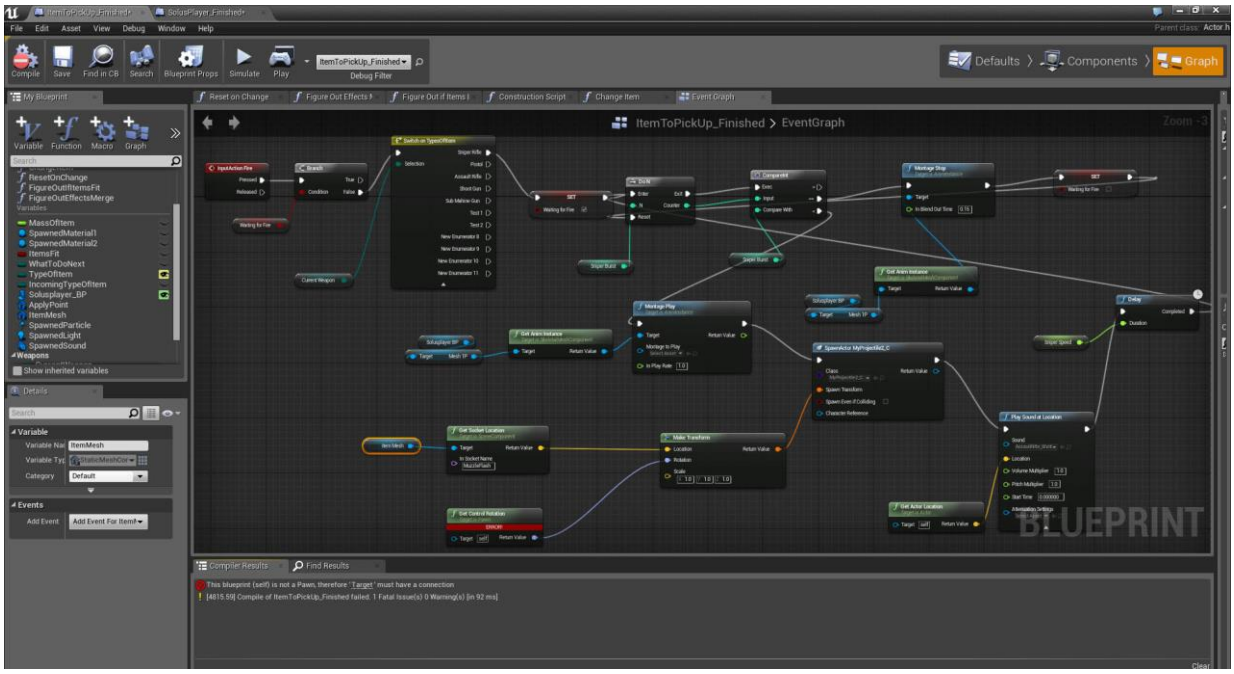
Туториал по Unreal Engine. Часть 2: Blueprints



Blueprints — это система визуального скриптинга Unreal Engine 4. Она является быстрым способом создания прототипов игр. Вместо построчного написания кода всё можно делать визуально: перетаскивать ноды (узлы), задавать их свойства в интерфейсе и соединять их «провода».

Кроме быстрого прототипирования, Blueprints также упрощают создание скриптов для непрограммистов.

В этой части туториала мы будем использовать Blueprints для следующих операций:

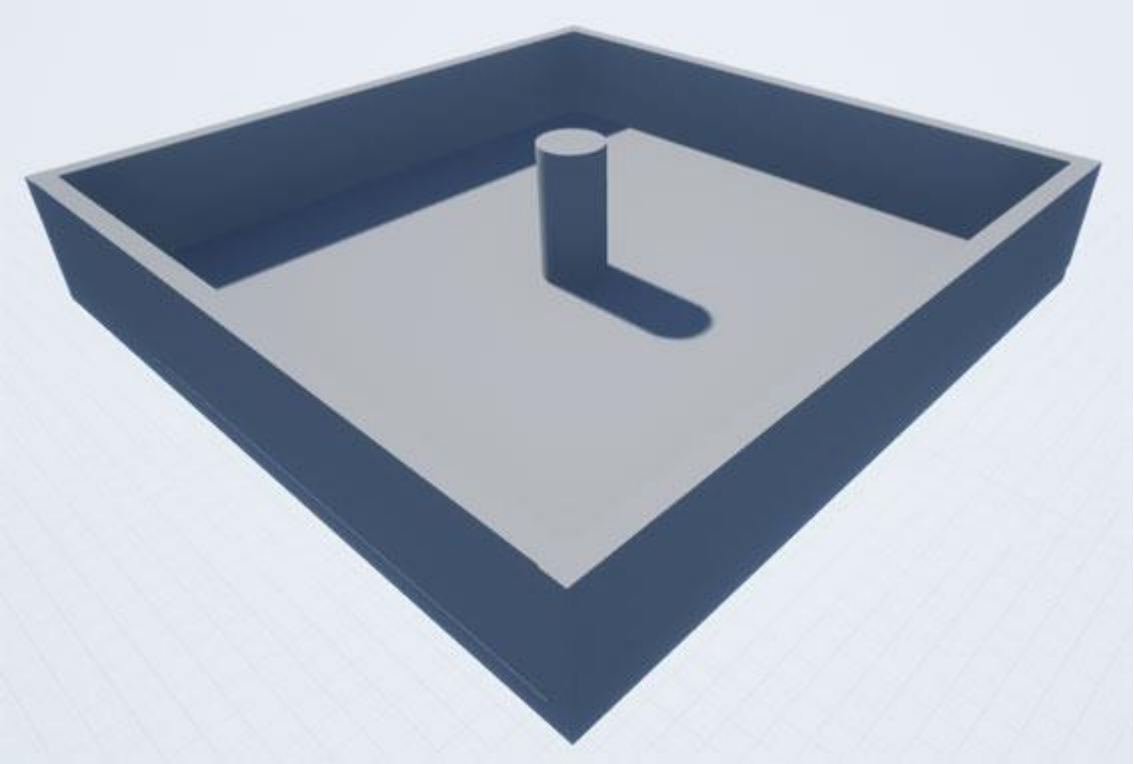
- Настройка камеры с видом сверху
- Создание управляемого игроком актора с простыми движениями
- Настройка ввода игрока
- Создание элемента, исчезающего при контакте с игроком

Приступаем к работе

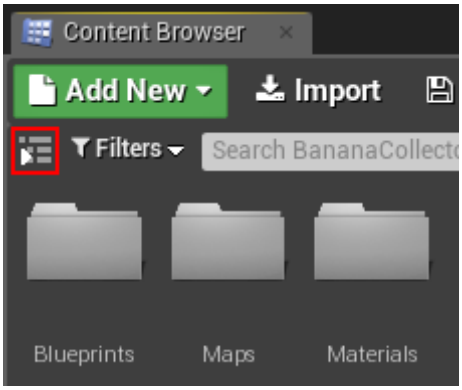
Скачайте https://koenig-media.raywenderlich.com/uploads/2017/04/BananaCollectorStarter.zip и распакуйте его. Чтобы открыть проект, перейдите в папку проекта и откройте *BananaCollector.uproject*.

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.

На рисунке ниже показана сцена. Именно в ней игрок будет перемещаться и собирать предметы.



Для простоты навигации я разбил файлы проекта на папки, как показано на рисунке:

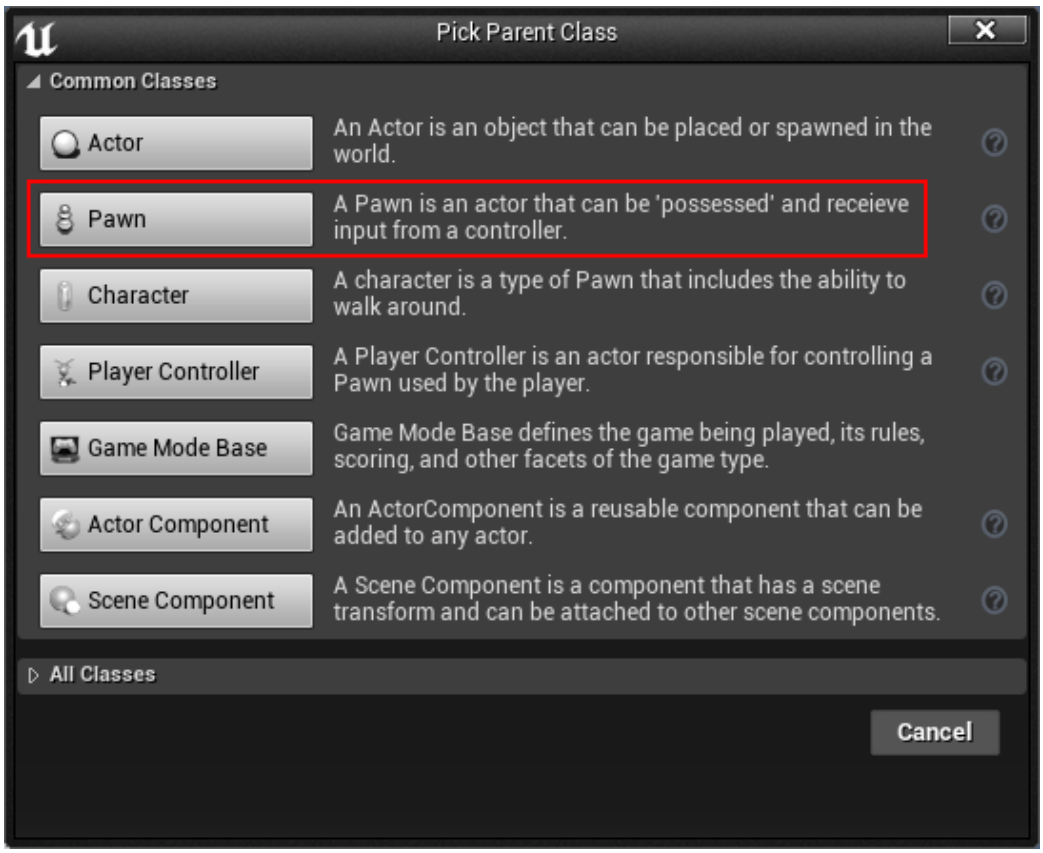


Выделенную красным кнопку можно использовать, чтобы показать или скрыть панель исходников.

Создание игрока

В Content Browser перейдите к папке *Blueprints*. Нажмите на кнопку *Add New* и выберите *Blueprint Class*.

Мы хотим, чтобы актор получал вводимую игроком информацию, поэтому нам подходит класс *Pawn*. Выберите во всплывающем окне *Pawn* и назовите его *BP_Player*.



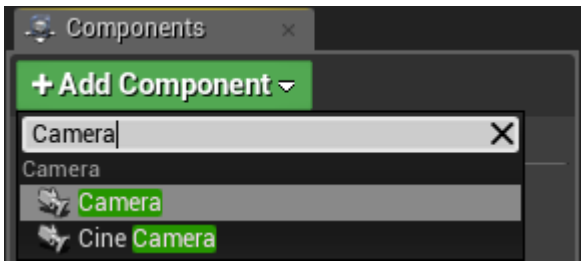
Примечание: класс *Character* тоже подойдёт. В нём даже по умолчанию есть компонент перемещения. Однако мы будем реализовывать собственную систему движения, поэтому класса *Pawn* нам достаточно.

Прикрепление камеры

Камера — это способ игрока смотреть на мир. Мы создадим камеру, смотрящую на игрока сверху вниз.

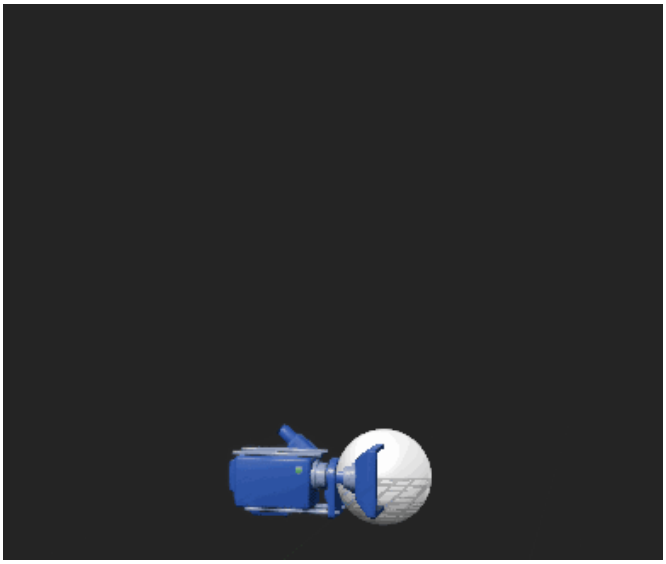
В Content Browser *дважды нажмите* на *BP_Player*, чтобы открыть его в Blueprint editor.

