

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»  
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ  
Кафедра компьютерной инженерии и моделирования

## **«Unreal Engine»**

Отчет по лабораторным работам  
по дисциплине «Компьютерная графика»  
студента 2 курса группы ИВТ-б-о-201(1)

Шор Константина Александровича

Направления подготовки 09.03.01«Информатика и вычислительная техника»

Симферополь, 2022

# 1. Знакомство с движком

## Импорт ассетов

Чтобы работать уже с существующими фалами, нужно их импортировать. Для этого нужно начать на *import* и выбрать нужные файлы.

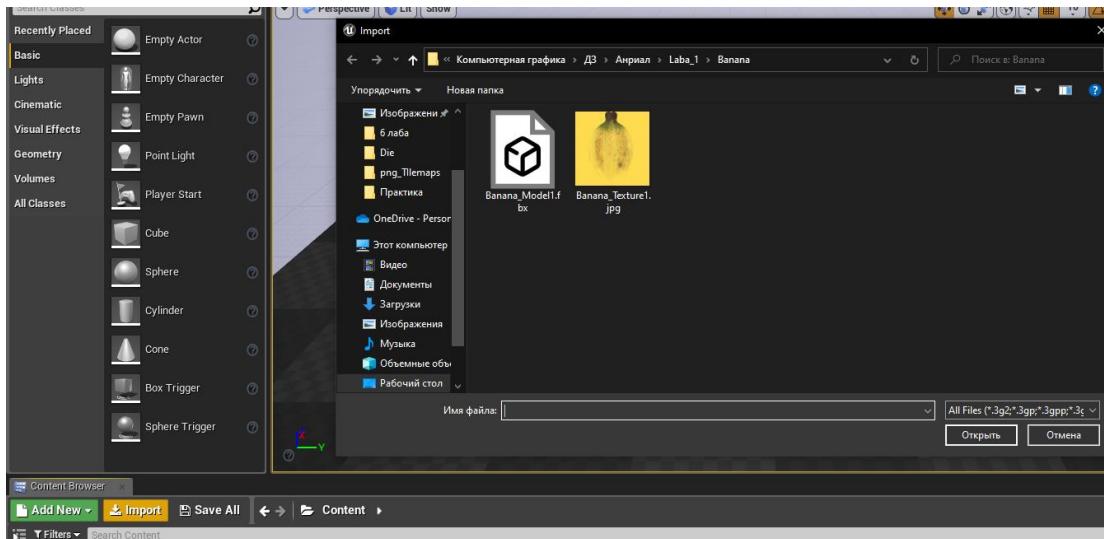


Рис. 1 Импорт ассетов

После выбора фалов появится меню импорта, в нём нужно снять галочку с *Import Materials*.

