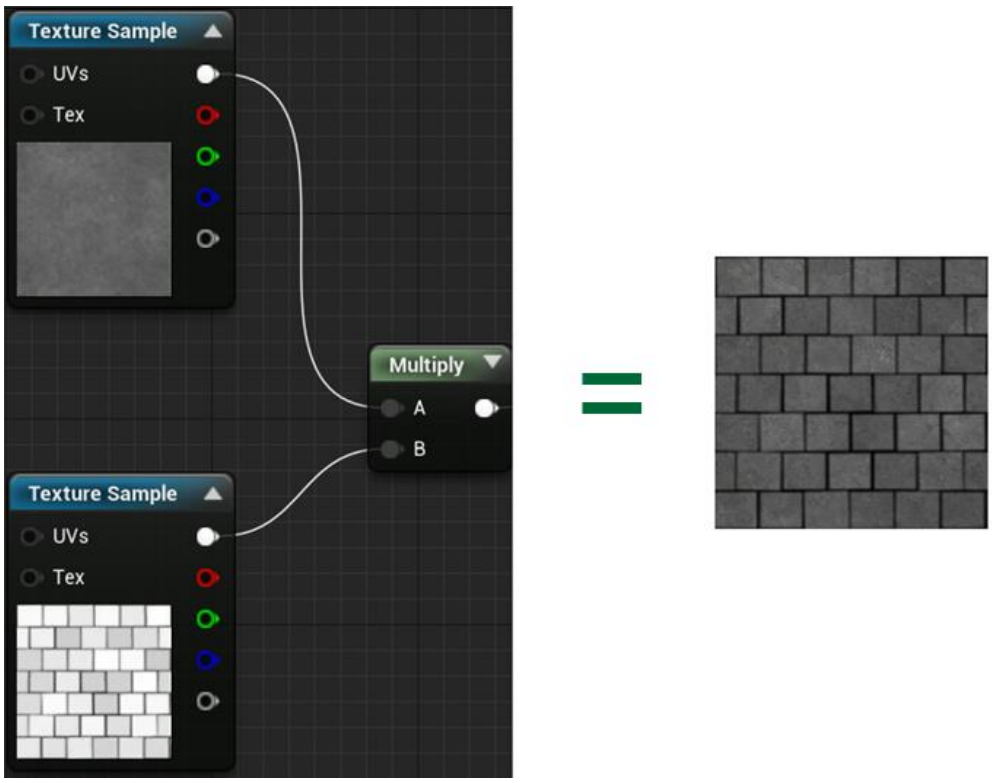
С помощью умножения можно изменять яркость пикселя, не затрагивая его оттенок или насыщенность. Ниже представлен пример уменьшения яркости наполовину умножением каждого канала на 0.5.



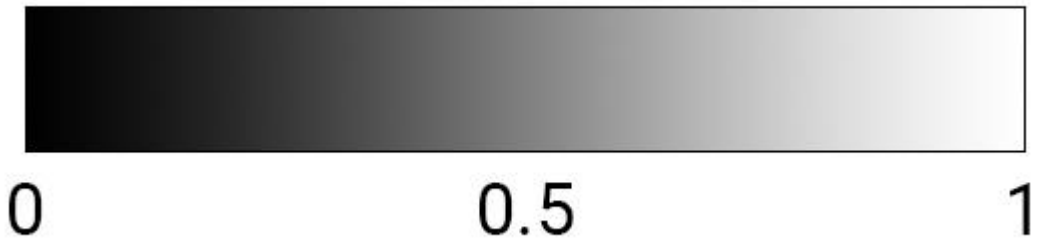
Выполнив эту операцию для каждого пикселя, мы можем изменить яркость всей текстуры.



Мы не будем рассматривать это в туториале, но вы можете использовать нод Multiply в сочетании с текстурой-маской. С помощью маски можно указать, какие области базовой текстуры должны быть темнее. Вот пример наложения маски из текстуры плитки на текстуру камня:



Наложение маски работает, потому что градации серого представляют собой интервал от 0 (чёрный) до 1 (белый).

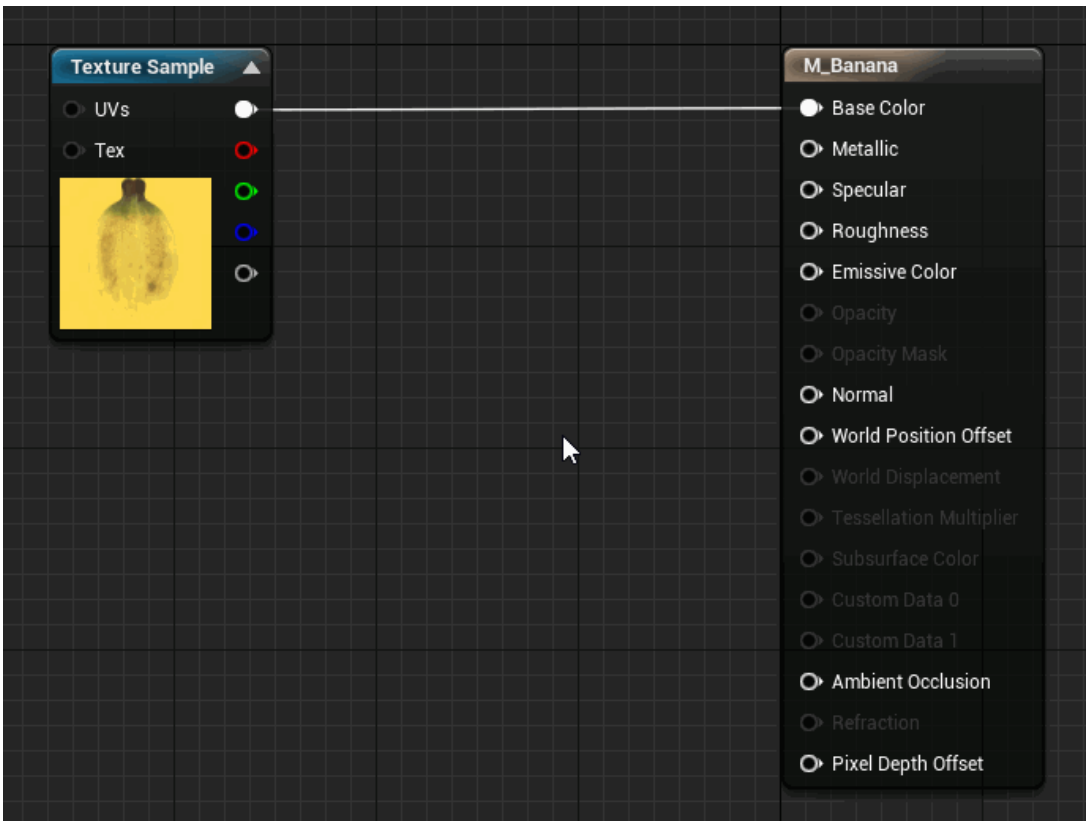


Белые области имеют полную яркость, потому что каналы умножены на 1. Серые области темнее, потому что каналы умножаются на значения меньше 1. Чёрные области не имеют яркости, потому что каналы умножены на 0.

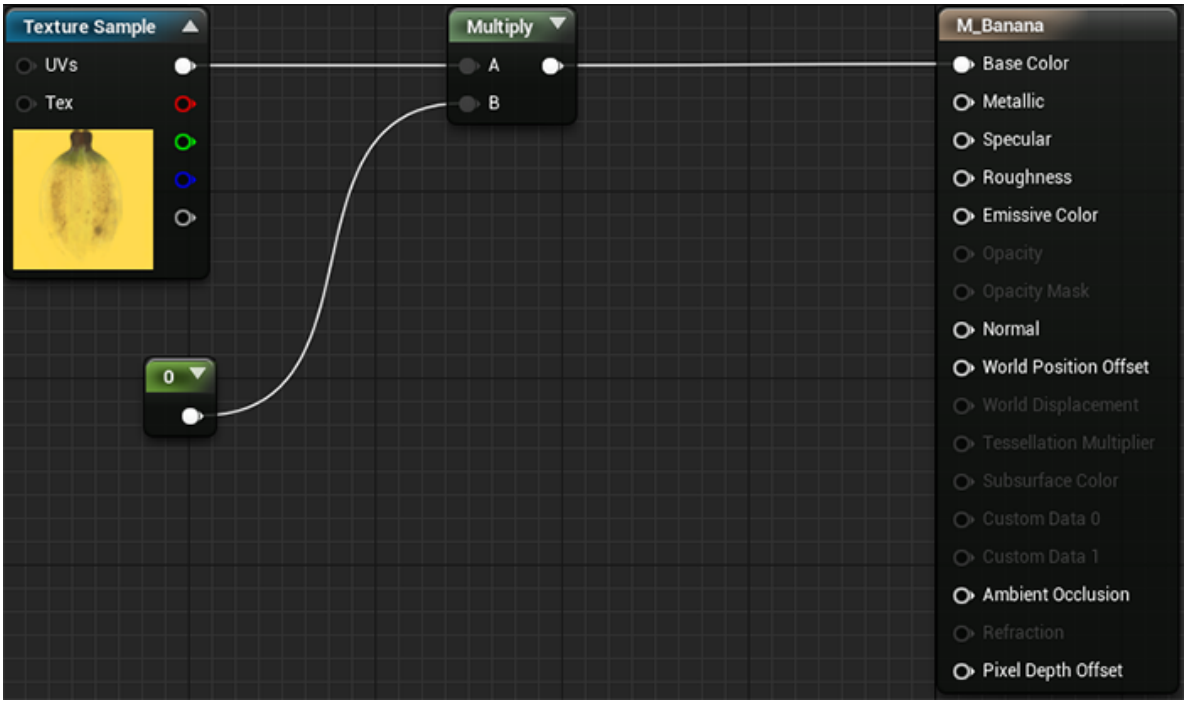
Теперь настало время воспользоваться нодом Multiply.

Регулировка яркости текстур

Разорвите связь между нодом *Texture Sample* и контактом *Base Color*. Это можно сделать, *нажав правой клавишей мыши* на любой контакт и выбрав *Break Link(s)*. Или же можно удерживать клавишу *Alt* и *нажать левой клавишей мыши* на соединяющей линии.