Для создания камеры перейдите на панель Components. Нажмите на *Add Component* и выберите *Camera*.



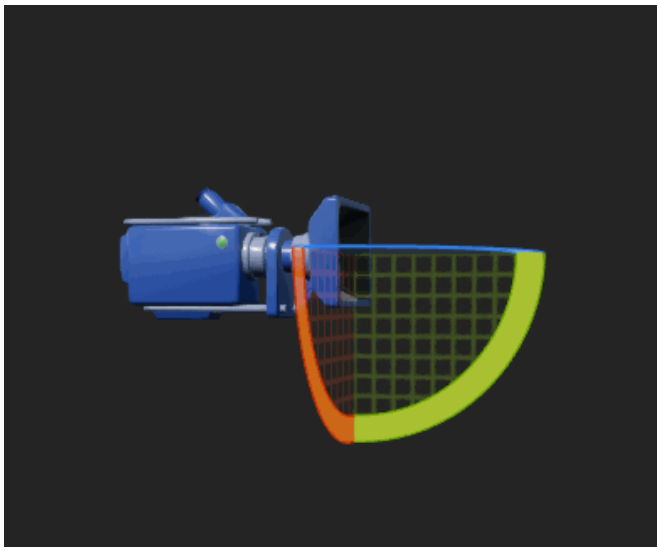
Чтобы камера смотрела сверху вниз, нужно расположить её над игроком. Выбрав компонент камеры, перейдите во вкладку Viewport.

Активируйте манипулятор перемещения, нажав клавишу *W*, а затем переместите камеру в *(-1100, 0, 2000)*. Или же можно ввести координаты в поля *Location*. Она находится в разделе *Transform* панели Details.



Если вы потеряли камеру из виду, нажмите клавишу *F*, чтобы сфокусироваться на ней.

Затем активируйте манипулятор поворота, нажав клавишу *E*. Поверните камеру вниз на *-60* градусов по оси *Y*.

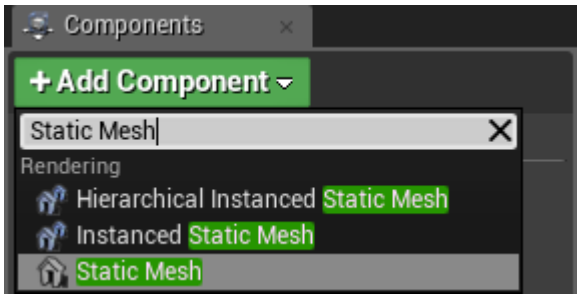


Отображаем игрока

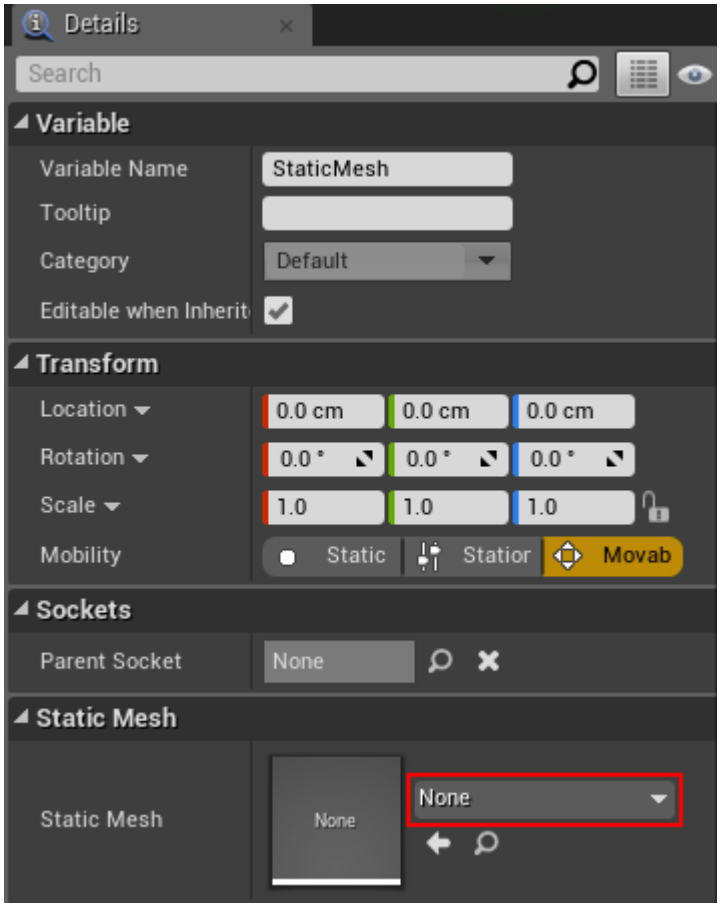
Мы обозначим персонажа игрока красным кубом, поэтому для его отображения нужно будет использовать компонент Static Mesh.

Во-первых, снимите выделение с компонента *Camera*, нажав *левой клавишей мыша* на пустом пространстве в панели Components. Если этого не сделать, то следующий добавленный компонент будет дочерним по отношению к камере.

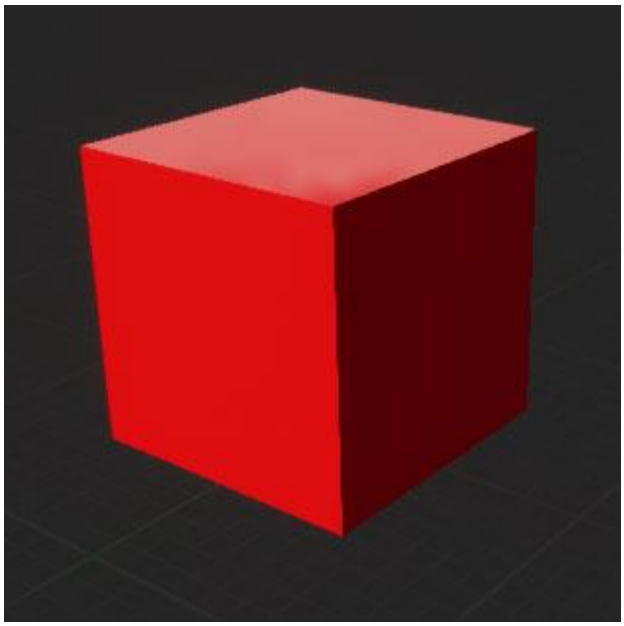
Нажмите на *Add Component* и выберите *Static Mesh*.



Чтобы отобразить красный куб, выберите компонент *Static Mesh*, а затем перейдите во вкладку Details. Нажмите на *раскрывающийся список*, находящийся справа от *Static Mesh* и выберите *SM_Cube*.



Вы должны увидеть следующее (можно нажать *F* внутри Viewport, чтобы сфокусироваться на кубе, если вы его не видите):



Теперь настало время заспаунить актора Pawn игрока. Нажмите на *Compile* и вернитесь к основному редактору.

Спаун игрока

Чтобы игрок мог управлять Pawn, нужно указать две вещи:

1. Класс Pawn, которым будет управлять игрок
2. Место спауна Pawn

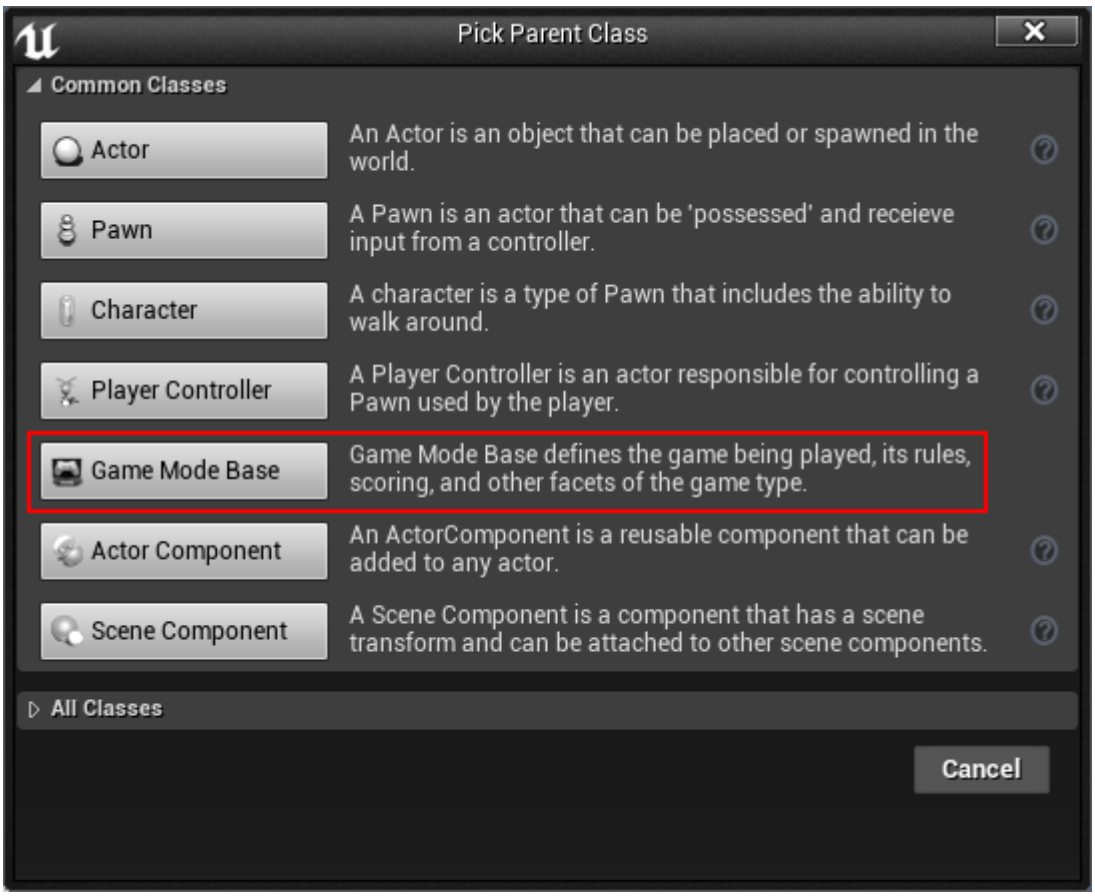
Первую задачу можно выполнить, создав новый класс *Game Mode*.

Создание Game Mode

Класс Game Mode (игровой режим) — это класс, управляющий тем, как игрок входит в игру. Например, в многопользовательской игре Game Mode используется для задания спауна каждого игрока. Что более важно, Game Mode определяет. какой Pawn будет использовать игрок.

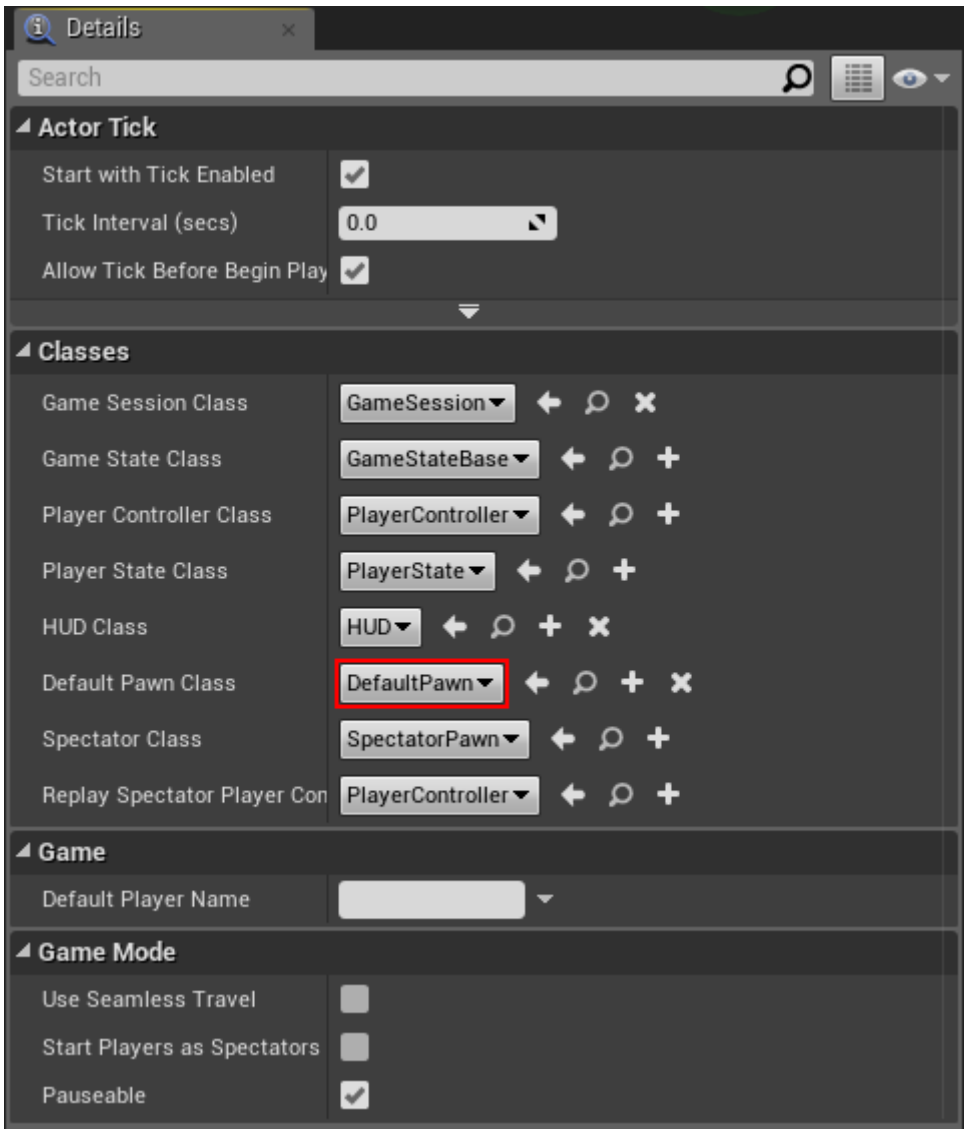
Перейдите к Content Browser и зайдите в папку *Blueprints*. Нажмите на кнопку *Add New* и выберите *Blueprint Class*.