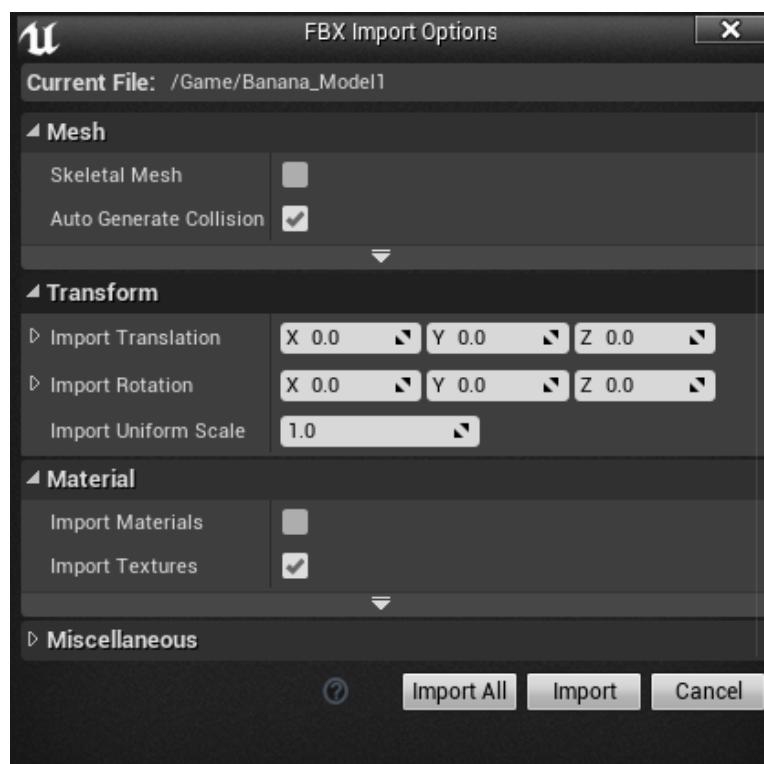


Рис. 2 Меню импорта

После удачного импорта должны появится два аксессуара в *Content Brows*

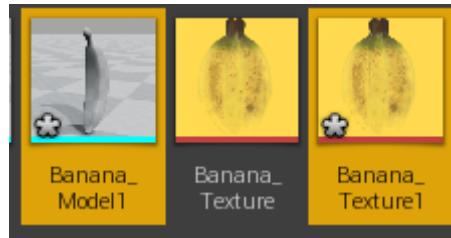


Рис. 3 Результат импорта

### Добавление мешей на уровень

Чтобы добавить меш (модель) на уровень нужно плавно перенести его мышью из *Content Brows* в *Viewport*

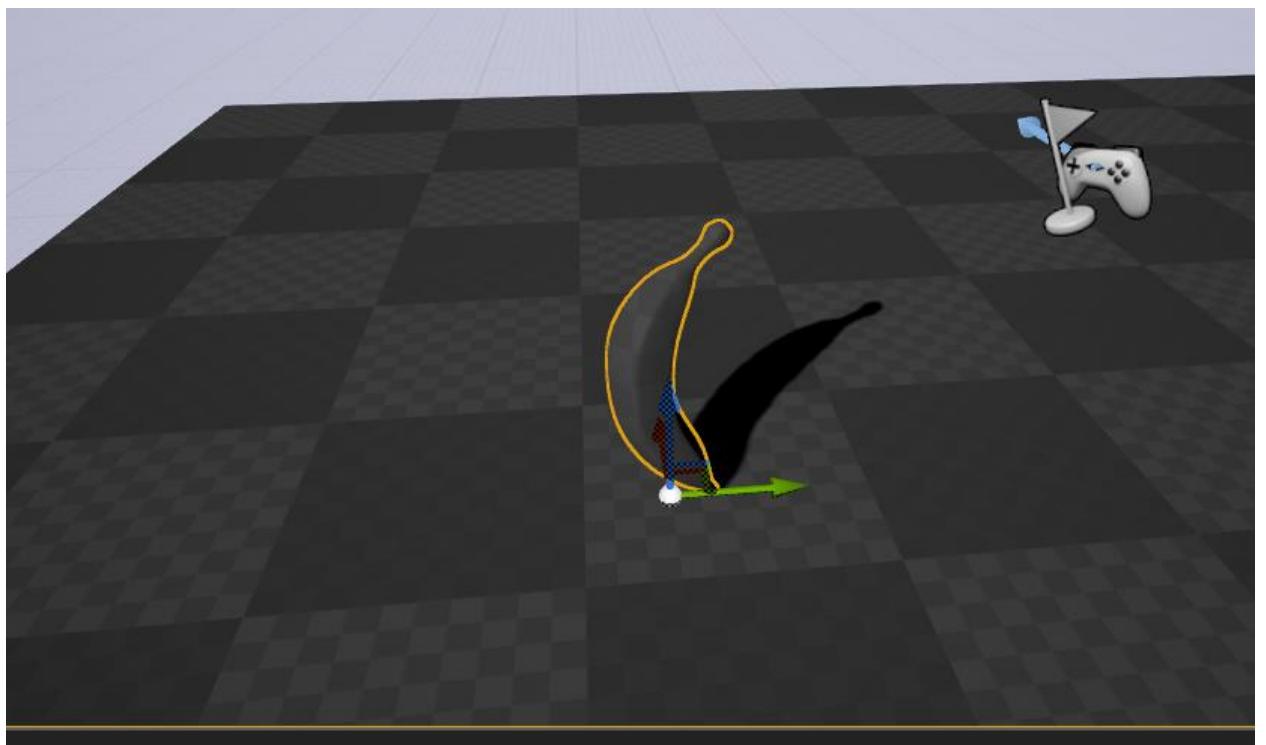


Рис. 4 Меш на сцене

Затем нужно создать материал меша. Для этого в *Content Brows* нажимаем на *Add New* выбираем *Material*. Нажимаем на него два раза и переходим в его редактирование

## Добавление текстур

В меню редактирование материала в панели Palette выбираем Texture Sample и перетаскиваем её на схему. Добавляем в неё нашу текстуру и соединяем блюпринты. Сохраняем и выходим.