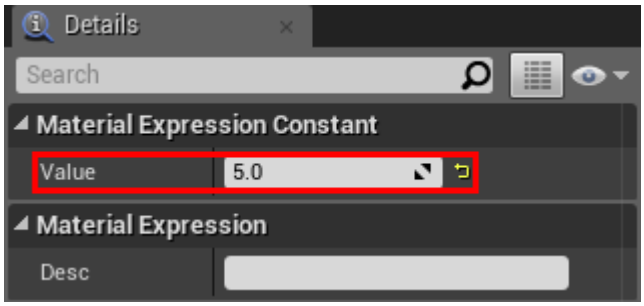
Создайте нод *Multiply* и *Constant*. Их можно создать быстро, удерживая клавишу *M* (для нода Multiply) или клавишу *1* (для нода Constant) и *левой клавишей мыши* на пустом пространстве в графе. После этого соединение должно выглядеть следующим образом:



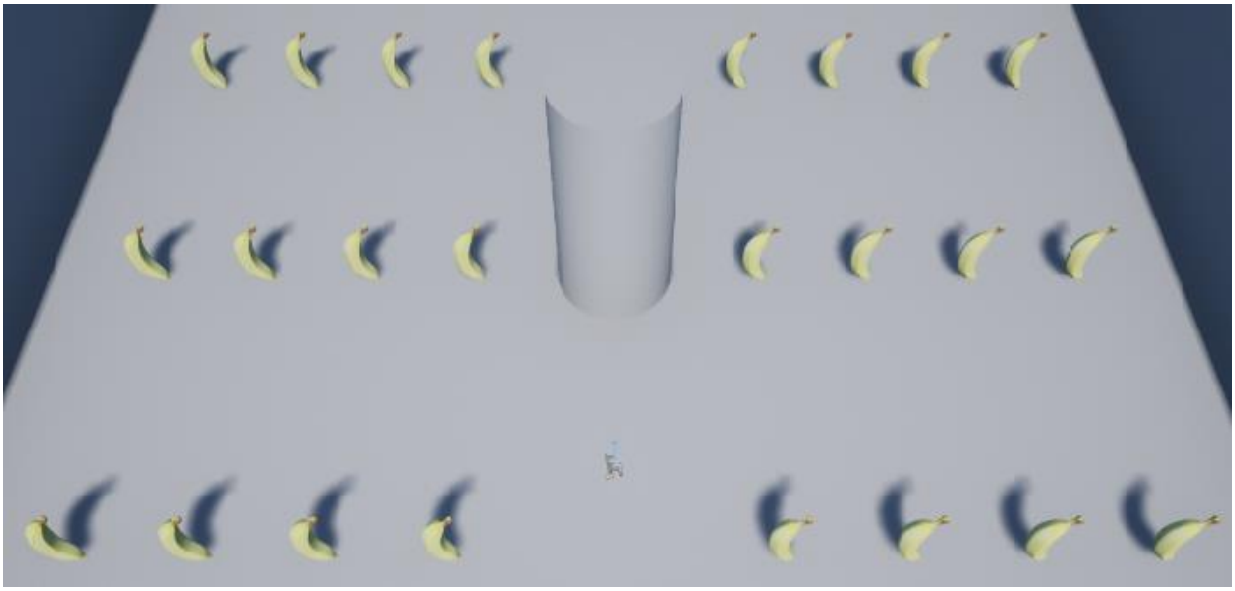
Такая схема будет итеративно обходить каждый пиксель и умножать каждый канал на значение нода *Constant*. Наконец, получившаяся текстура будет выведена как *Base Color*.

Пока получившаяся текстура будет чёрной, потому что множитель равен нулю. Для

изменения значения множителя выберите нод *Constant* и перейдите в панель Details. Введите в поле *Value* значение *5*.



Нажмите *Apply* и вернитесь к основному редактору. Вы увидите, что бананы теперь стали намного ярче.



Давайте разнообразим сцену, добавив бананы разного цвета. Конечно, можно создавать новый материал для каждого цвета, но проще создать *экземпляр материала*.

Об экземплярах материалов

Экземпляр материала — это копия материала. Все изменения, вносимые в базовый материал, отображаются также и на экземпляре материала.

Экземпляры материалов очень удобны, потому что в них можно вносить изменения без рекомпиляции. При нажатии Apply на материале вы могли заметить, что показывается уведомление о том, что компилируются шейдеры.

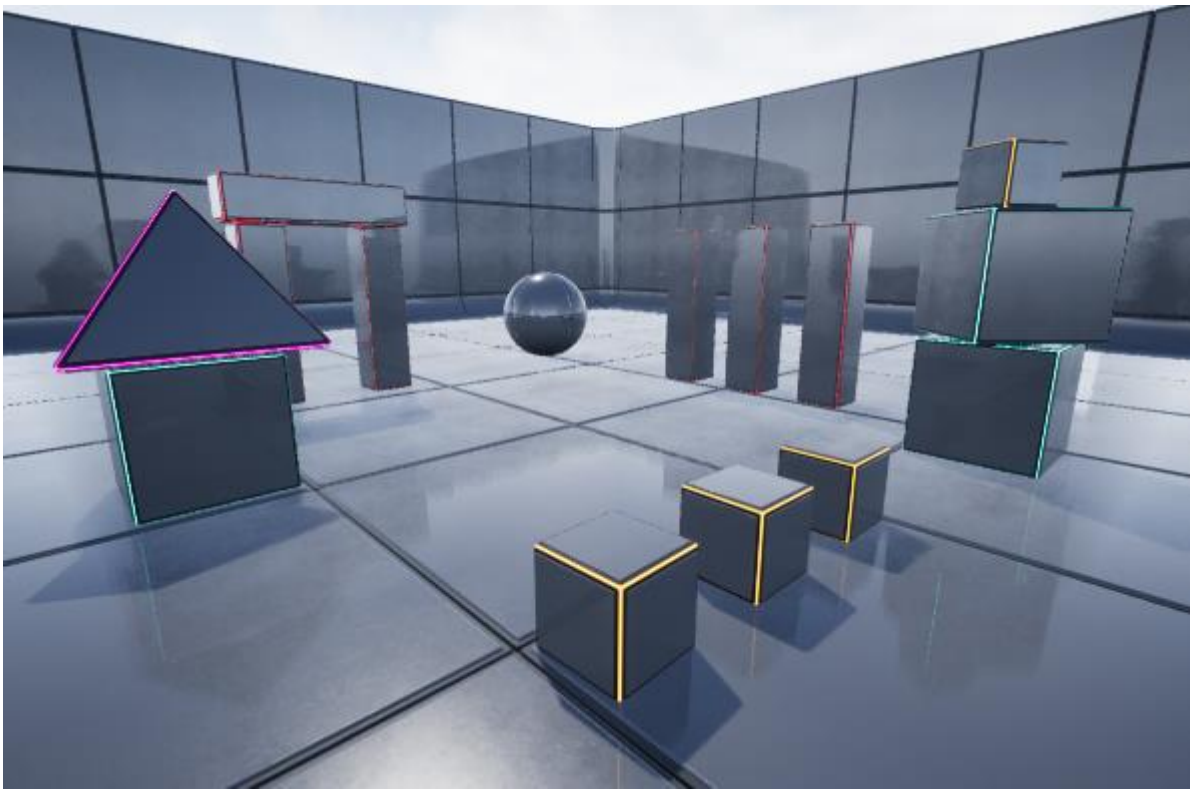
Для простых материалов этот процесс занимает всего несколько секунд. Однако для сложных материалов время компиляции может быть значительно больше.

Экземпляры материалов удобно применять в следующих случаях:

- При наличии сложного материала, в который нужно быстро внести изменения

- Необходимо создать вариации базового материала. Это может быть что угодно — изменение цвета, яркости или даже самой текстуры.

Ниже показан пример сцены, в которой используются экземпляры материалов для создания вариаций цвета. Все экземпляры имеют одинаковый базовый материал.

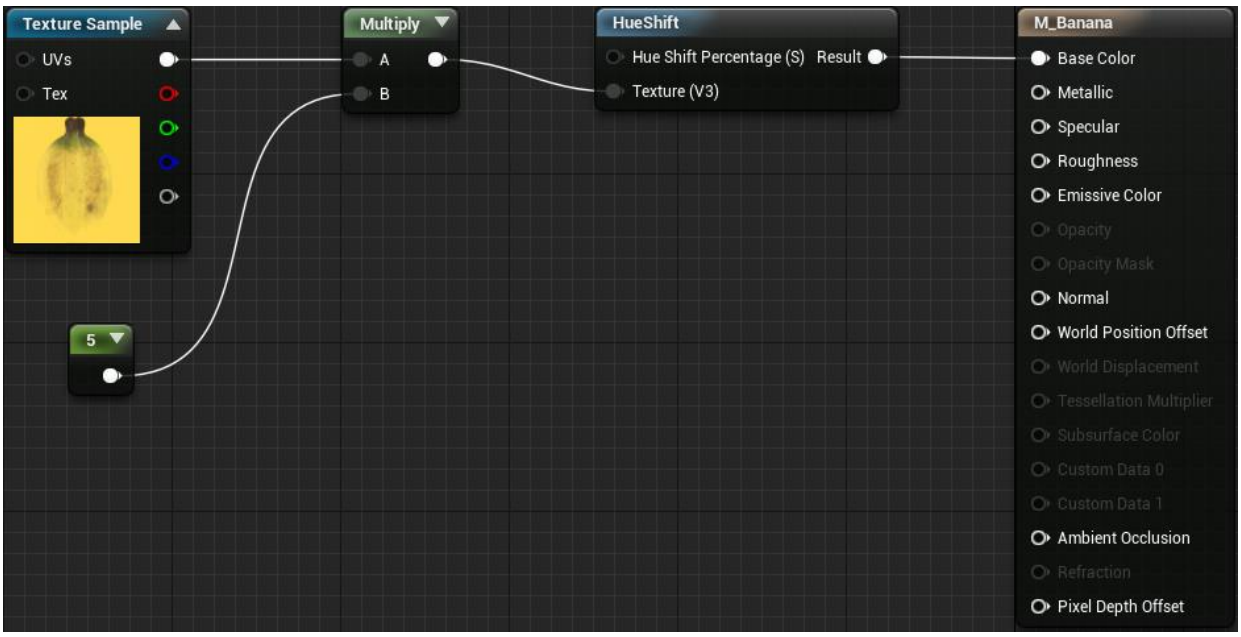


Прежде чем создать экземпляр, нам нужно создать *параметры* в базовом материале. Они будут отображаться в экземпляре материала и позволят настраивать свойства материала.