Во всплывающем меню выберите *Game Mode Base* и назовите его *GM_Tutorial*.



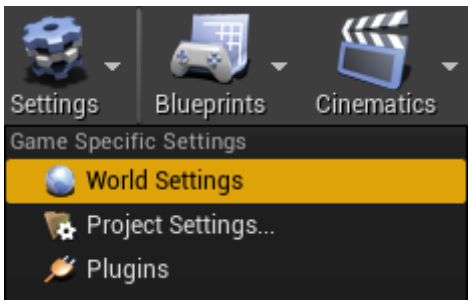
Теперь нужно указать, какой класс Pawn будет использоваться по умолчанию. *Дважды нажмите* на *GM_Tutorial*, чтобы открыть его.

Перейдите на панель Details и загляните в раздел *Classes*. Нажмите на *раскрывающийся список Default Pawn Class* и выберите *BP_Player*.

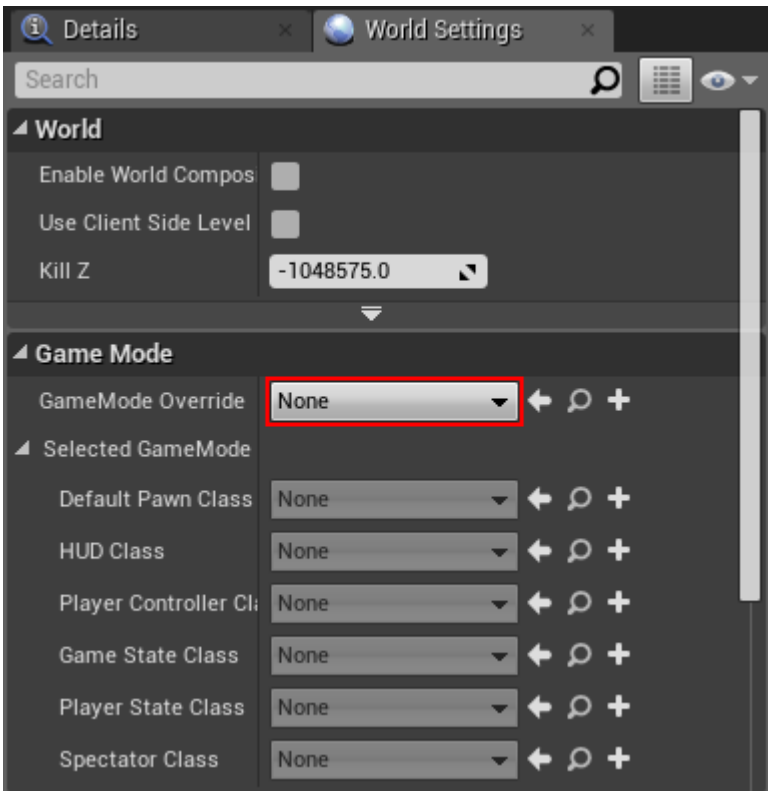


Чтобы использовать новый Game Mode, нужно сообщить уровню, какой Game Mode он должен использовать. Это можно указать в *World Settings*. Нажмите на *Compile* и закройте Blueprint editor.

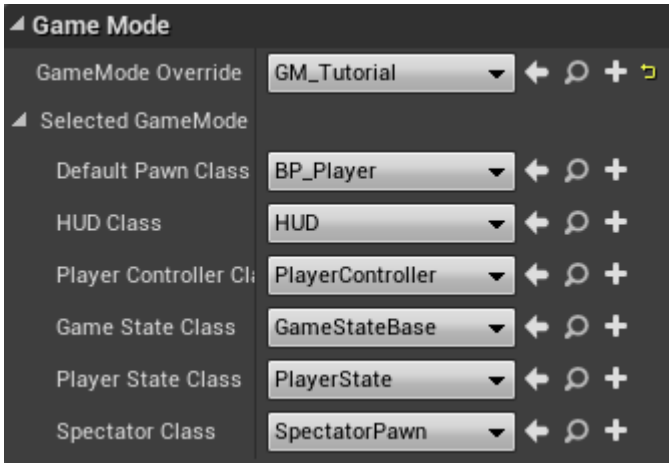
Каждый уровень имеет собственные параметры. Получить доступ к этим параметрам можно, выбрав *Window\World Settings*. Или же можно зайти в Toolbar и выбрать *Settings\World Settings*.



Рядом со вкладкой Details откроется новая вкладка World Settings. В ней нажмите на *раскрывающийся список GameMode Override* и выберите *GM_Tutorial*.



Теперь вы увидите, что классы сменились на те, которые выбраны в *GM_Tutorial*.

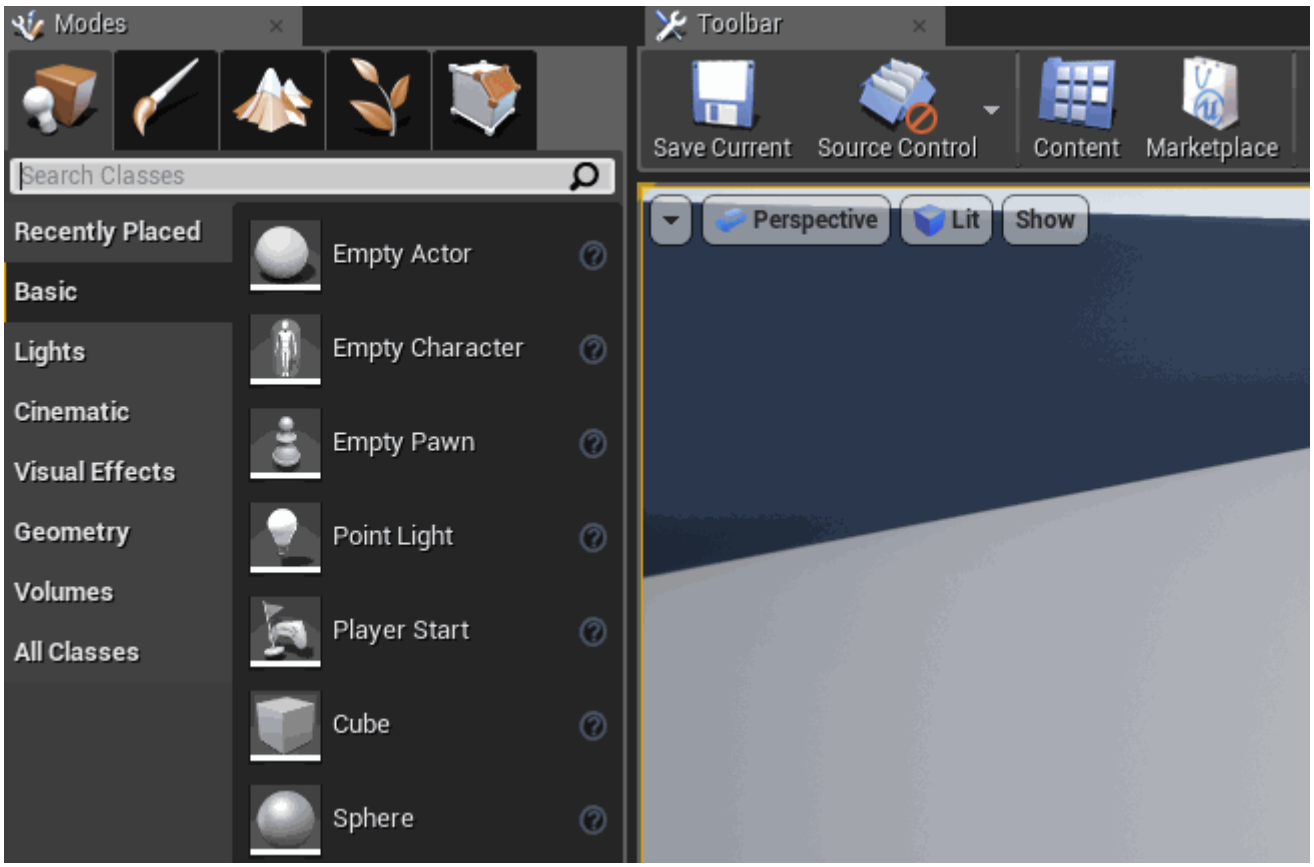


Наконец, нам нужно задать точку спауна игрока. Это реализуется размещением на уровне актора *Player Start*.

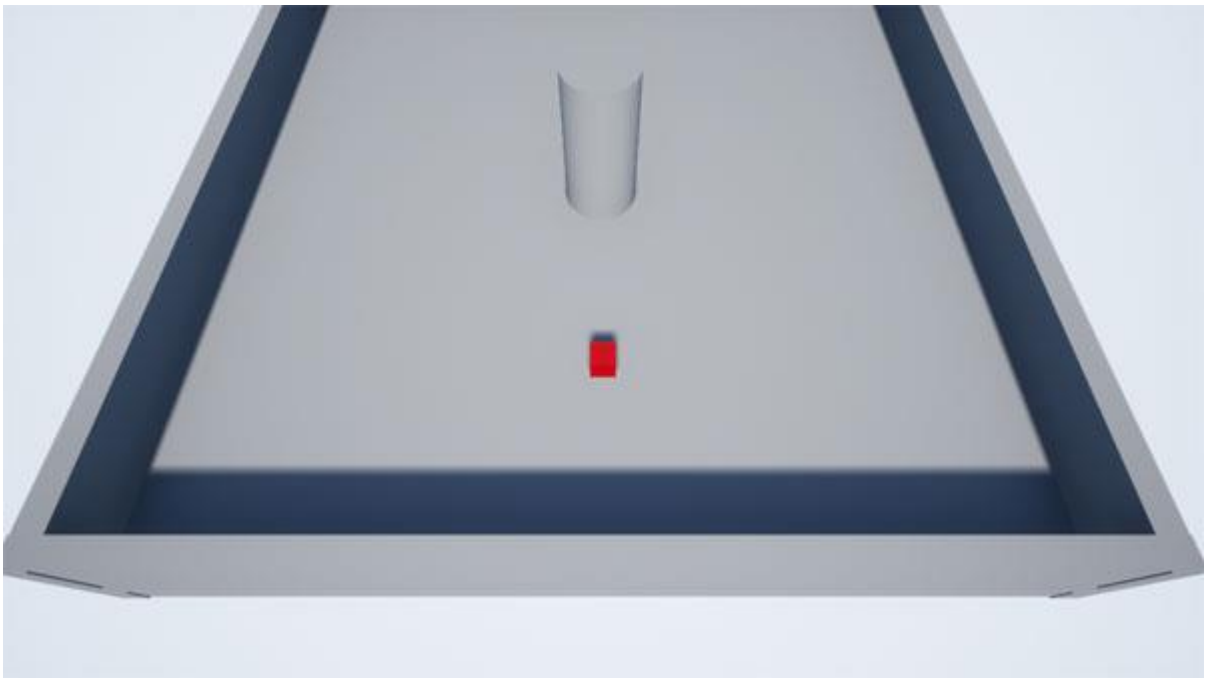
Размещение Player Start

В процессе спауна игрока Game Mode ищет актор Player Start. Если Game Mode находит его, то предпринимает попытку заспаунить игрока там.

Чтобы разместить Player Start, перейдите к панели Modes и найдите *Player Start*. *Нажмите левой клавишей* и перетащите *Player Start* из панели Modes во Viewport. Отпустите *левую клавишу мыши*, чтобы разместить его.



Можете разместить его где угодно. Когда закончите, перейдите в Toolbar и нажмите *Play*. Вы будете заспаунены в точке расположения Player Start.



Чтобы выйти из игры, нажмите кнопку *Stop* в Toolbar или нажмите клавишу *Esc*. Если вы не видите курсор, нажмите *Shift+F1*.