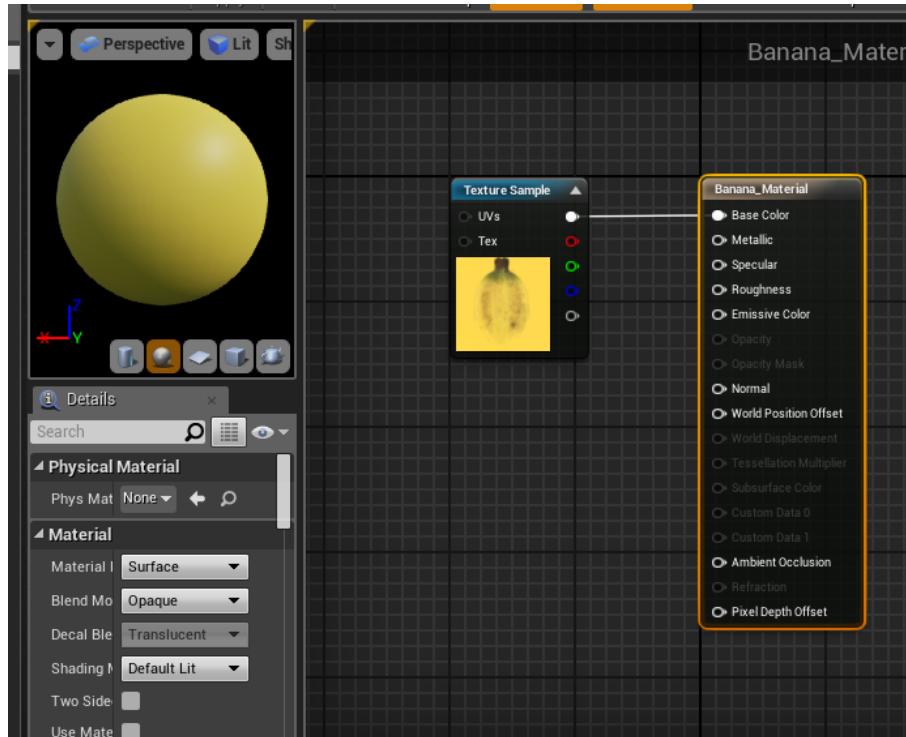


Рис. 5 Добавление текстур

## Использование материалов

В *Content Browser* дважды нажмите на *Banana\_Model* – это окно редактора нашего меша. Переходим в Details в раздел Material и выбираем уже созданный нами материал.