Создание параметров материалов

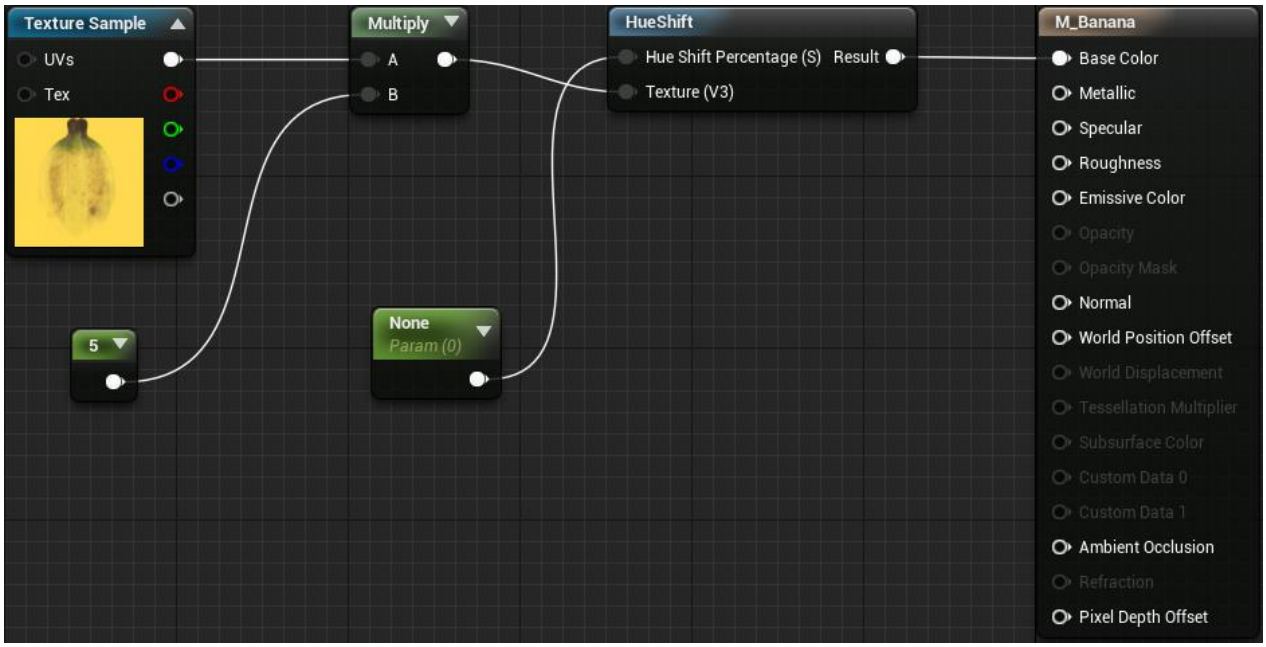
Вернитесь в редактор материалов и выберите материал *M_Banana*.

Во-первых, нам нужен нод, меняющий оттенок текстуры. Для этого можно использовать нод *HueShift*. Добавьте его в граф и соедините следующим образом:

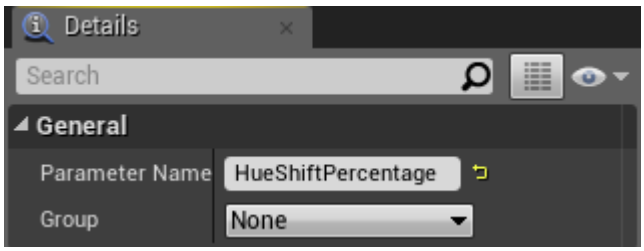


Забыли, как это сделать? Решение внутри

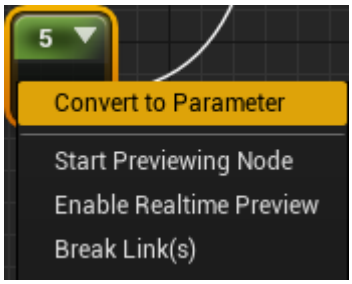
Теперь нам нужно создать нод *Scalar Parameter*. В этом ноде хранится единственное значение и его можно будет изменять в экземпляре материала. Этот нод можно быстро создать, удерживая клавишу *S* и *нажав левой клавишей мыши* на пустом пространстве в графе. После создания нода соедините его с контактом *Hue Shift Percentage (S)* нода *HueShift*.



Неплохо также будет давать названия параметрам. Выберите нод *Scalar Parameter* и перейдите в панель Details. Измените *Parameter Name* на *HueShiftPercentage*.



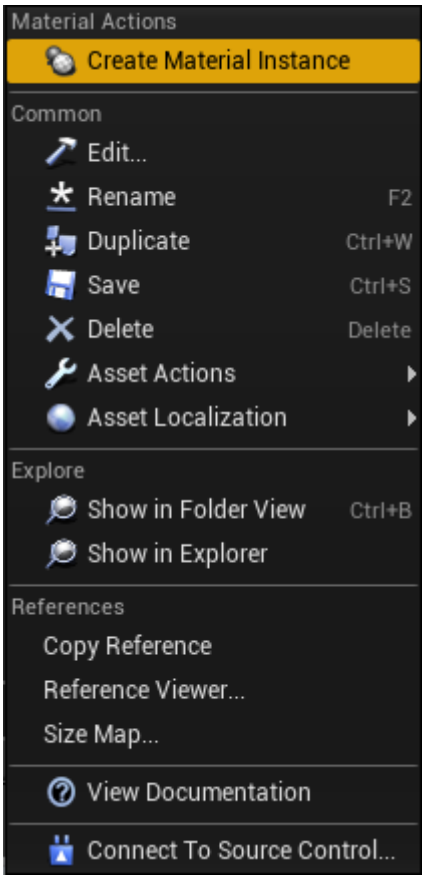
Можно также преобразовать ноды Constant в Scalar Parameter. *Нажмите правой клавишей мыши* на добавленный ранее нод *Constant*, а затем выберите *Convert to Parameter*. Переименуйте параметр в *Brightness*.



Теперь мы создадим экземпляр материала.

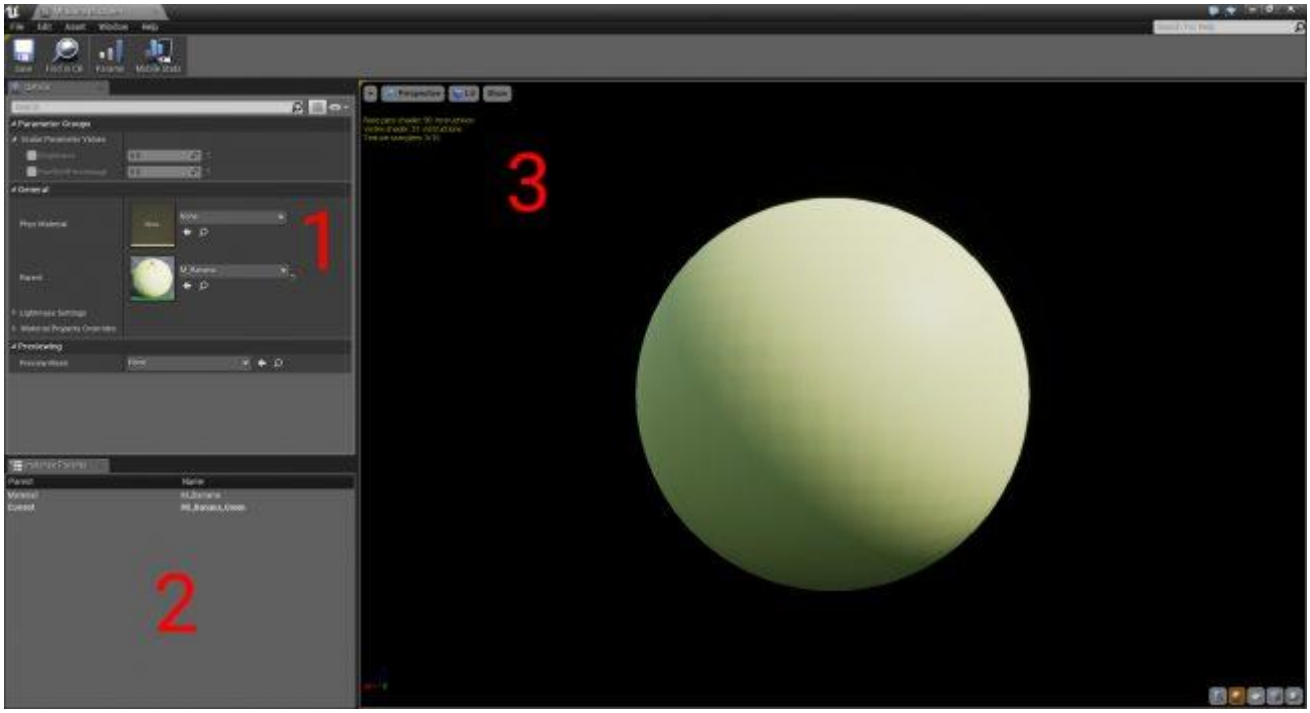
Создание экземпляра материала

Перейдите в Content Browser и выберите папку *Materials*. *Нажмите правой клавишей мыши* на *M_Banana* и выберите *Create Material Instance*. Переименуйте новый ассет в *MI_Banana_Green*.



Дважды нажмите на *MI_Banana_Green*, чтобы открыть его. При этом он откроется в редакторе экземпляров материалов.