Это не похоже на игру, если мы не можем двигаться. Наша следующая задача — настроить параметры ввода.

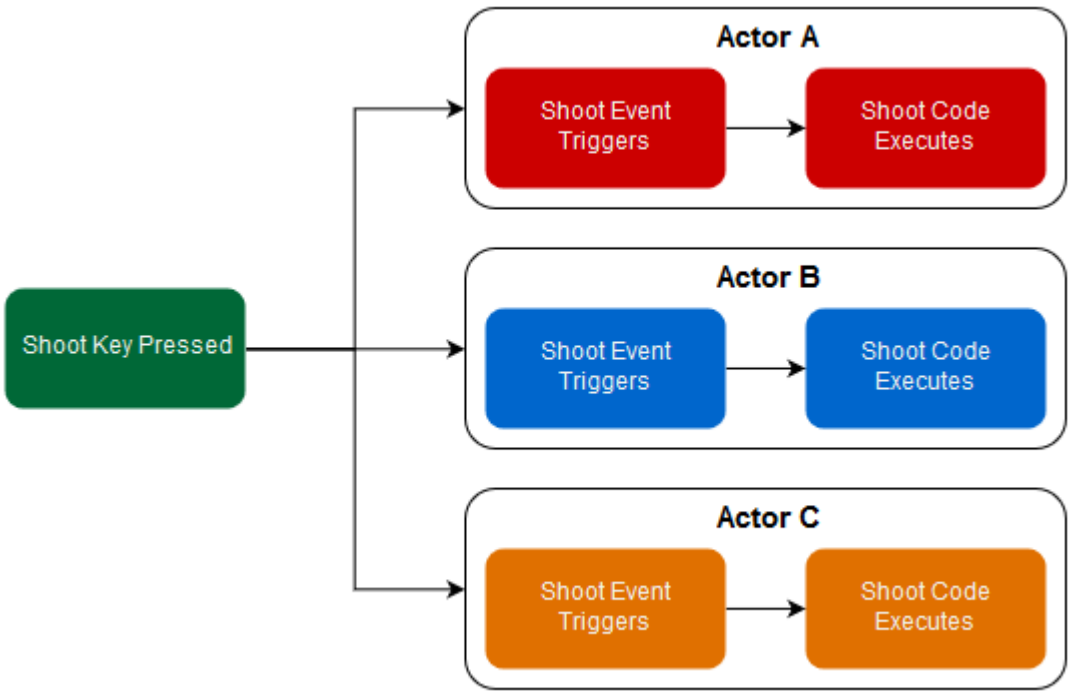
Настройка ввода

Назначение клавиши действию называется привязкой клавиши.

В Unreal можно настроить привязки клавиш, чтобы при их нажатии срабатывали *события*. События — это ноды, выполняющиеся при определённых действиях (в этом случае — при нажатии указанной клавиши). При срабатывании события выполняются все ноды, соединённые с событием.

Такой способ привязки клавиш удобен, потому что он означает, что нам не нужно жёстко задавать клавиши в коде.

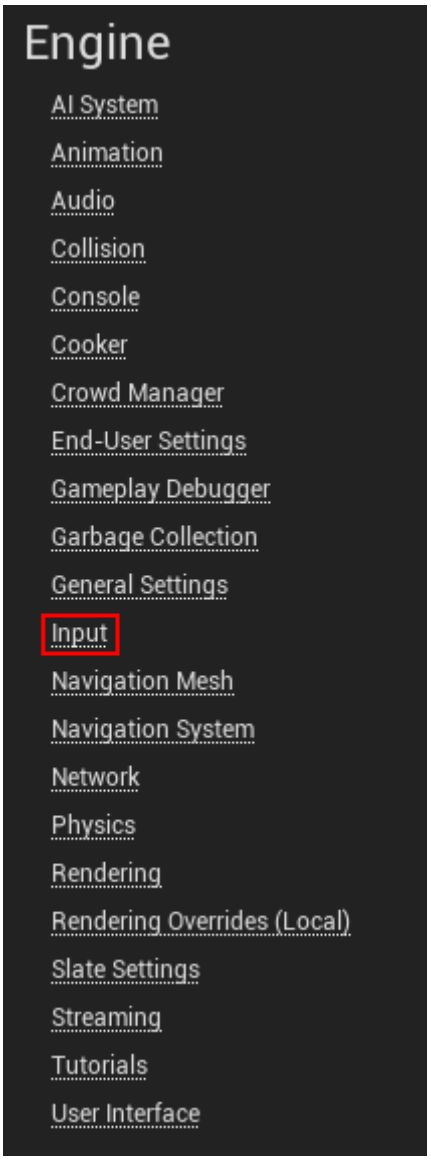
Например, можно привязать нажатие левой клавиши мыши и назвать её Shoot. Любой актор, умеющий стрелять, может использовать событие Shoot, чтобы знать, когда игрок нажимает на левую клавишу мыши. Если вы хотите изменить клавишу, то это можно сделать в параметрах ввода.



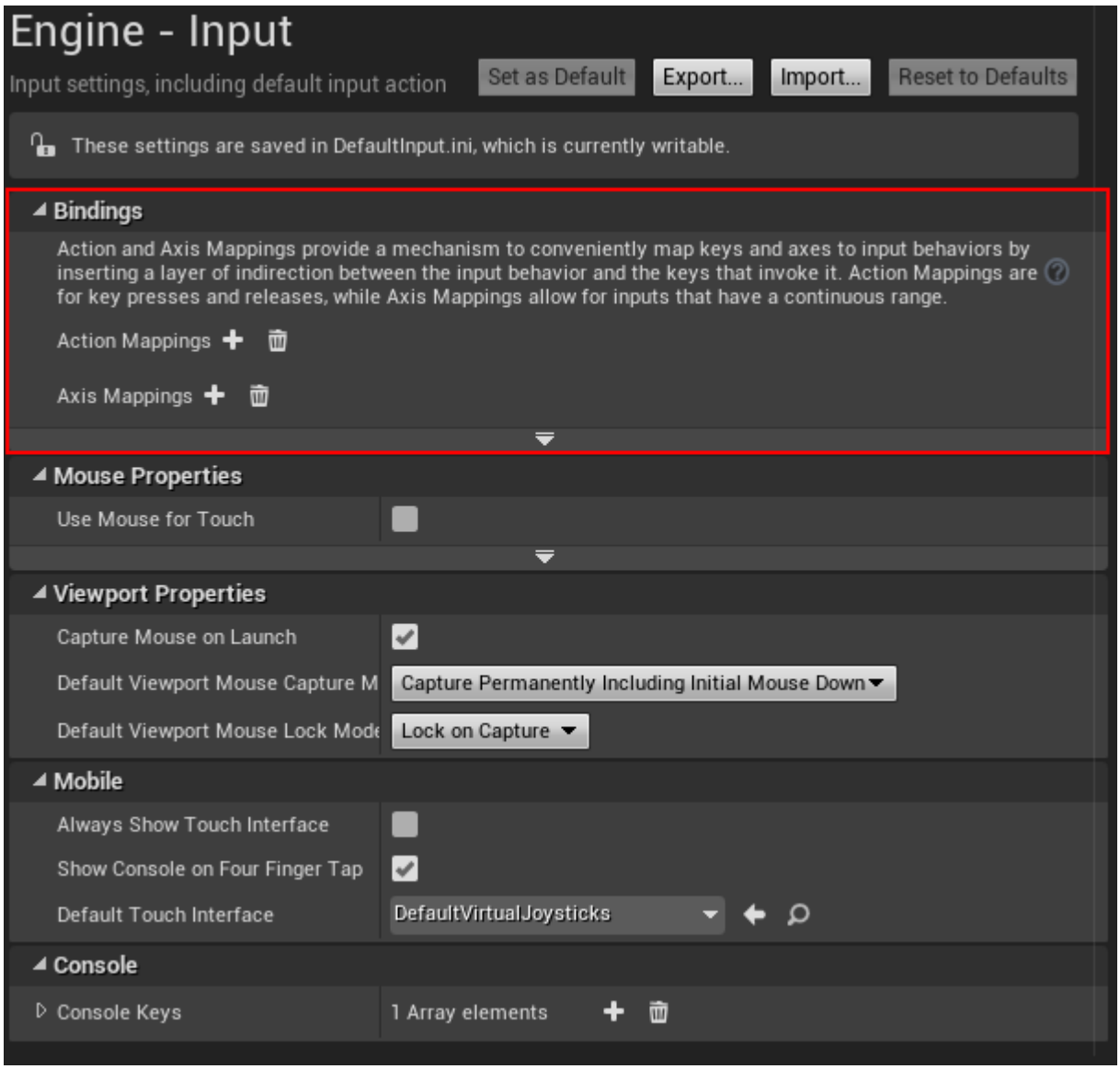
Если мы будем задавать клавиши жёстко, то нам придётся заходить в каждый актор и менять клавиши отдельно.

Привязка осей и действий

Чтобы перейти к параметрам ввода, зайдите в *Edit\Project Settings*. В разделе *Engine* выберите слева *Input*.



В разделе *Bindings* выполняется настройка ввода.



Unreal предоставляет два способа создания привязок клавиш:

- *Привязка действий:* они могут находиться всего в двух состояниях — нажато и не нажато. События действий срабатывают только когда вы нажимаете или отпускаете клавишу. Используются для действий, не имеющих промежуточных состояний, например, для стрельбы из пушки.
- *Привязка осей:* оси передают на выход численное значение, называемое *значением оси* (подробнее об этом позже). События оси срабатывают каждый кадр. Обычно используются для действий, требующих управления стиками или мышью.

В этом туториале мы будем использовать привязки *осей*.

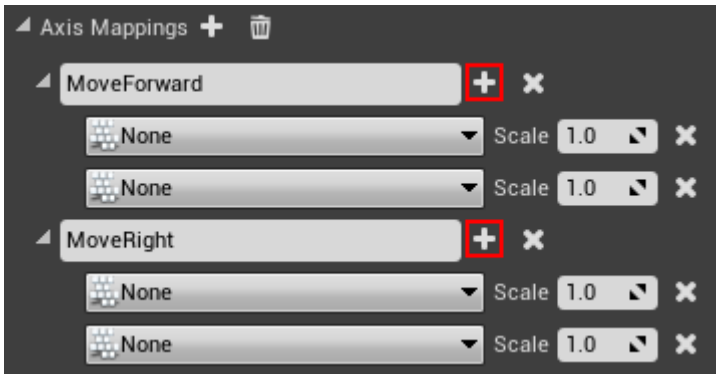
Создание привязок движения

Во-первых, мы создадим две *группы привязки осей*. Группы позволяют привязывать несколько клавиш к одному событию.

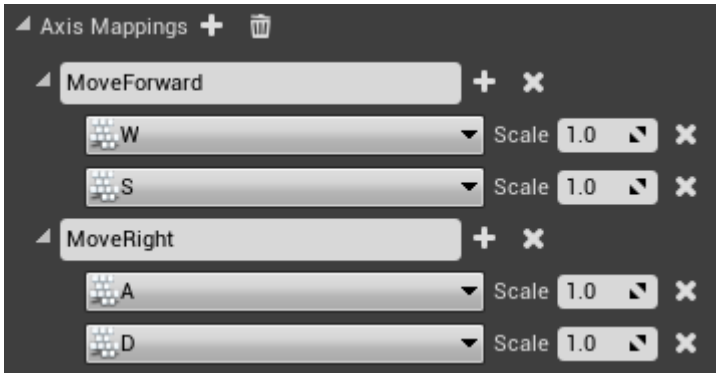
Для создания новой *группы привязки осей* нажмите на значок + справа от *Axis Mappings*. Создайте две группы и назовите их *MoveForward* и *MoveRight*.

MoveForward будет управлять движением вперёд и назад. *MoveRight* будет управлять движением влево и вправо.

Мы привяжем движение к четырём клавишам: *W*, *A*, *S* и *D*. Пока у нас есть только два слота для привязки клавиш. Добавим к каждой группе ещё одну *привязку осей*, нажав на значок + рядом с полем *имени группы*.



Чтобы привязать клавишу, нажмите на *раскрывающийся список* с перечислением клавиш. Привяжите клавиши *W* и *S* к *MoveForward*. Привяжите клавиши *A* и *D* к *MoveRight*.