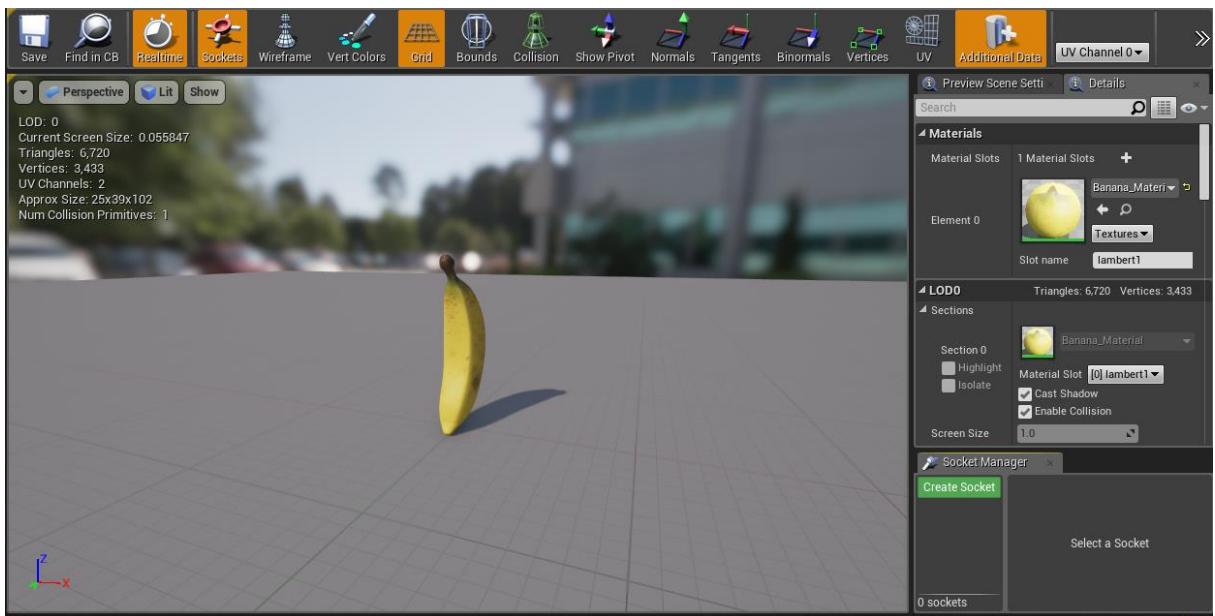


Рис. 6 Материал на банане

## Создание поворотного стола

Чтобы банан крутился нужно компоненты, которые будут это обеспечивать.

Для этого в *Content Browser* нажимаем на *Add New* выбираем *Blueprint Class*.

Нажимаем на него два раза и переходим в меню создания. Выбираем Actor.

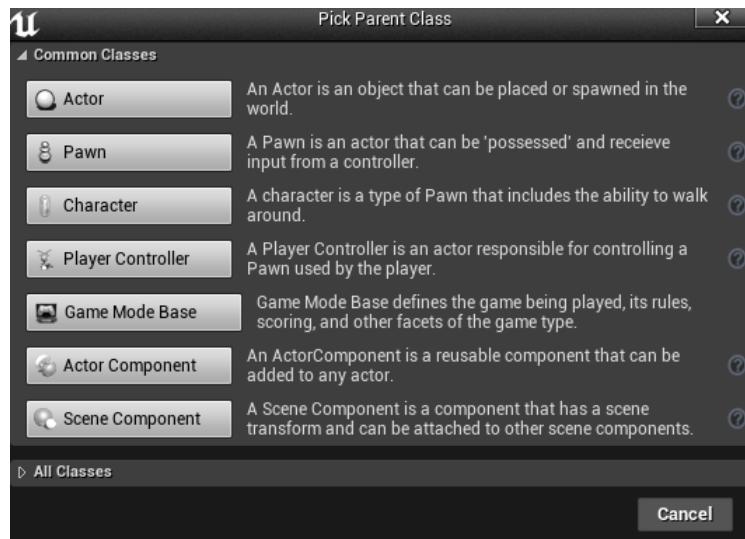


Рис. 7 Меню создания Blueprint

Чтобы добавить основание, перейдите в панель Components. Нажмите на *Add Component* и выберите *Cylinder*. Затем нажмите на *Add Component* и выберите из списка *Static Mesh*. Для отображения банана выберите компонент *Static Mesh*, а затем нажмите на вкладку *Details*. Нажмите на раскрывающий список в правой части *Static Mesh* и выберите *Banana\_Model*

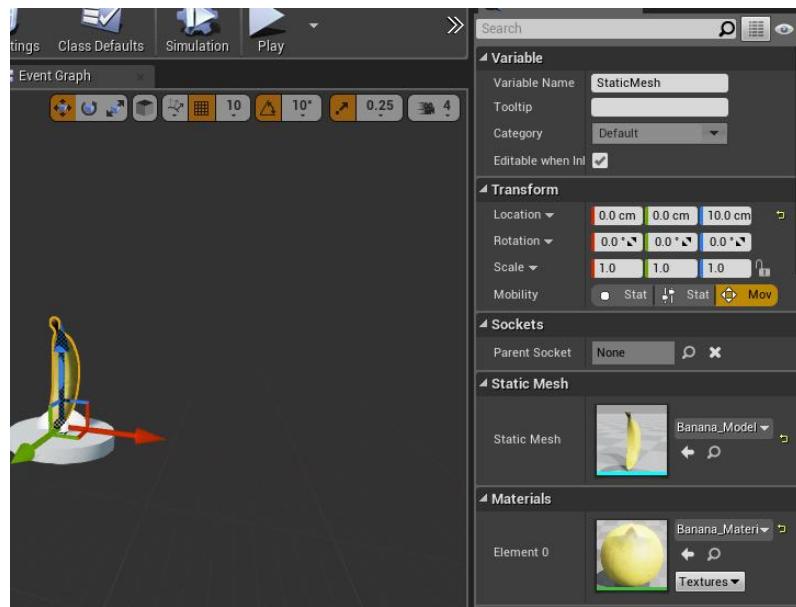


Рис. 8 Добавление банана

## Вращение стола

Чтобы заставить банан крутиться нужно создать событие. Для этого переходим в Event Graph и создаём блюпринт.