Редактор экземпляров материалов состоит из трёх панелей:



1. *Details:* здесь отображаются параметры и другие общие настройки
2. *Instance Parents:* отображает список родительских материалов текущего экземпляра. В этом случае единственным родительским материалом является *M_Banana*
3. *Viewport:* содержит меш предварительного просмотра, на котором показывается экземпляр материала. Можно вращать камеру, удерживая *левую клавишу мыши* и *перемещая* мышь. Масштабирование выполняется *прокруткой колёсика мыши*.

Чтобы увидеть изменения на меше банана, перейдите в панель Details и найдите раздел *Previewing*. *Нажмите левой клавишей мыши* на *раскрывающийся список* рядом с *Preview Mesh* и выберите *SM_Banana*. Теперь вы будете видеть вместо сферы меш банана.

Теперь мы изменим параметры, чтобы сменить цвет банана на зелёный. Чтобы сделать параметры редактируемыми, *нажмите левой клавишей мыши* на *флажок* рядом с каждым параметром.

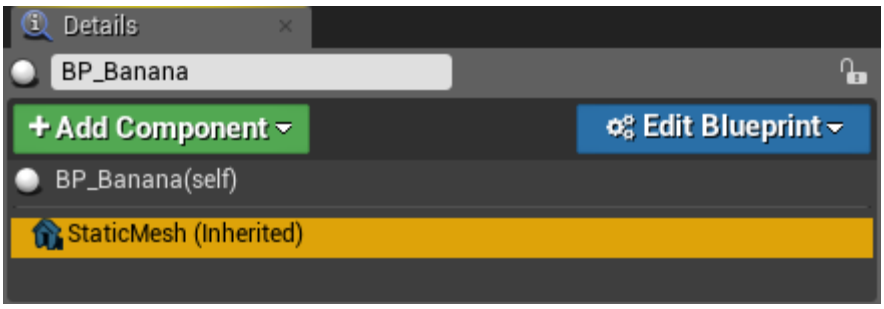
Укажите для *Brightness* значение *0.5*, а для *HueShiftPercentage* — значение *0.2*. В результате у вас получится следующее:

Мы создали экземпляр материала, теперь можно применить его на некоторые из бананов! Закройте экземпляр материала и перейдите во Viewport основного редактора.

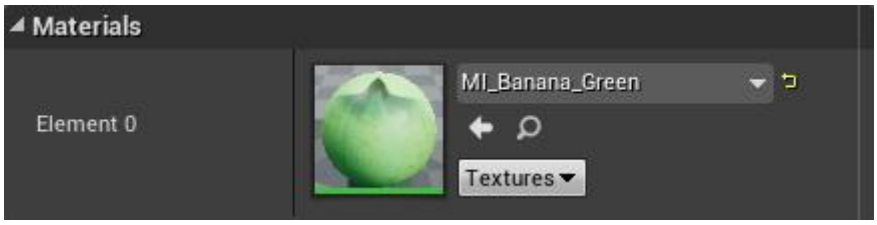
Применение экземпляра материала

Располагаемые в сцене акторы можно редактировать индивидуально. Это значит, что если вы измените материал для одного банана, то это не повлияет на остальные. Можно воспользоваться этим, чтобы изменить цвет некоторых бананов на зелёный.

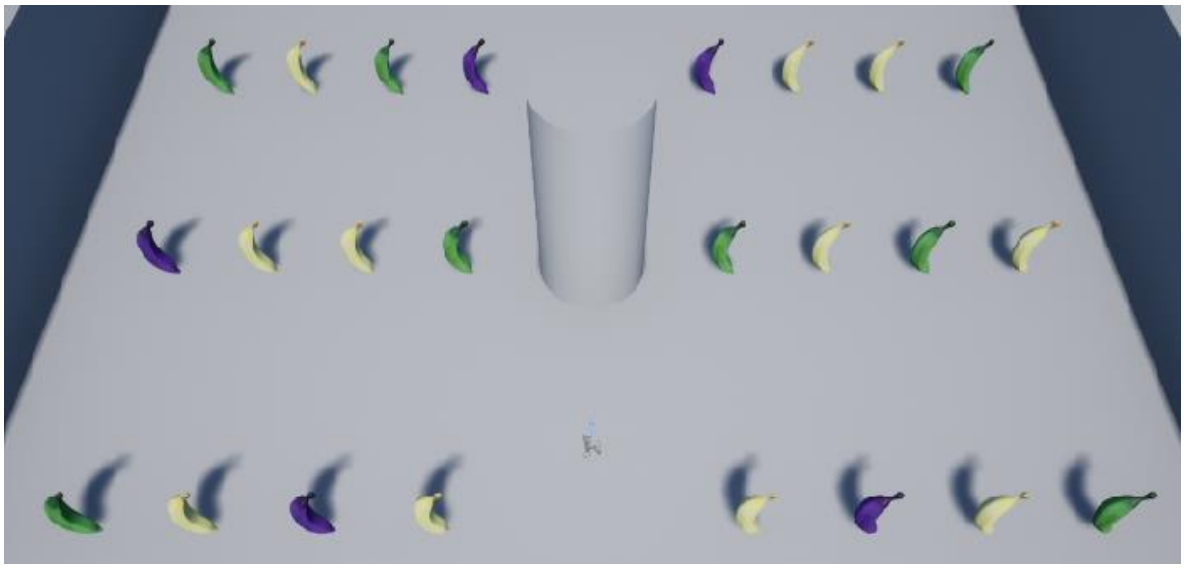
Выберите любой банан и перейдите в панель Details. В списке компонентов выберите компонент *StaticMesh*.



В панели Details отобразятся свойства компонента *StaticMesh*. Измените материал на *MI_Banana_Green*.



Повторите процесс ещё несколько раз для лучшего распределения жёлтых и зелёных бананов. Попробуйте создать ещё один экземпляр материала, чтобы у вас были и фиолетовые бананы!



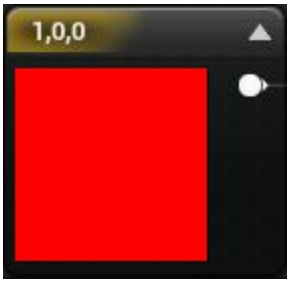
Динамически изменяемые материалы

Материалы не обязательно должны быть только косметическими: можно использовать их и для помощи в геймдизайне. Сейчас мы разберёмся, как динамически изменять цвет куба с белого до красного, когда игрок собирает бананы.

Прежде чем создавать экземпляр материала, нужно будет настроить материал куба.

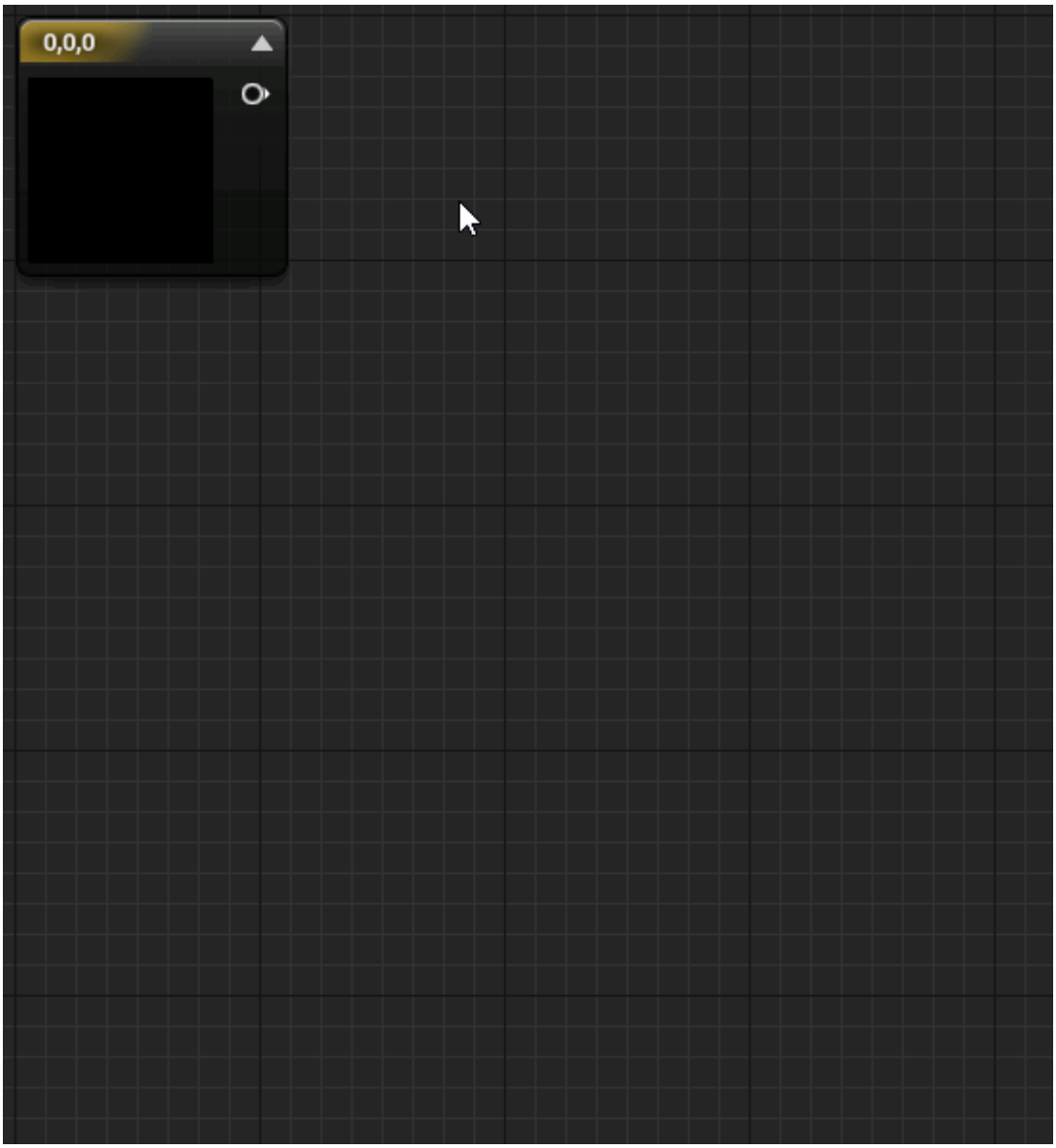
Убедитесь, что находитесь в папке *Materials* и *дважды щёлкните* на *M_Cube*, чтобы открыть его.