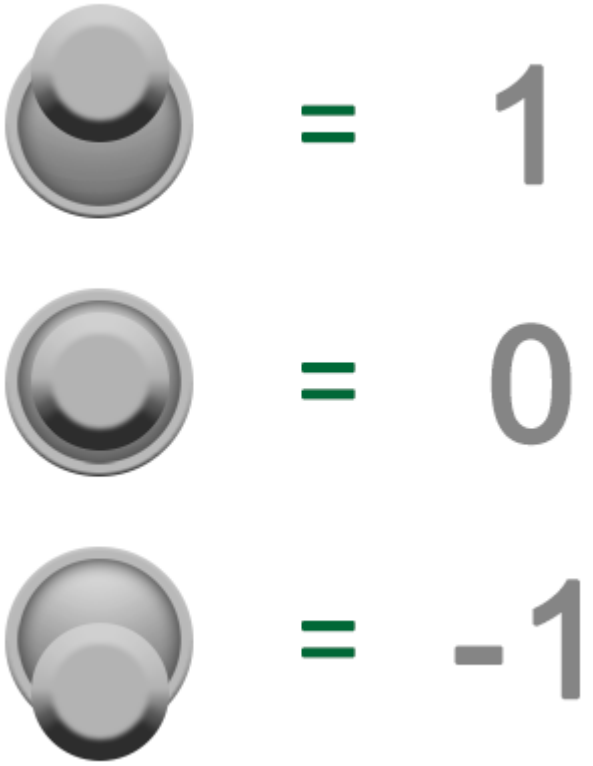
Теперь нужно задать значения в полях *Scale*.

Значение оси и масштаб ввода

Перед заданием полей Scale нам нужно больше узнать о том, как работать со *значениями*

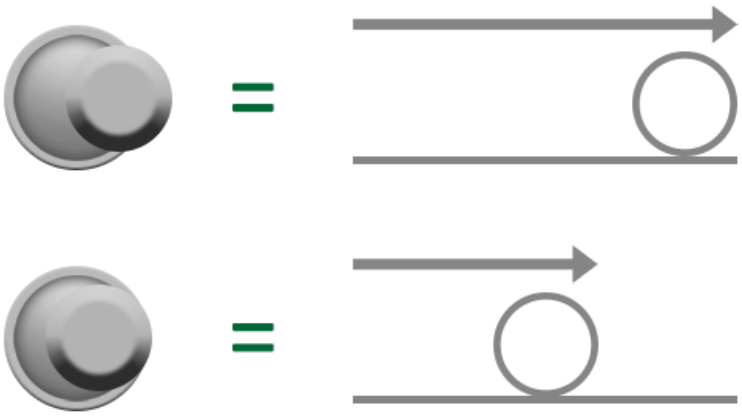
осей.

Значение оси — это численное значение, определяемое типом ввода и способом его использования. При нажатии на кнопки и клавиши на выход подаётся 1. Стики имеют выходные значения от -1 до 1, зависящие от направления и наклона стика.

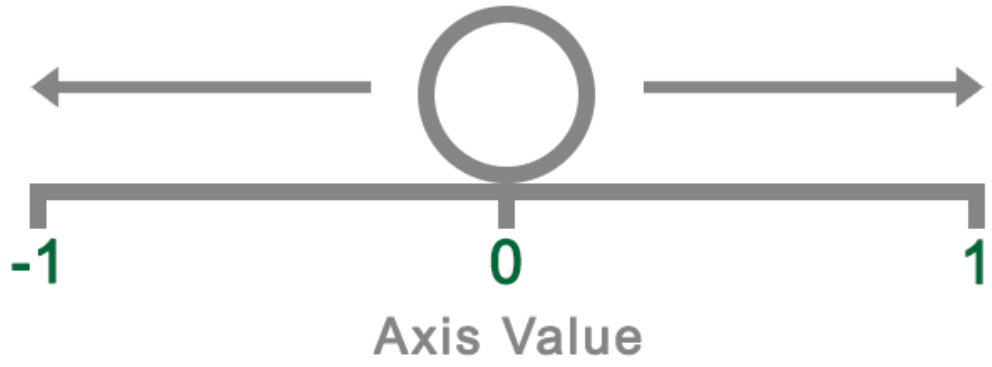


Мы можем использовать значение оси для управления скоростью Pawn. Например, если мы нажмём стик до упора, то значение оси будет 1. Если нажать наполовину, то значение будет 0.5.

Умножая значение оси на переменную скорости, мы можем регулировать с помощью стика скорость движения.



Также значение оси можно использовать для задания направления вдоль оси. Если умножить скорость Pawn на положительное значение оси, то мы получим положительное смещение. При использовании отрицательного значения оси получим отрицательное смещение. Прибавляя это смещение к местонахождению Pawn, мы задаём направление его движения.



Клавиши клавиатуры могут подавать на выход только значения 1 или 0, то можно использовать *scale* для преобразования их в отрицательные числа. Это можно сделать, взяв значение оси и умножив его на масштаб.

Умножив положительное (значение оси) на отрицательный (масштаб), мы получим отрицательное значение.

Задайте масштаб клавиш *S* и *A*, нажав на поле *Scale* и введя *-1*.

Теперь начинается интересное: заставим Pawn двигаться! Закройте Project Settings и откройте *BP_Player* в Blueprints editor, дважды нажав на него.

Перемещение игрока

Сначала нам нужно выбрать события для привязок движения. *Нажмите правой клавишей мыши* на пустом пространстве в Event Graph, чтобы открыть список нодов. Найдите в этом меню *MoveForward*. Добавьте нод *MoveForward* из списка *Axis Events*. Учтите, что вам нужен красный нод в Axis Events, а не зелёный нод в Axis Values.

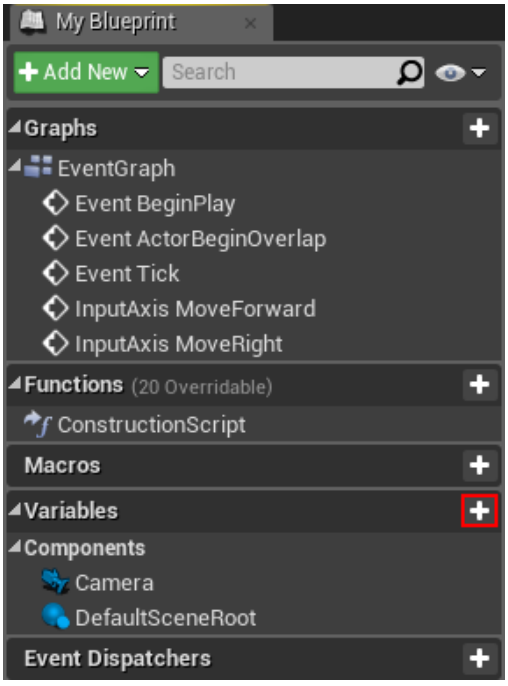
Повторите процесс для *MoveRight*.

Теперь мы настроим ноды для *MoveForward*.

Использование переменных

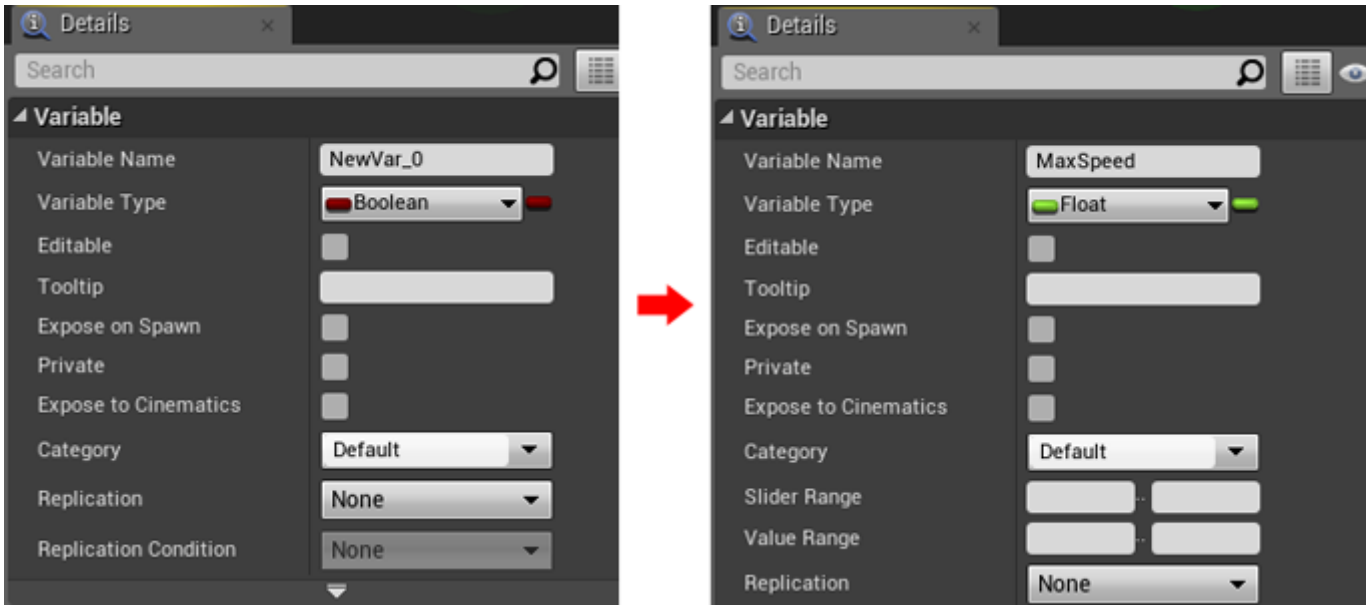
Для перемещения необходимо указать, с какой скоростью будет двигаться Pawn. Один из простых способов указания скорости — хранение её в *переменной*.

Чтобы создать переменную, зайдите во вкладку My Blueprint и нажмите на значок + в правой части раздела *Variables*.



Выбрав новую переменную, перейдите во вкладку Details. Измените имя переменной на

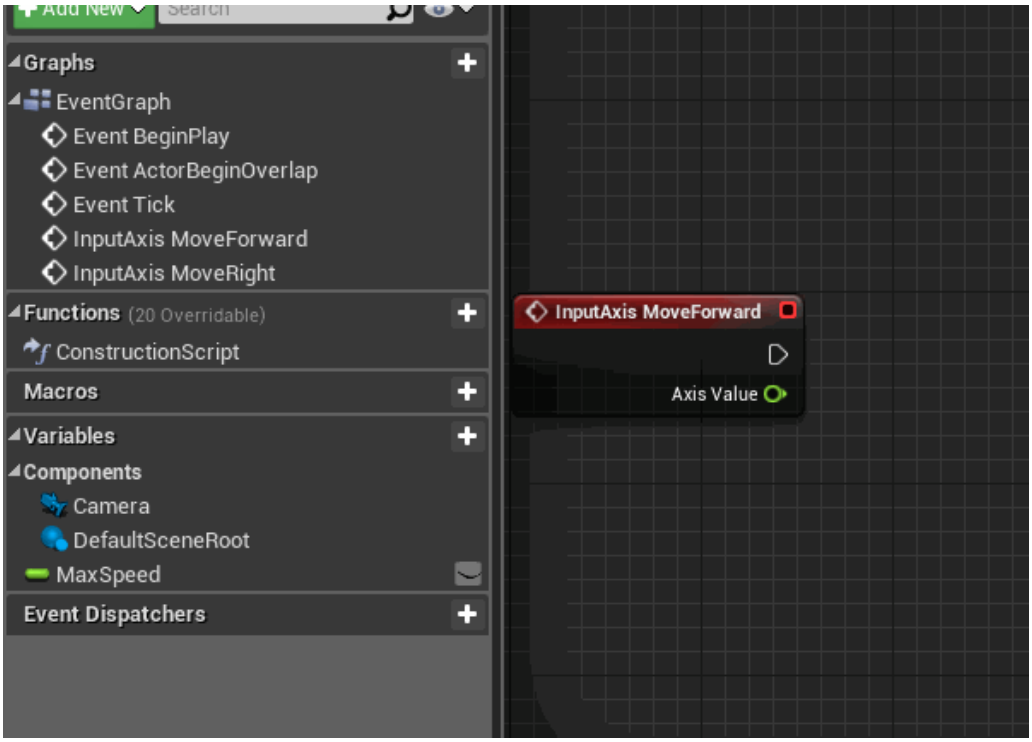
MaxSpeed. После этого замените тип переменной на *Float*. Для этого нужно нажать на *раскрывающийся список* рядом с *Variable Type* и выбрать *Float*.



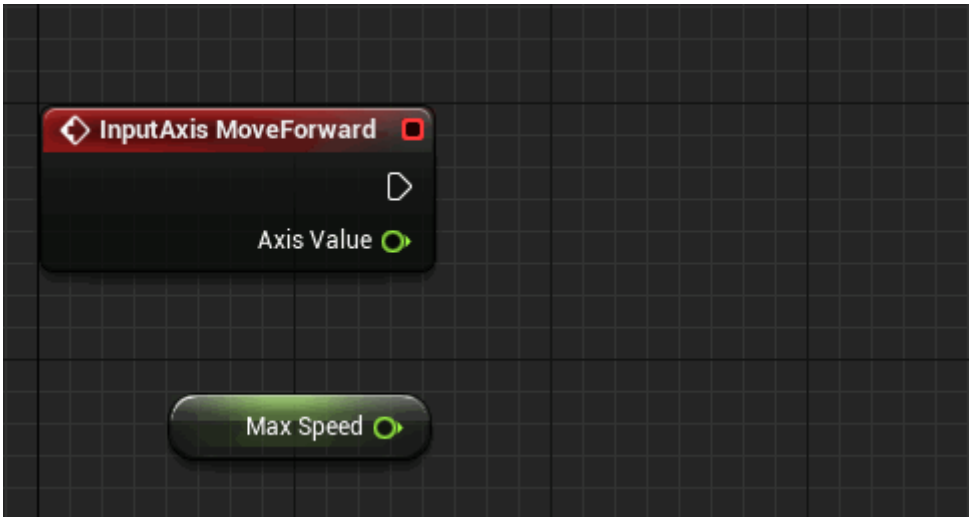
Теперь необходимо задать значение по умолчанию. Но чтобы его задать, нужно будет нажать *Compile* в Toolbar.

Выбрав переменную, перейдите ко вкладке Details. Зайдите в раздел *Default Value* и измените значение *MaxSpeed* по умолчанию на *10*.

Затем *перетащите* переменную *MaxSpeed* из вкладки My Blueprint на Event Graph. Выберите из меню пункт *Get*.

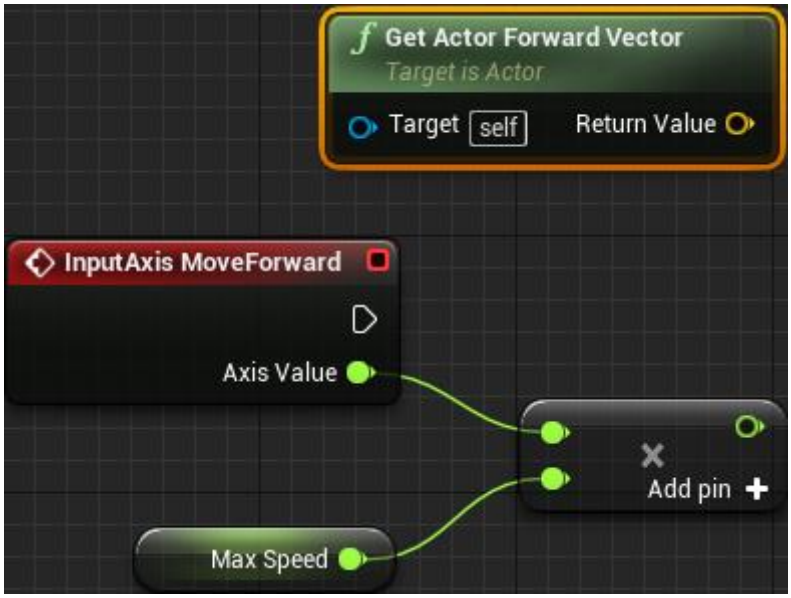


Теперь нужно умножить *MaxSpeed* на *значение оси*, чтобы получить конечную скорость и направление. Добавим нод *float * float* и присоединим к нему *Axis Value* и *MaxSpeed*.

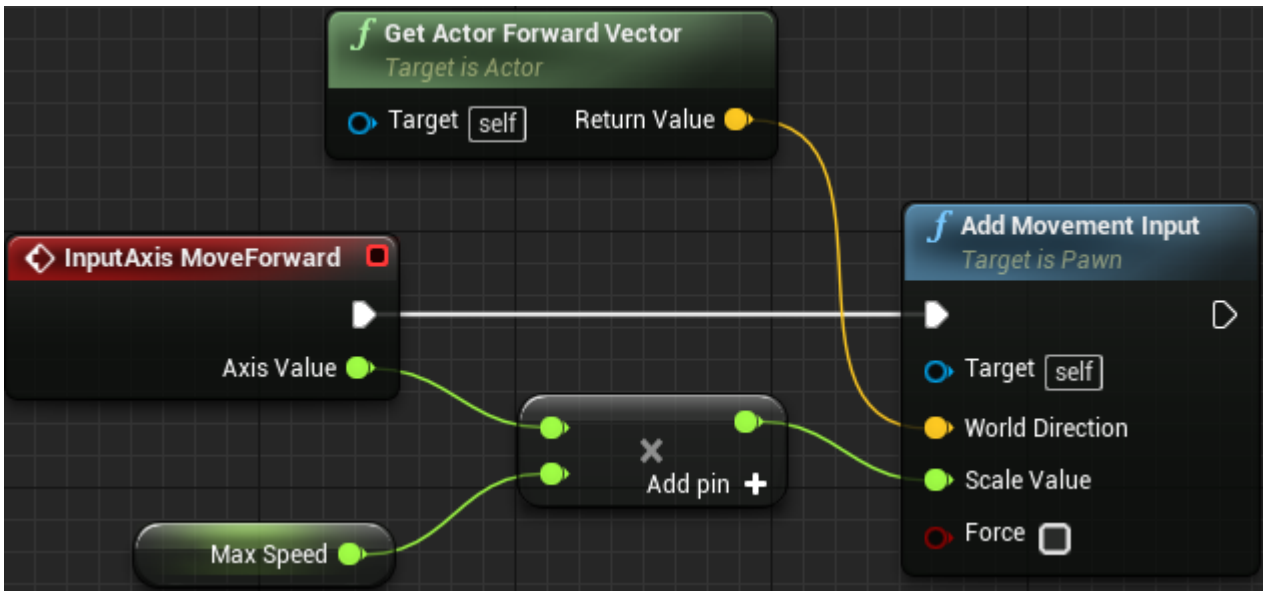


Получение направления игрока

Чтобы двигаться вперёд, нам нужно знать, куда смотрит Pawn. К счастью, в Unreal есть для этого нод. Добавьте нод *Get Actor Forward Vector*.



Затем добавьте нод *Add Movement Input*. Этот нод получает направление и значение, преобразуя их в хранимое смещение. Соедините ноды следующим образом:

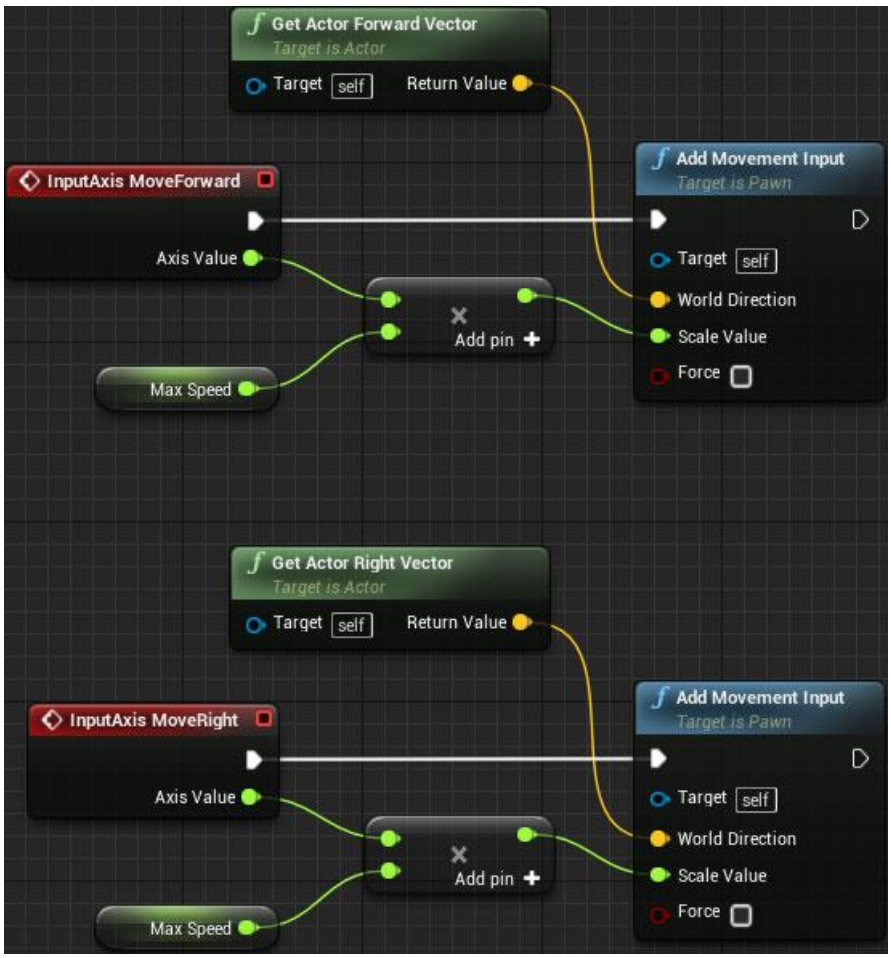


Белая линия обозначает цепочку выполнения. Другими словами, когда игрок перемещает ось ввода, то генерируется событие, выполняющее нод *InputAxis MoveForward*. Белая линия показывает, что после этого выполняется нод *Add Movement Input*.

Нод *Add Movement Input* получает на входе следующие данные:

- *Target*: задайте self, что в нашем случае является персонажем игрока (красным кубом).
- *World Direction*: направление для движения цели, которое в нашем случае является направлением, в котором смотрит игрок.
- *Scale Value*: как далеко мы двигаем игрока, в нашем случае это макс_скорость * значение_оси (которое, как мы помним, является значением в интервале от -1 до 1).

Повторим процесс для *MoveRight*, но заменим *Get Actor Forward Vector* на *Get Actor Right Vector*. Попробуйте сделать всё сами, не сверяясь с вышеприведёнными инструкциями!

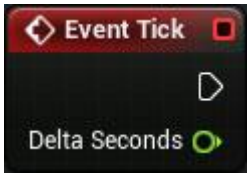


Добавление смещения

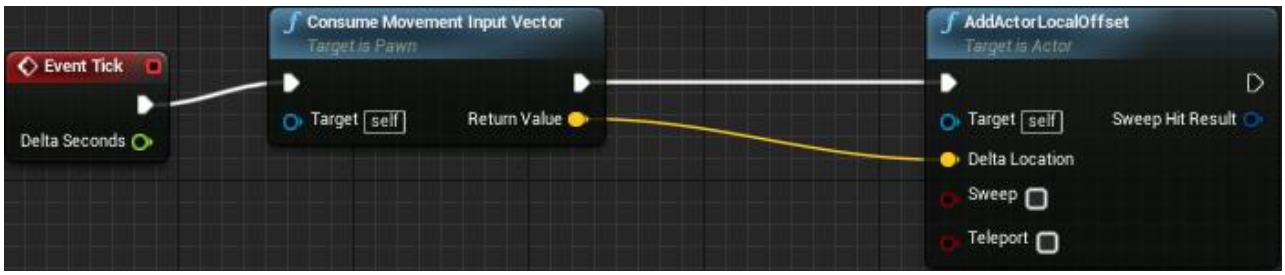
Чтобы действительно двигать Pawn, нам нужно получить смещение, вычисленное Add Movement Input, и прибавить его к местоположению Pawn.

В сущности, наша стратегия будет заключаться в перемещении игрока на небольшую величину в каждом кадре игры, поэтому нам нужно добавить перемещение к событию *Event Tick*, которое генерируется каждый кадр.

Перейдите к ноду *Event Tick* в Event Graph. Он должен быть неактивным и находиться слева, но если его нет, то создайте нод самостоятельно.