Во-первых, нам нужен способ создания цветов. К ноду *Base Color* подсоединён нод *Constant3Vector*. Эти ноды идеально подходят для выбора цветов, потому что у них есть красный, зелёный и синий каналы.

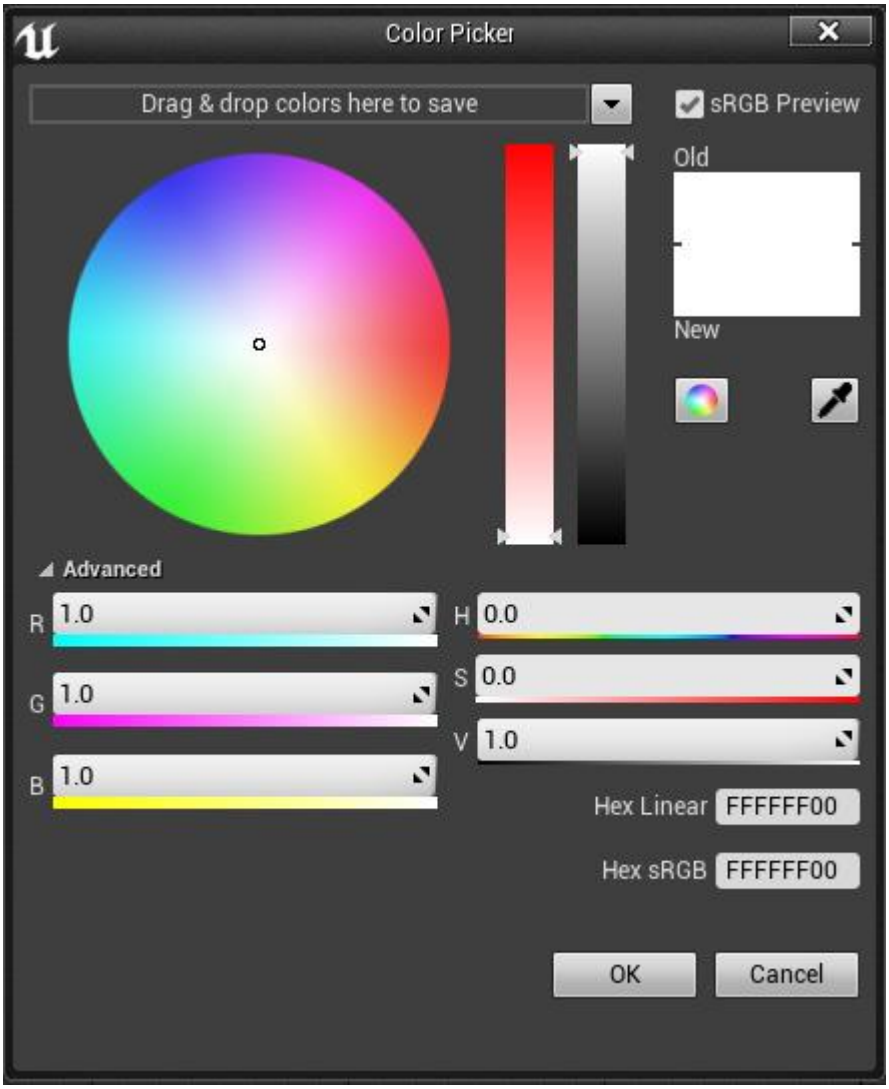


Поскольку красный цвет уже был создан, мы создадим белый цвет. Добавьте ещё один нод *Constant3Vector*. Это можно сделать быстро, удерживая клавишу *3* и *нажав левой клавишей мыши* на пустом пространстве в графе.

Откройте инструмент выбора цвета, *дважды щёлкнув* на нод *Constant3Vector*.



Выберите белый цвет, или с помощью ползунков, или введя значение *1.0* для каналов *R*, *G* и *B*. Затем нажмите кнопку *OK*.



Чтобы изменить цвет с белого на красный, нам нужен способ создания плавного перехода между ними. Есть простой способ его реализации — *линейная интерполяция*.

Что такое «линейная интерполяция»?

Линейная интерполяция — это способ нахождения значений между A и B. Например, можно использовать линейную интерполяцию для нахождения значения посередине между 100 и 200.

Линейная интерполяция становится ещё более мощным инструментом, если можно управлять alpha. Мы можем воспринимать alpha как процент между A и B. Alpha = 0 вернёт значение A, а alpha = 1 — значение B.

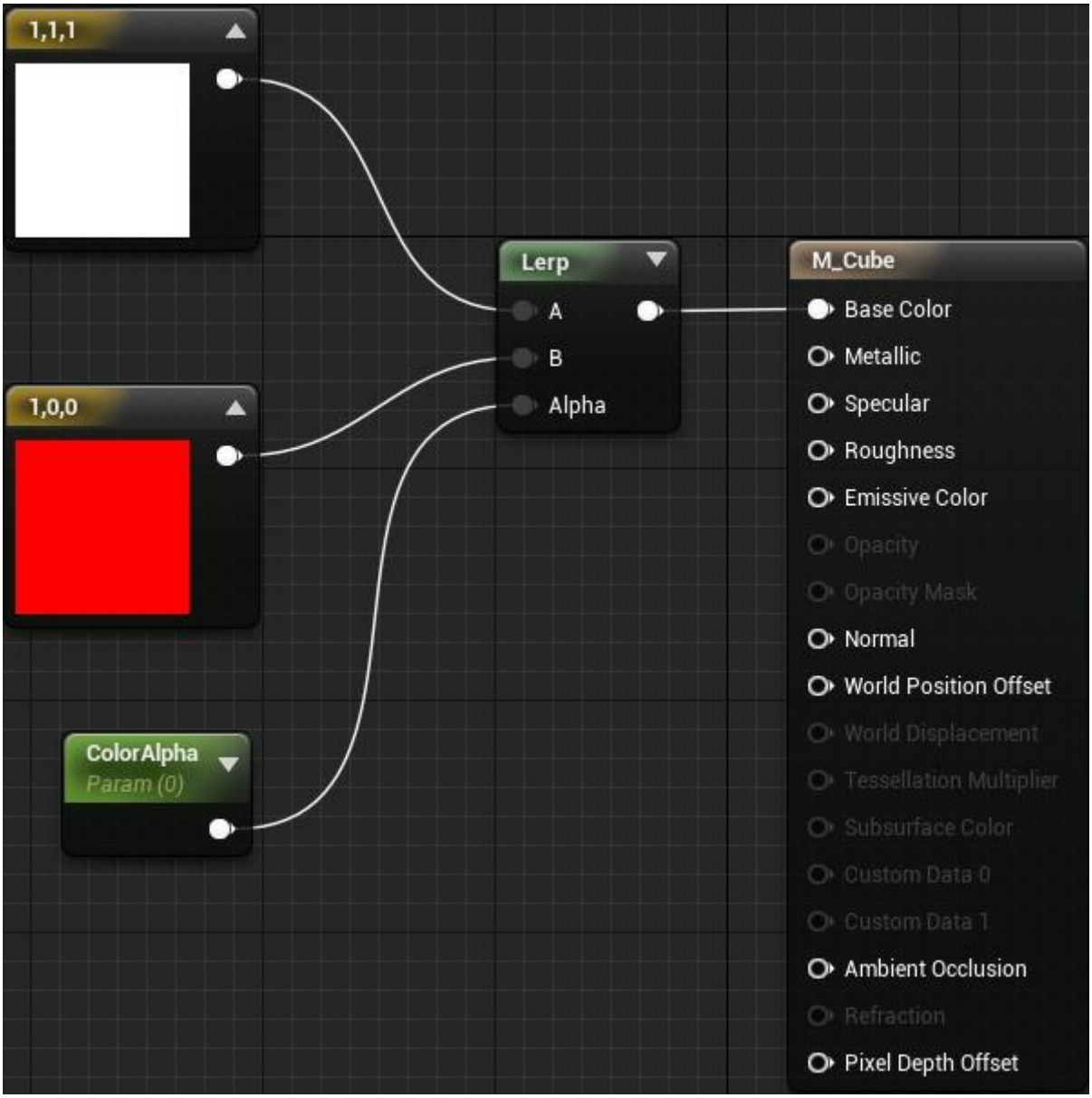
Например, можно увеличивать alpha постепенно, чтобы плавно перемещать объект из точки A в точку B.

В этом туториале мы будем управлять alpha с помощью количества собранных бананов.