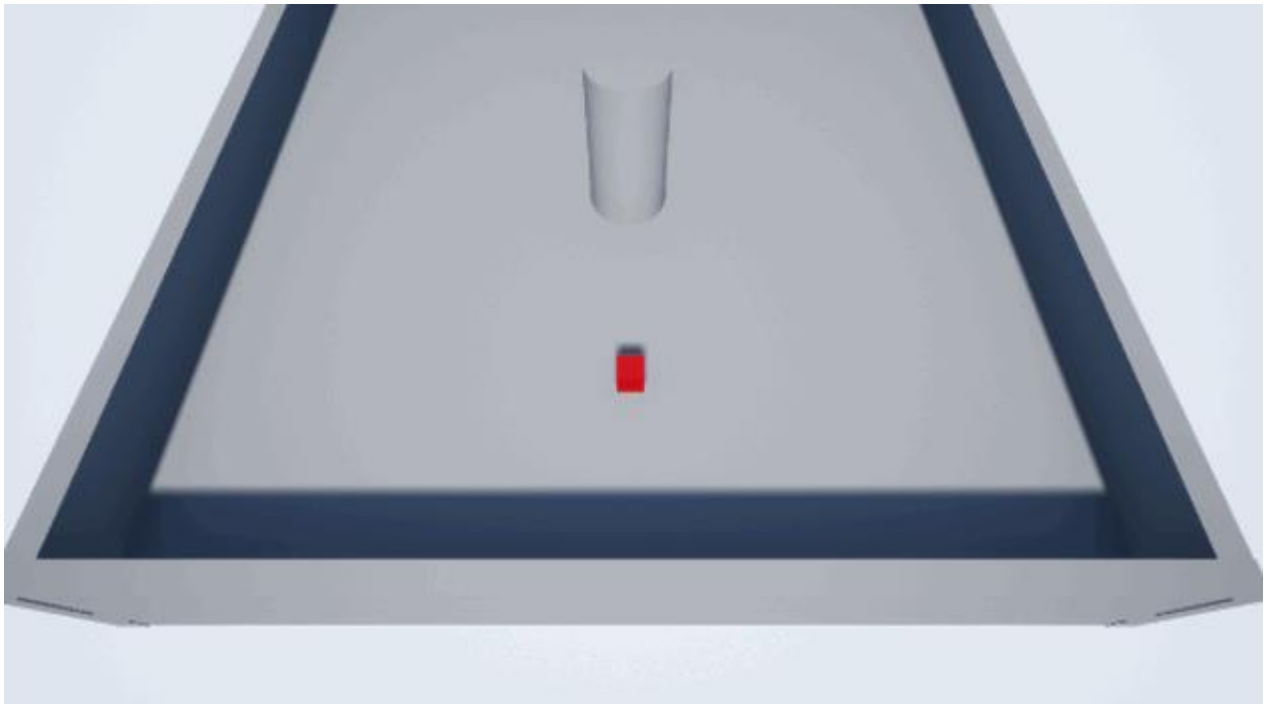


Чтобы получить смещение, создадим нод *Consume Movement Input Vector*. Чтобы прибавить смещение, создадим нод *AddActorLocalOffset*. После этого соединим их следующим образом:



Это означает, что в каждом кадре игры мы будем сохранять весь ввод перемещения и прибавлять его к текущему местоположению актора.

Нажмите *Compile*, перейдите к основному редактору и нажмите на *Play*. Теперь вы можете двигаться в сцене!



Однако у нас есть небольшая проблема. Мощные компьютеры могут рендерить кадры с большей частотой. Event Tick вызывается каждый кадр, поэтому ноды перемещения будут выполняться чаще. Это значит, что Pawn будет двигаться на мощных компьютерах быстрее, и наоборот.

Чтобы решить эту проблему, наше движение должно быть *независимым от частоты кадров*.

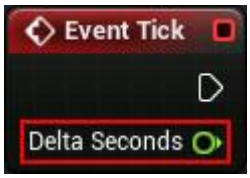
Примечание: я настроил привязки клавиш, чтобы показать влияние зависимости от частоты кадров. Нажмите *0*, чтобы ограничить частоту 60 кадрами в секунду, и нажмите *1*, чтобы снять ограничение. Попробуйте перемещаться при обоих частотах кадров и вы заметите разницу в скорости.

Независимость от частоты кадров

Независимость от частоты кадров означает, что мы постоянно будем получать одинаковые результаты, вне зависимости от частоты кадров. К счастью, достичь такой независимости

в Unreal очень просто.

Выйдите из игры, а затем откройте *BP_Player*. Затем перейдите к узлу *Event Tick* и посмотрите на *Delta Seconds*.

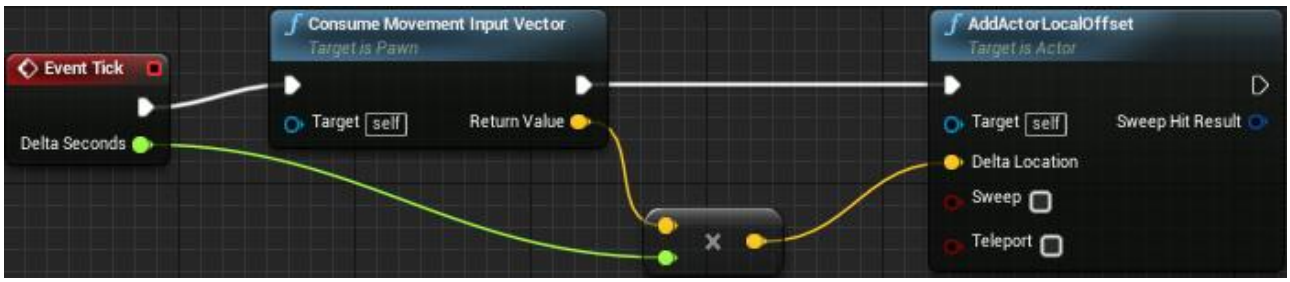


Delta Seconds — это величина времени, прошедшего после предыдущего Event Tick. Умножив смещение на Delta Seconds, мы сделаем перемещение независимым от частоты кадров.

Например, наш Pawn имеет максимальную скорость 100. Если после предыдущего Event Tick прошла одна секунда, то Pawn переместится на полные 100 единиц. Если прошли полсекунды, то он переместится на 50 единиц.

Если движение зависимо от частоты кадров, то Pawn будет перемещаться на 100 единиц в каждом кадре, вне зависимости от времени между кадрами.

Чтобы умножить смещение на Delta Seconds, добавьте нод *vector * float*. После этого соедините ноды следующим образом:



Время между кадрами (Delta Seconds) очень мало, поэтому Pawn будет двигаться намного медленнее. Это можно исправить, заменив значение *MaxSpeed* по умолчанию на на *600*.

Поздравляю вам удалось добиться независимости от частоты кадров!

Можно заметить, что куб проходить сквозь все объекты. Чтобы исправить это, нам нужно познакомиться с *коллизиями*.

Надевайте шлем, потому что сейчас нам придётся *столкнуться* с теорией!

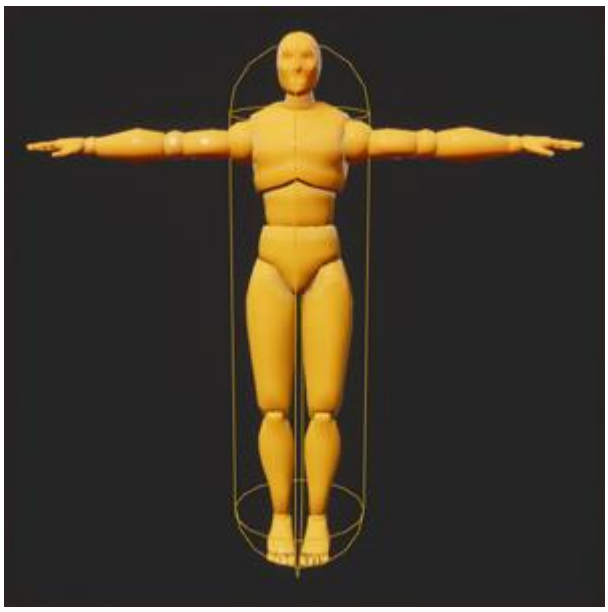
Коллизии актора

Когда мы вспоминаем о столкновениях, то представляем автомобильные аварии. К счастью, коллизии в Unreal намного безопаснее.

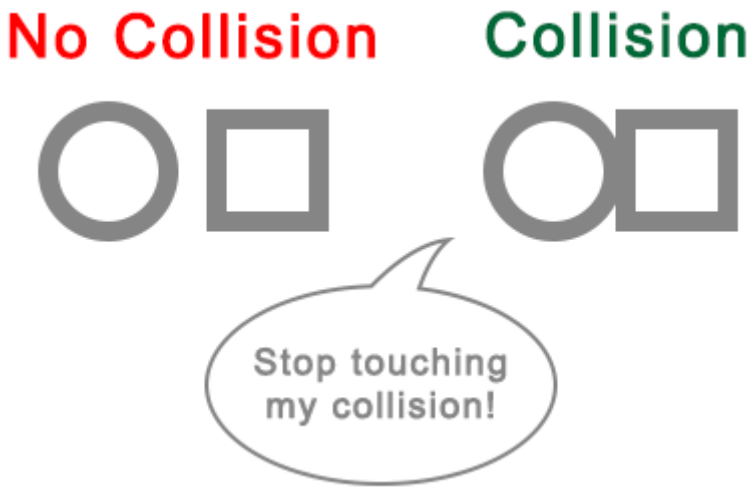
Чтобы иметь возможность сталкиваться с объектами, актору нужно обозначение его пространства столкновений (обычно называемого коллизией). Можно использовать одно из следующих пространств:

- *Меш коллизии:* они автоматически генерируются (если выбрать такую опцию) при импорте мешей. Пользователь также может создать в 3D-редакторе произвольный меш коллизии. У красного куба уже есть автоматически сгенерированный меш коллизии.
- *Компоненты коллизии:* они могут иметь одну из трёх форм: параллелепипед, капсула и сфера. Их можно добавлять на панели Components. Обычно используются для простых коллизий.

Ниже представлен пример персонажа и его коллизии.



Коллизия происходит, когда коллизия актора касается коллизии другого актора.



Теперь настало время включить коллизии.

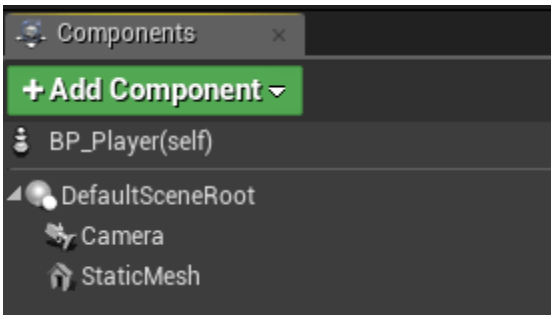
Включение коллизий

Вы, наверно, недоумеваете, почему куб не сталкивается с объектами, хотя у него есть меш коллизии. При перемещении актора Unreal учитывает для коллизий только *корневой* компонент. Поскольку корневой компонент Pawn не имеет коллизии, он проходит сквозь все объекты.

Примечание: актор, не имеющий коллизии в корневом компоненте, всё равно может блокировать других акторов. Но если *перемещать* актора, то он не будет ни с чем сталкиваться.

Итак, чтобы использовать меш коллизии, *StaticMesh* должен быть *корневым*. Для этого перейдите в панель Components. Затем *зажмите левую клавишу мыши* и *перетащите*

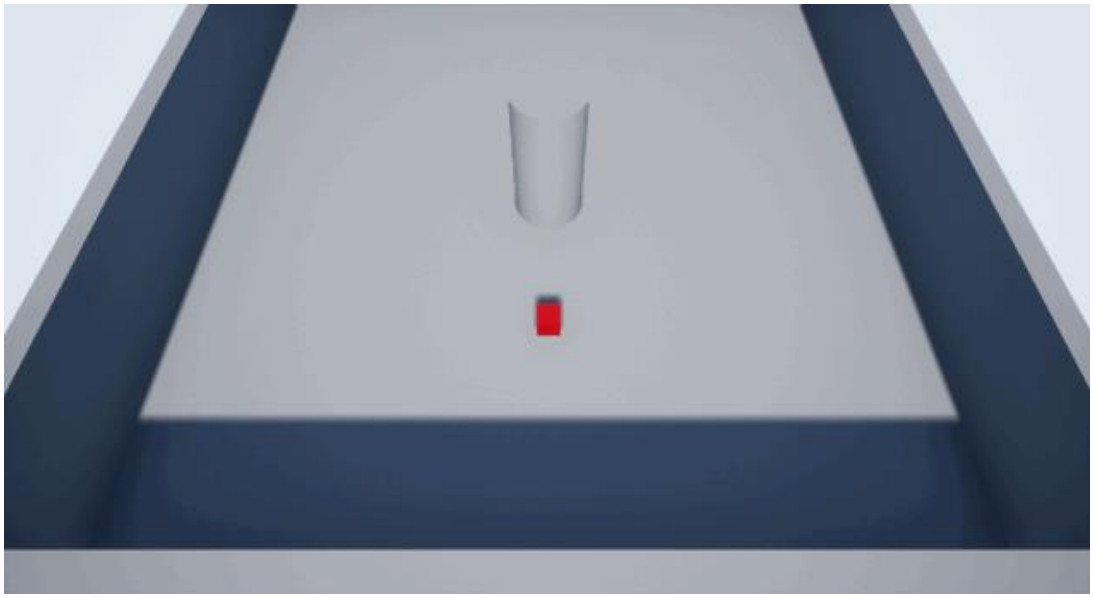
StaticMesh на *DefaultSceneRoot*. Отпустите *левую клавишу мыши*, чтобы сделать *StaticMesh* новым корневым компонентом.



Чтобы коллизии начали работать, нужно выполнить ещё одно действие. Переключитесь на Event Graph и перейдите к узлу *AddActorLocalOffset*. Найдите вход *Sweep* и измените значение на *true*, *нажав левой клавишей мыши* на *флажок*.

AddActorLocalOffset занимается тем, что телепортирует актора в новое место. *Sweep* гарантирует, что актор будет сталкиваться со всем, что находится между старым и новым местоположением.

Перейдите в основной редактор и нажмите на *Play*. Теперь куб будет реагировать на коллизии с уровнем!



Последнее, что мы создадим — это *предмет*, исчезающий при контакте с персонажем игрока.

Создание предмета

В общем случае предметом является любой собираемый игроком объект. Мы используем в качестве предмета *BP_Banana*.

Чтобы распознать контакт куба и предмета, нам нужен нод события, срабатывающего при коллизии. Для генерирования таких событий можно использовать *реакции на коллизи*.

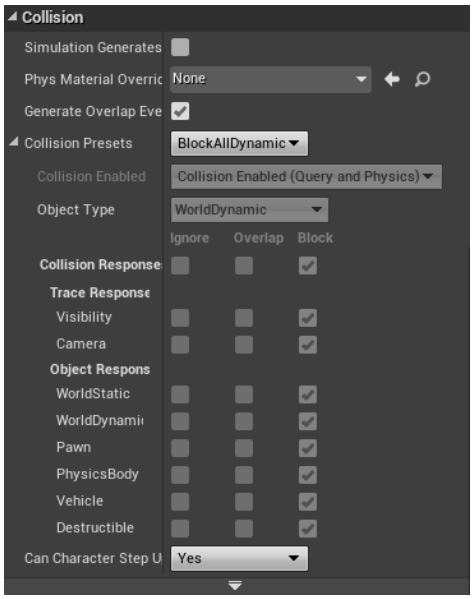
Реакция на коллизию также определяет, как актор реагирует на коллизию с другим актором. Существует три типа реакций на коллизии: *Ignore, Overlap* и *Block*. Вот как они взаимодействуют друг с другом:

Actor A	Actor B	Result
Ignore	Any response	Actors will pass through each other. No events will trigger.
Overlap	Overlap or Block	Actors will pass through each other. An overlap event will trigger for both actors.
Block	Block	Actors will block each other. A hit event will trigger for both actors.

Хотя здесь можно использовать и Overlap, и Block, в этом туториале мы будем использовать только Overlap.

Задание реакции на коллизию

Закройте игру и откройте *BP_Banana*. Выберите компонент *StaticMesh*, а затем перейдите в панель Details. Реакции на коллизии задаются в разделе *Collision*.

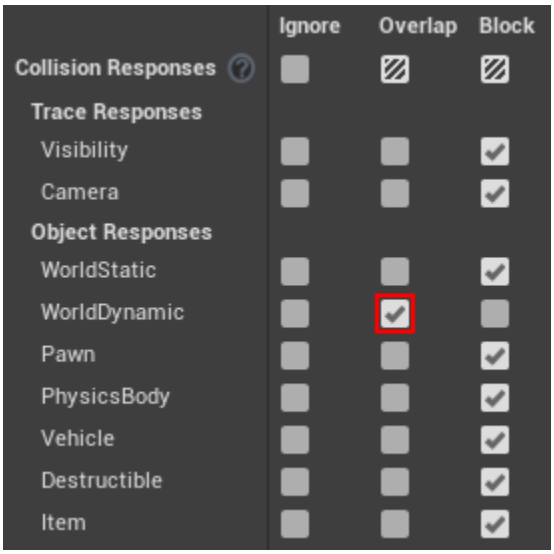


Как вы видите, большинство параметров неактивно. Чтобы сделать их изменяемыми, *нажмите* на *раскрывающийся список* рядом с *Collision Presets*. Выберите в списке *Custom*.

Теперь нам нужно указать реакцию на коллизию между предметом и кубом.

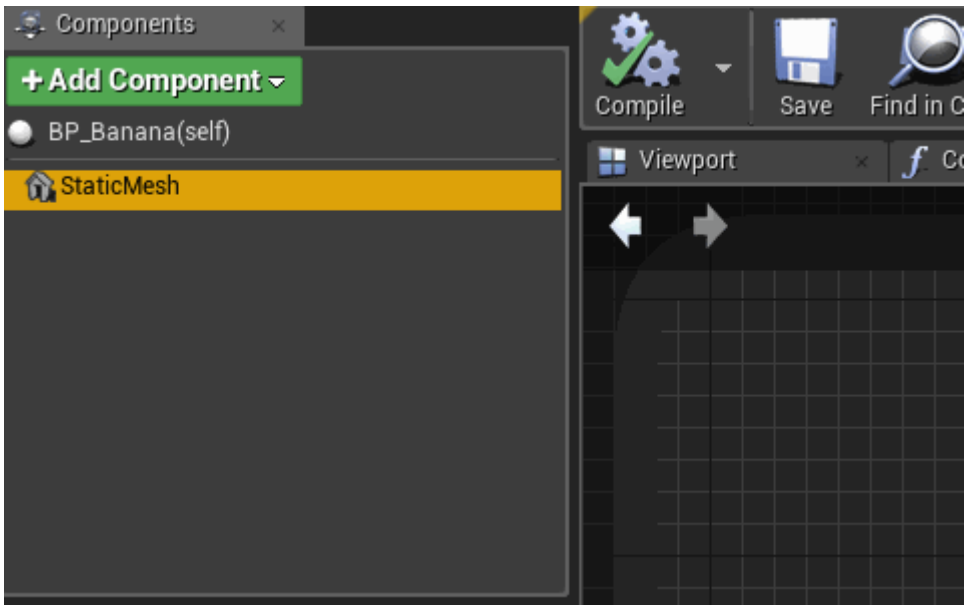
Компоненты имеют атрибут под названием *object type* (тип объекта). Тип объекта — это просто удобный способ группировки похожих акторов. Подробнее о типах объектов можно прочитать здесь.

Куб имеет тип *WorldDynamic*, поэтому нам нужно изменить реакцию на коллизию этого типа. В разделе *Collision Responses* измените реакцию на коллизию *WorldDynamic* на *Overlap*. Это можно сделать, *нажав* на *средний флажок* справа от *WorldDynamic*.

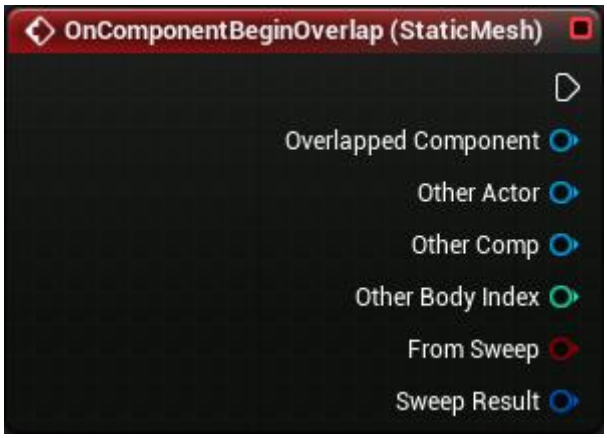


Обработка коллизий

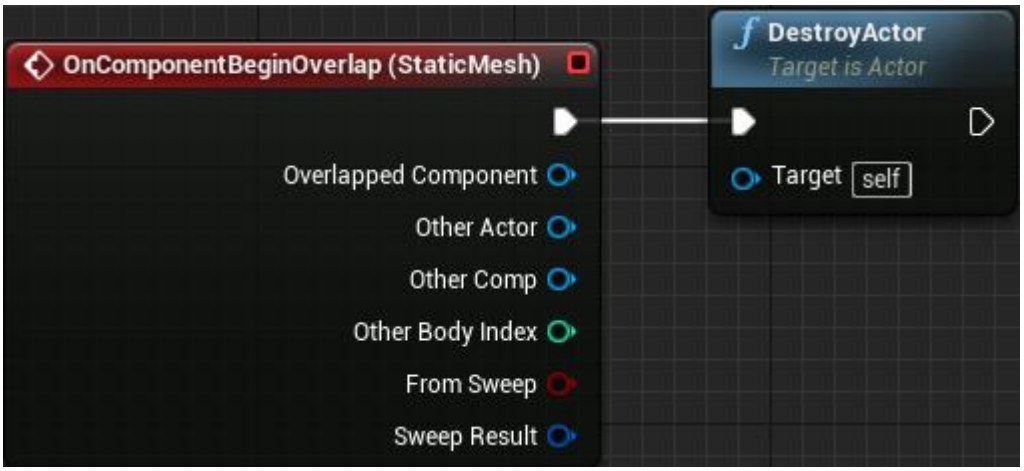
Для обработки коллизий нужно использовать событие *наложения*. Перейдите в панель Components и *нажмите правой клавишей мыши* на *StaticMesh*. В контекстном меню выберите *Add Event\Add OnComponentBeginOverlap*.



Так мы добавим в Event Graph нод *OnComponentBeginOverlap (StaticMesh)*.



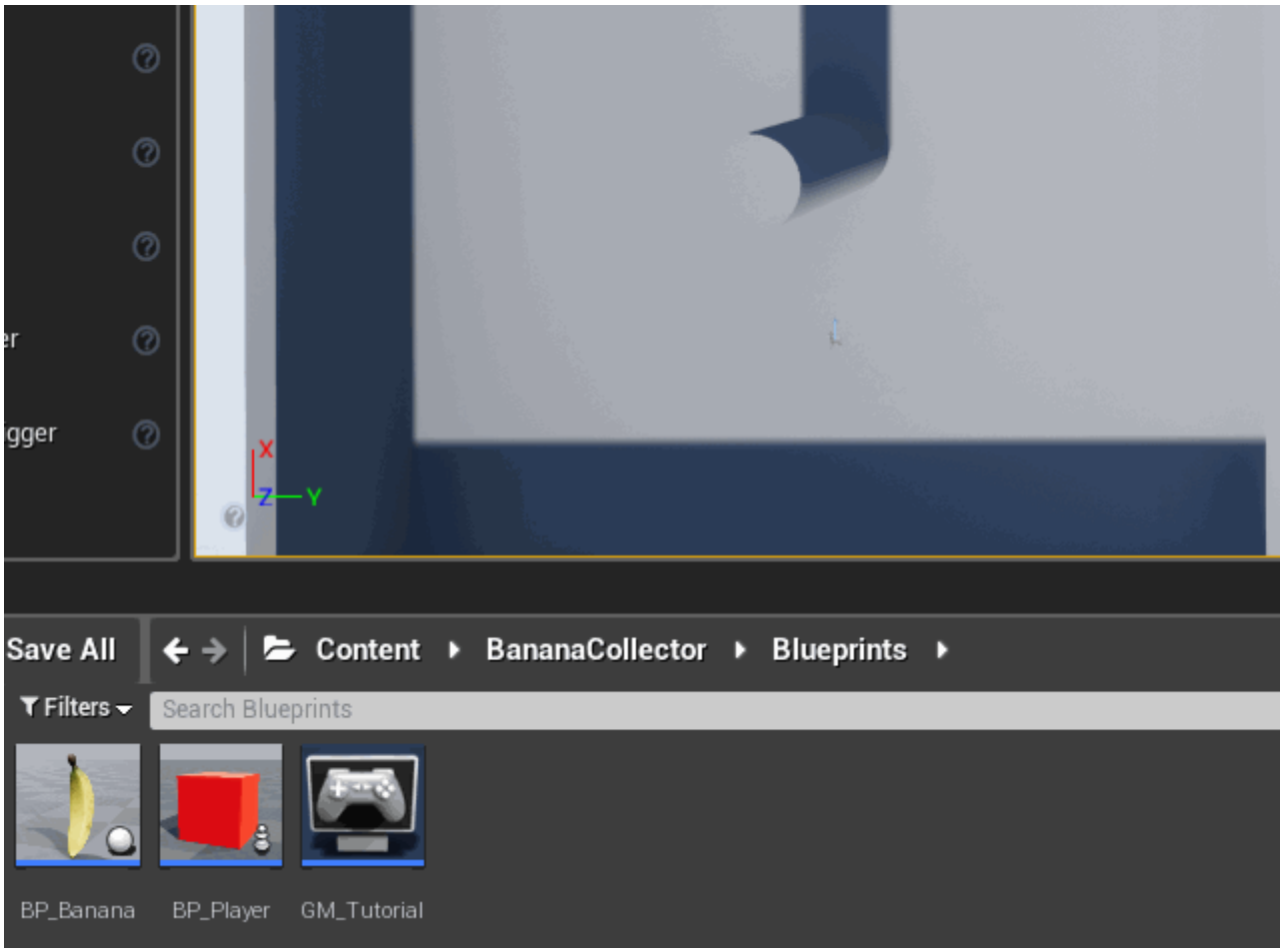
Наконец, создадим нод *DestroyActor* и соедините его с нодом *OnComponentBeginOverlap (StaticMesh)*. Как можно догадаться по названию, он удаляет целевой актор из игры. Однако поскольку целевого актора нет, он уничтожит актор, *вызвавший* его.



Размещение предмета

Закройте Blueprint editor и перейдите в папку *Blueprints*.

Начните располагать бананы на уровне *зажав левую клавишу мыши* и *перетаскивая BP_Banana* во Viewport.



Нажмите *Play* и начните собирать бананы!