Использование нода LinearInterpolate

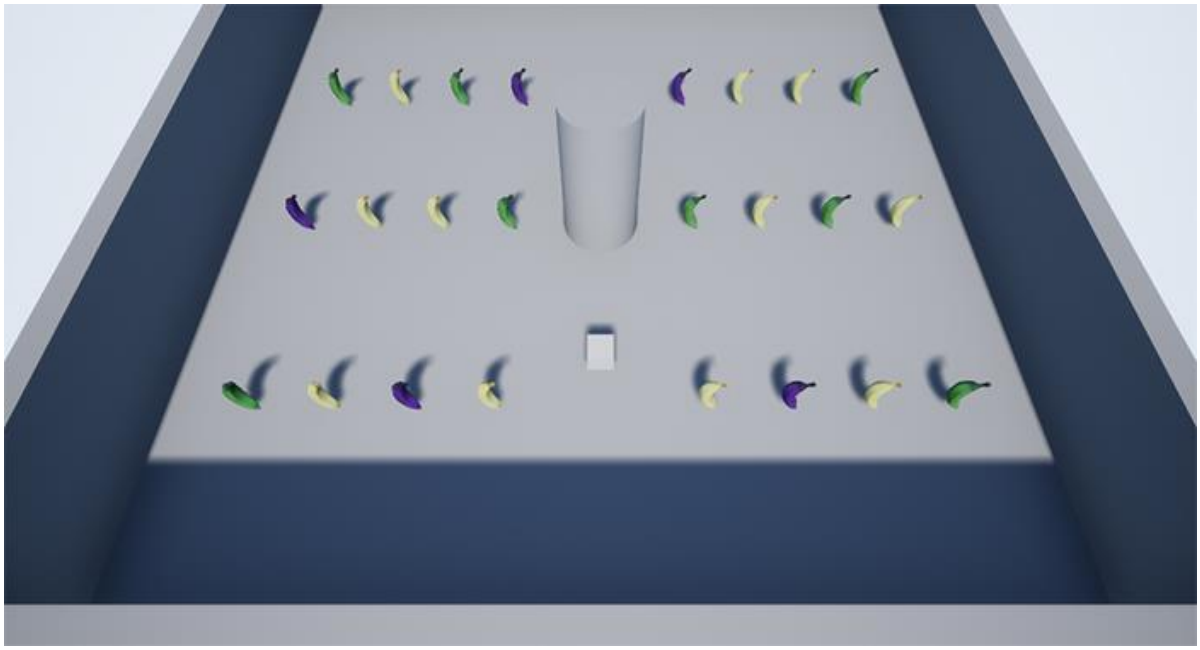
Во-первых, добавим нод *LinearInterpolate*. Это можно сделать быстро, удерживая клавишу *L* и *нажав левой клавишей мыши* на пустом пространстве в графе.

Затем создадим нод *Scalar Parameter* и назовём его *ColorAlpha*. Соединим ноды следующим образом (заметьте, что белый теперь находится сверху):



Подведём итог: нод *LinearInterpolate* будет выводить значение входного значения *A*. Так происходит потому, что начальное значение *alpha* равно *0*. При приближении *alpha* к *1*, выходное значение будет приближаться ко входному значению *B*.

Материал уже готов. Нам ещё предстоит многое сделать, но чтобы посмотреть, что у нас получилось, нажмите на *Apply* и закройте редактор материалов. Если нажать *Play*, то вы увидите, что теперь куб не красный, а белый.

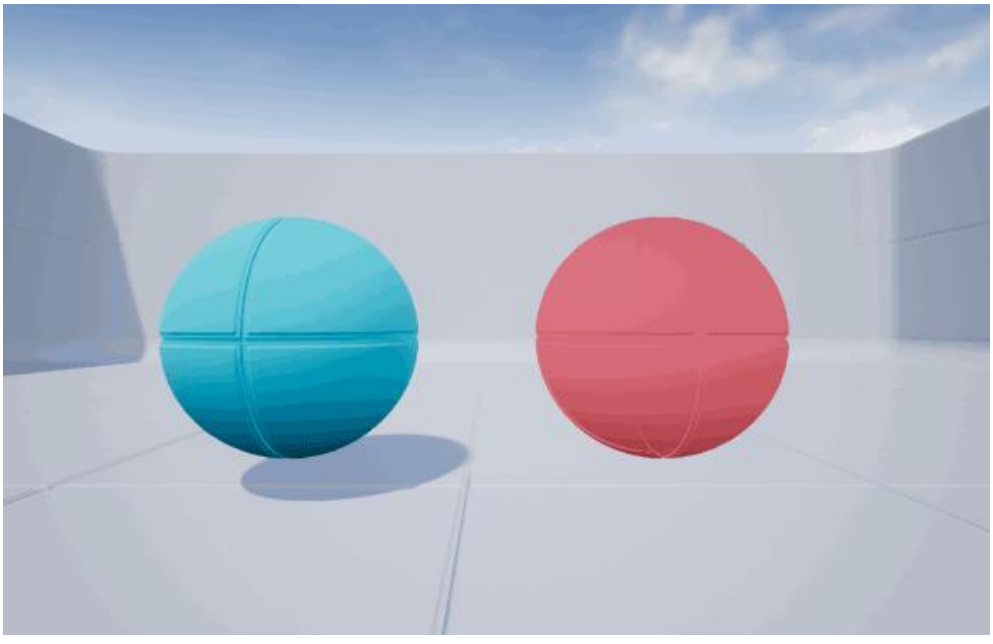


Чтобы куб менял цвет, необходимо изменять параметр *ColorAlpha*. Однако существует одна проблема. Во время выполнения игры мы не можем изменять параметры экземпляра материала. Решением будет использование *динамических экземпляров материалов*.

О динамических экземплярах материалов

В отличие от обычного экземпляра, динамический экземпляр материала можно изменять во время игрового процесса. Это можно делать с помощью Blueprints или C++.

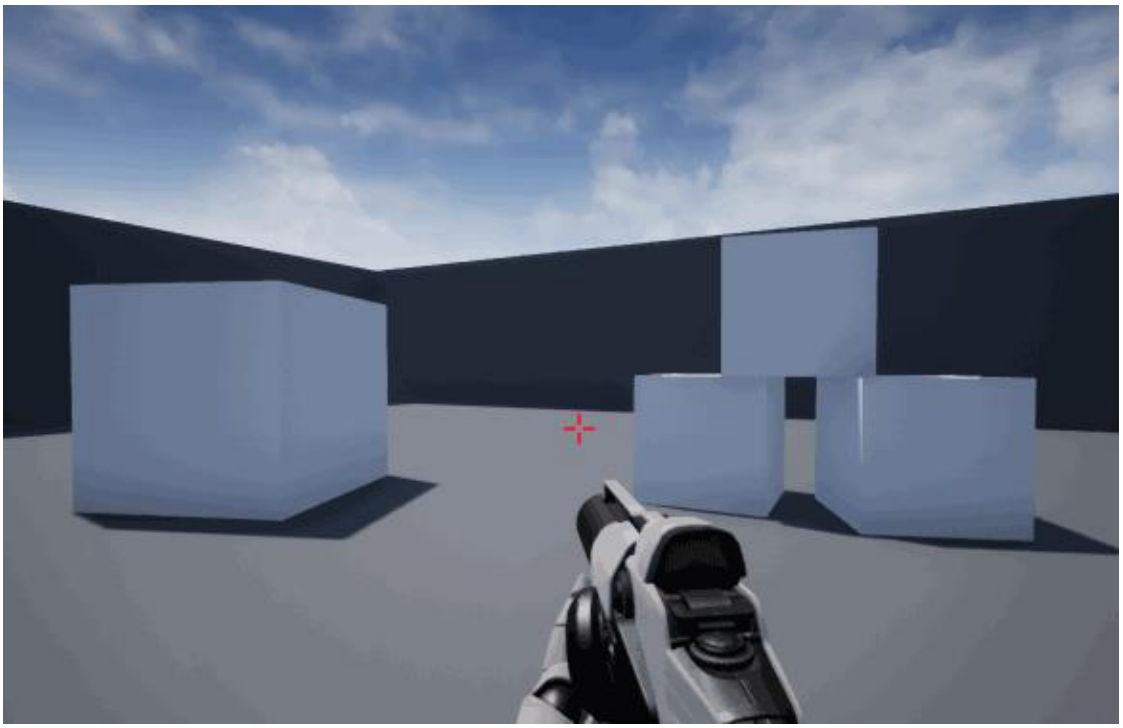
Динамические материалы можно использовать различными способами, например, изменять прозрачность объекта, чтобы делать его невидимым. Или менять блеск объекта, когда он становится мокрее.



Ещё один хороший аспект динамических экземпляров материалов заключается в том, что

их можно изменять индивидуально.

Ниже представлен пример обновления отдельных экземпляров для наложения маски на области объекта.

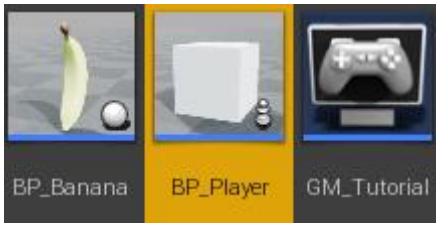


Давайте начнём с создания динамического экземпляра материала.

Создание динамического экземпляра материала

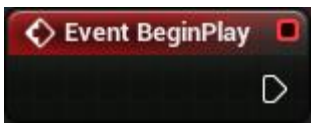
Динамические экземпляры материалов можно создавать только во время процесса игры. Для этого можно использовать Blueprints (или C++).

В Content Browser перейдите в папку *Blueprints* и *дважды щёлкните* на *BP_Player*, чтобы открыть его.

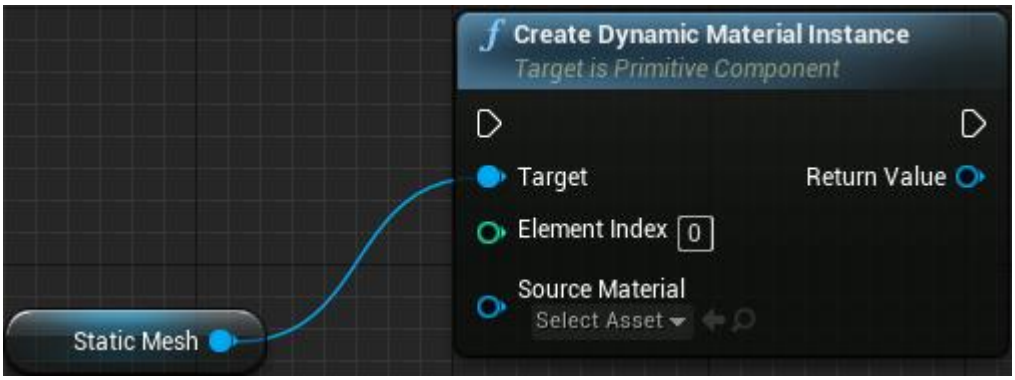


Первое, что мы сделаем — это создадим новый динамический экземпляр материала и затем применим его к мешу куба. Логично будет делать это, когда Unreal спаунит актора, а эту задачу выполняет нод *Event BeginPlay*.

Перейдите в Event Graph и найдите нод *Event BeginPlay*.

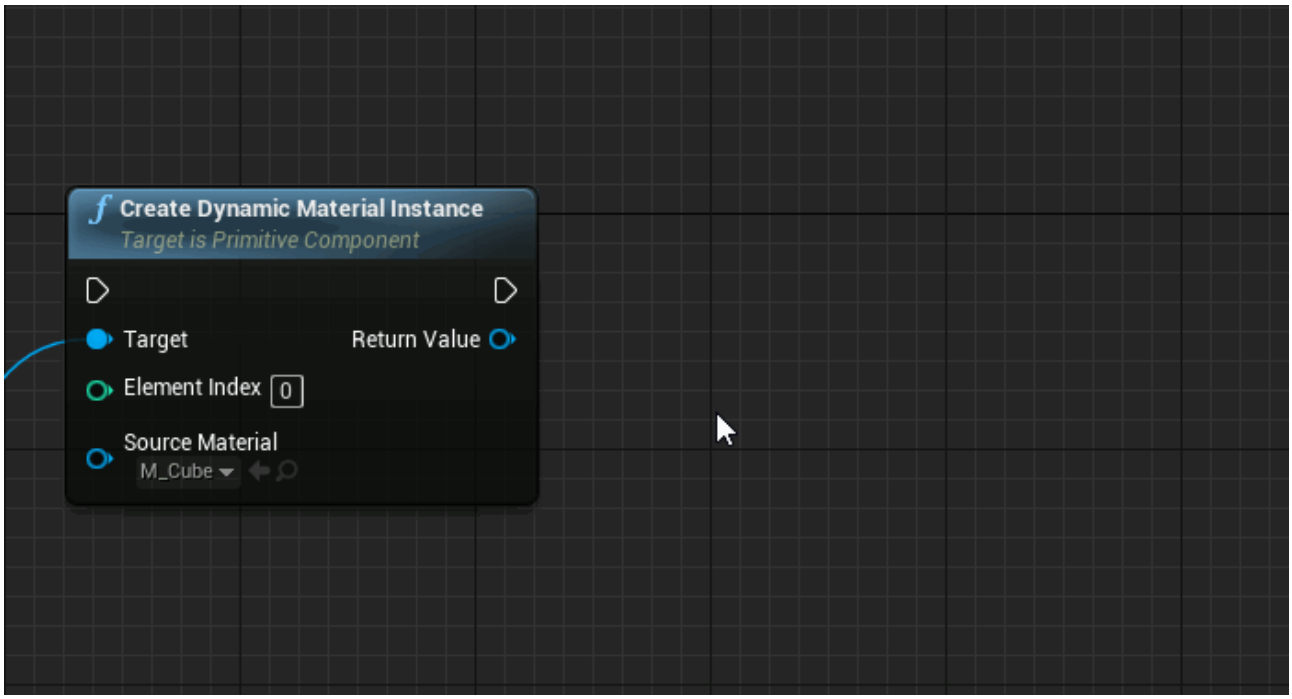


Теперь добавьте нод *Create Dynamic Material Instance (StaticMesh)*. Этот нод будет одновременно создавать и применять новый динамический экземпляр материала к мешу куба.

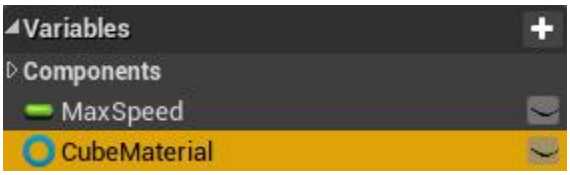


Теперь нам нужно указать, какой материал должен использовать куб. Нажмите на *раскрывающийся список* в *Source Material* и выберите *M_Cube*.

Чтобы удобнее ссылаться на материал, лучше всего хранить его в переменной. Проще всего это сделать *нажав правой клавишей мыши* на контакт *Return Value* нода *Create Dynamic Material Instance*. Затем выберите *Promote to Variable*.



Если заглянуть во вкладку My Blueprint, то вы заметите, что там появилась новая переменная. Переименуйте её в *CubeMaterial*. Это можно быстро сделать, нажав клавишу *F2*.



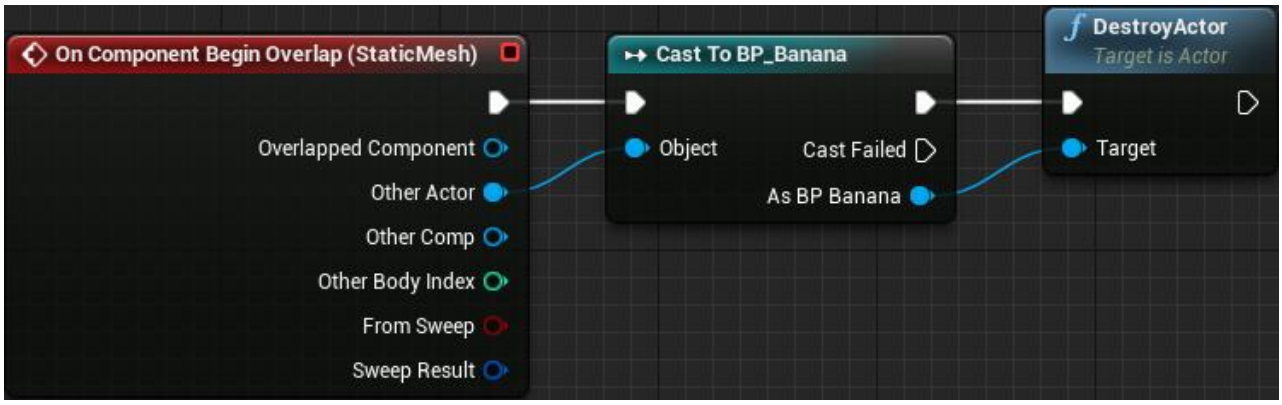
Наконец, соедините нод *Event BeginPlay* с нодом *Create Dynamic Material Instance*.

Подведём итог: когда Unreal спаунит *BP_Player*, он создаёт новый динамический экземпляр материала и применяет его к компоненту *StaticMesh*. Затем он сохраняет материал в переменную под названием *CubeMaterial*.

Следующим шагом будет создание счётчика для отслеживания количества собранных бананов.

Создание счётчика бананов

Если немного переместиться от нода *Event BeginPlay*, то можно увидеть следующую схему. Здесь мы будем обновлять счётчик бананов и материал.



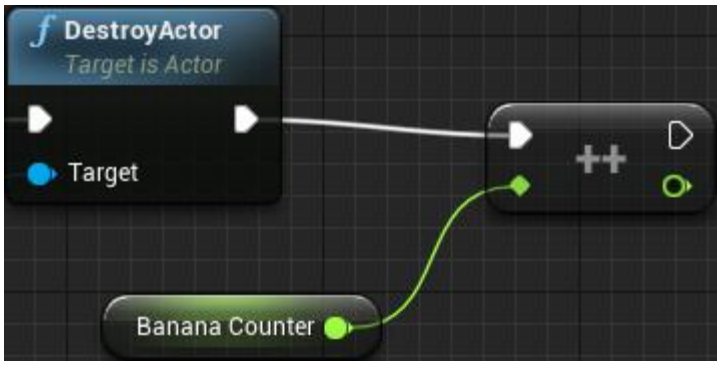
Нод *On Component Begin Overlap* выполняется, когда куб перекрывает другого актора. Затем нод *Cast to BP_Banana* проверяет, является ли перекрываемый актор бананом. Если актор — это банан, то нод *DestroyActor* уничтожает его и он исчезает из игры.

Первое, что нужно сделать — создать переменную для хранения количества собранных бананов. Позже мы будем увеличивать значение на единицу каждый раз, когда куб

касается банана.

Создайте новую переменную *Float* и назовите её *BananaCounter*. *Перетащите* переменную BananaCounter в Event Graph и выберите *Get*.

Далее присоедините нод *DestroyActor* к ноду *IncrementFloat*.

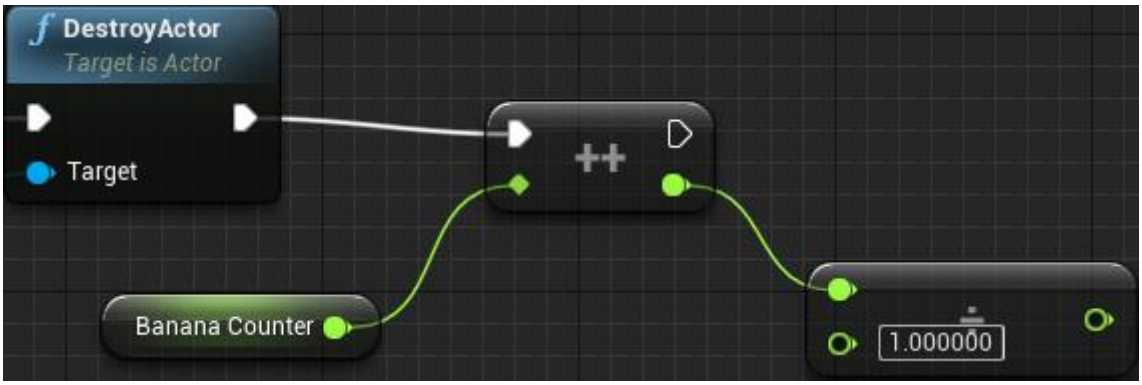


Теперь, когда игрок будет поднимать банан, переменная *BananaCounter* будет увеличиваться на единицу.

Если бы мы использовали *BananaCounter* в качестве alpha прямо сейчас, то получили бы неожиданные результаты, потому что нод *LinearInterpolation* ожидает значения в интервале от 0 до 1. Можно использовать нормализацию для преобразования счётчика в интервал от 0 до 1.

Для нормализации мы просто разделим *BananaCounter* на максимальное значение. Это значение равно количеству бананов, которое должен собрать куб, чтобы полностью стать красным.

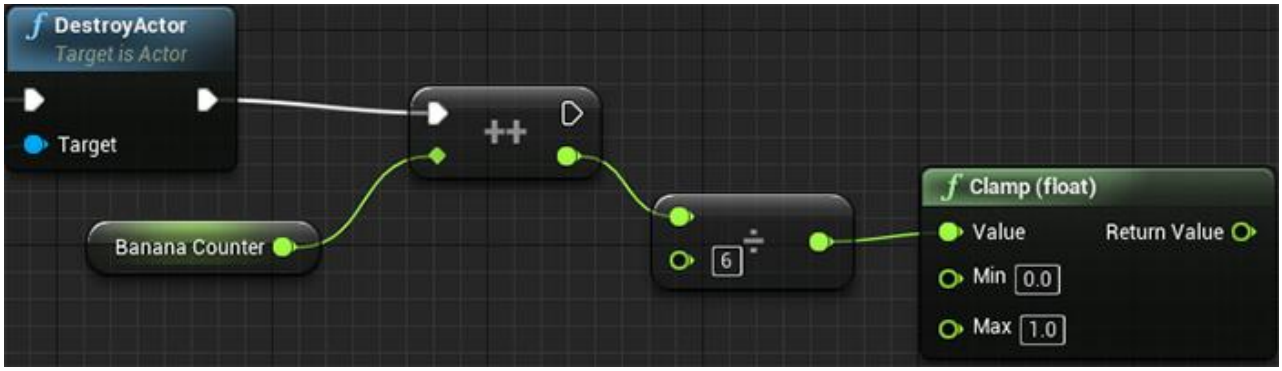
Добавим нод *float / float* и соединим его *верхний* контакт с оставшимся контактом нода *IncrementFloat*.



Зададим в качестве *нижнего* входного значения нода *float / float* значение *6*. Это значит, что куб станет полностью красным, когда игрок соберёт шесть бананов.

Но есть небольшая проблема. Когда игрок соберёт больше шести бананов, то alpha станет больше единицы. Чтобы исправить это, воспользуемся нодом *Clamp (float)* для ограничения alpha интервалом от 0 до 1.

Добавьте нод *Clamp (float)* и соедините контакт *Value* с *правым* контактом нода *float / float*.

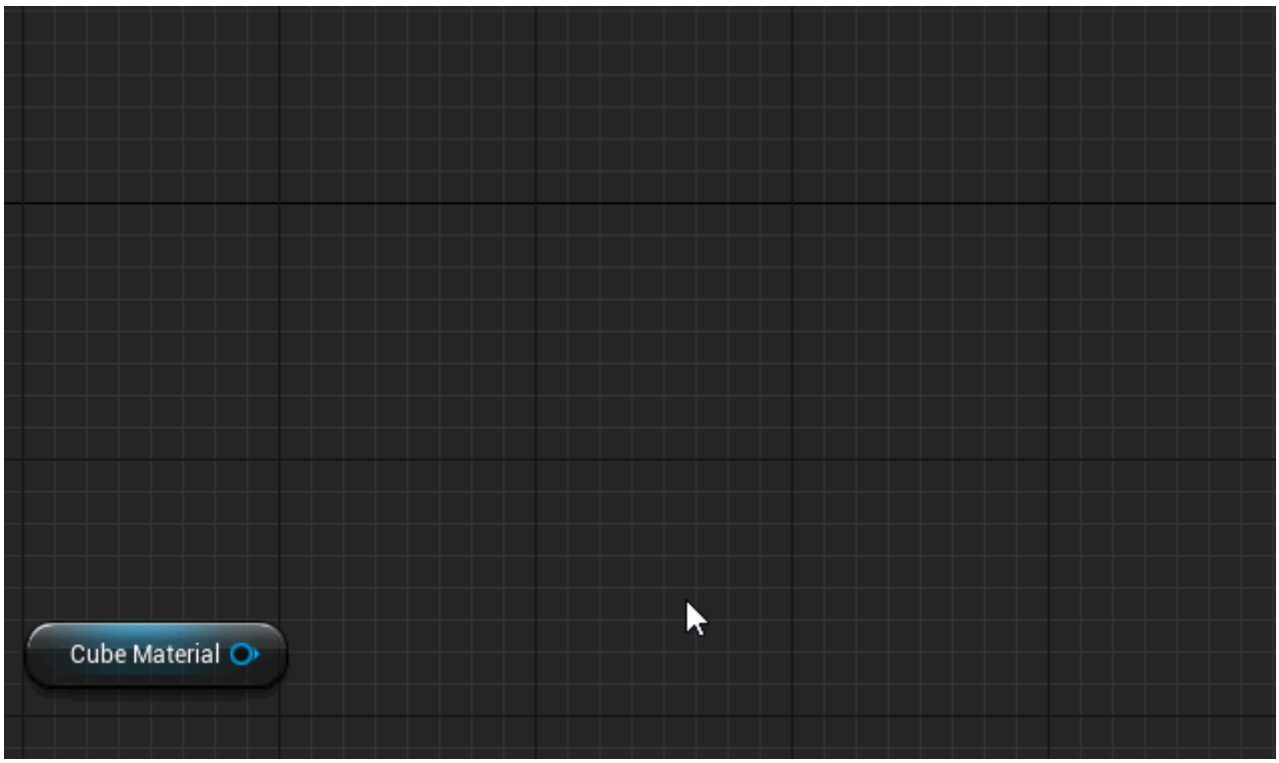


Теперь, когда у нас есть alpha, настало время для передачи её значения материалу.

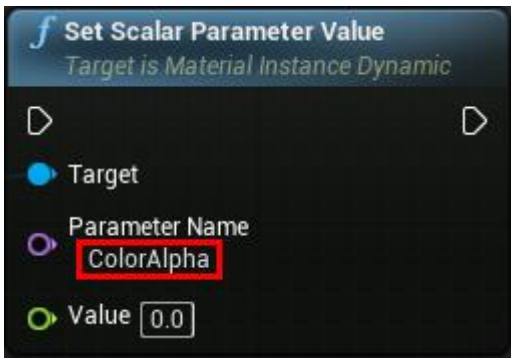
Обновление материала

Перетащите переменную *CubeMaterial* в Event Graph и выберите *Get*.

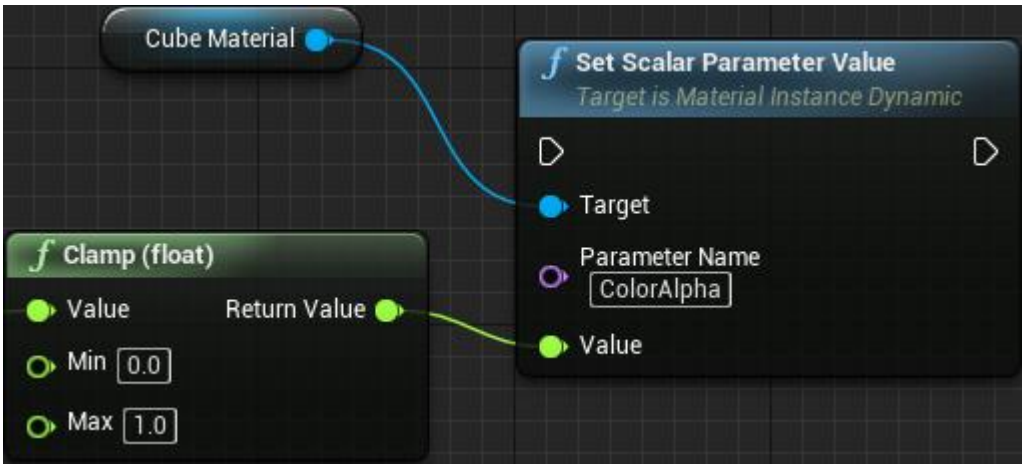
Затем *перетащите* контакт переменной *CubeMaterial* на пустое пространство и отпустите *левую клавишу мыши*. При этом появится список нодов, которые могут использовать переменную этого типа. Любой выбранный нод будет автоматически соединён с переменной. Добавьте нод *Set Scalar Parameter Value*. Этот нод будет задавать указанному параметру передаваемое значение.



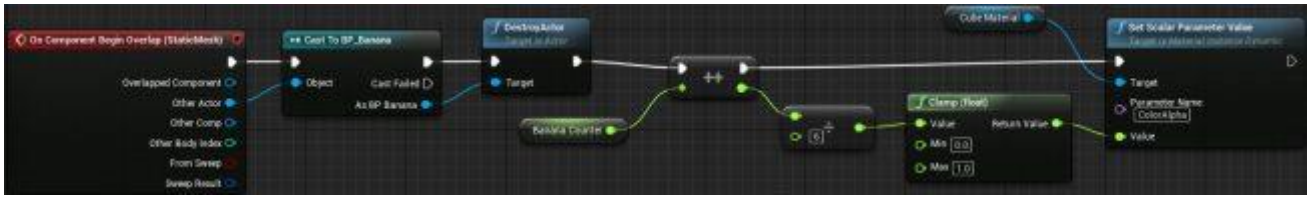
Теперь нужно указать обновляемый параметр. Выберите в поле *Parameter Name* значение *ColorAlpha*. Это параметр, который мы создали в материале куба.



Соедините результат нода *Clamp (float)* с контактом Value нода *Set Scalar Parameter Value*.



Наконец, соедините нод *IncrementFloat* с нодом *Set Scalar Parameter Value*.



Порядок выполнения будет следующим:

1. *On Component Begin Overlap (StaticMesh):* выполняется, когда меш куба пересекается с другим актором
2. *Cast to BP_Banana:* проверяет, является ли пересекаемый актор бананом
3. *DestroyActor:* если пересекаемый актор является бананом, то уничтожает его
4. *IncrementFloat:* увеличивает *BananaCounter* на единицу
5. *float / float:* делит счётчик на заданное число, чтобы нормализировать его
6. *Clamp (float):* ограничивает результат деления, чтобы не могло получиться значение больше 1
7. *Set Scalar Parameter Value:* задаёт параметру *ColorAlpha* материала куба передаваемое значение. В этом случае значение является нормализированной и ограниченной в интервале версией *BananaCounter*

Настало время всё проверить! Нажмите на *Compile* и закройте Blueprint editor.

Нажмите на *Play* и начните собирать бананы. Куб сначала будет белым и постепенно станет краснеть при собирании бананов. Когда вы соберёте шесть бананов, он станет полностью красным.