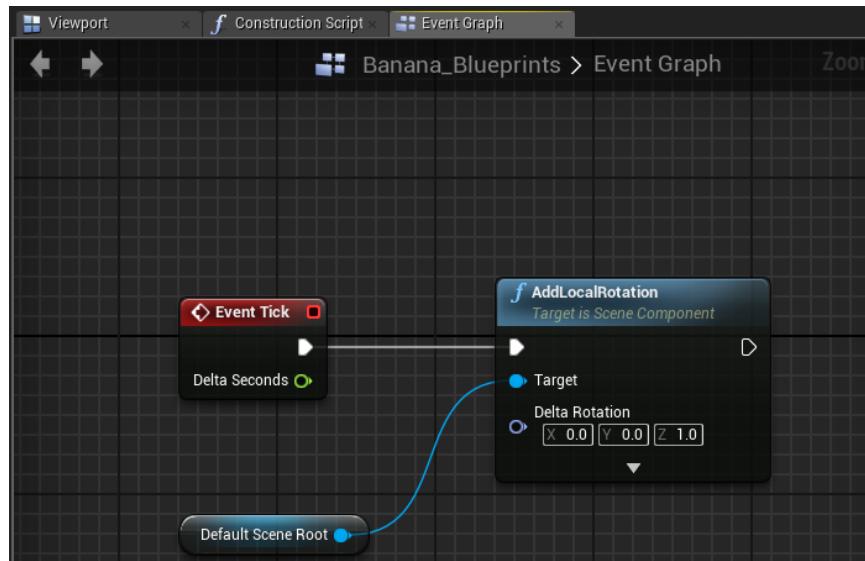


Рис. 9 Блюпринт вращения

Затем сохраняем всё и возвращаемся в главное меню. Нажимаем Play и банан крутится.

## 2. Blueprints

### Создание игрока

Нажмите на кнопку *Add New* и выберите *Blueprint Class*. Выберите во всплывающем окне *Pawn* и назовите его *BP\_Player*.

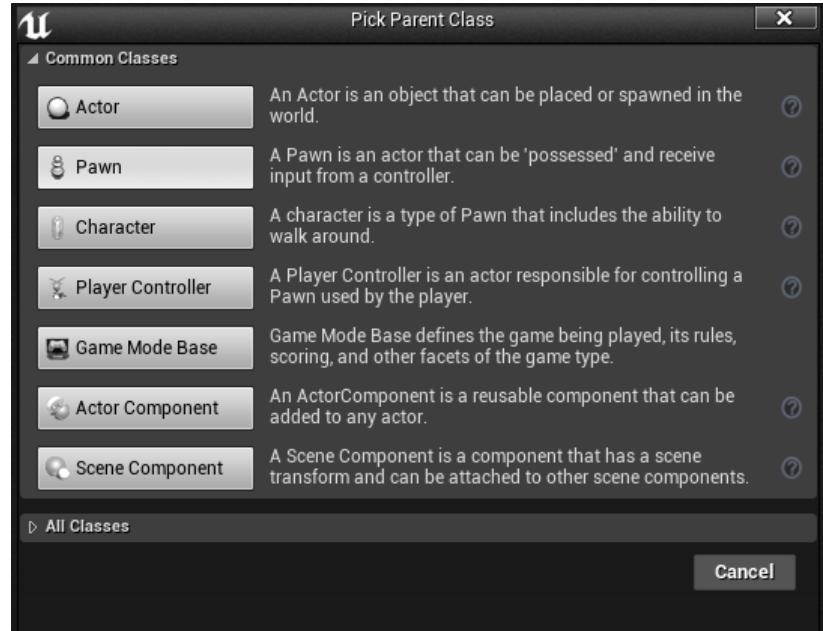


Рис. 1 Создание игрока

### Прикрепление камеры

Ведите координаты в поля *Location* (*-1100, 0, 2000*). Она находится в разделе *Transform* панели Details. Затем поверните камеру на *-60* по Y.

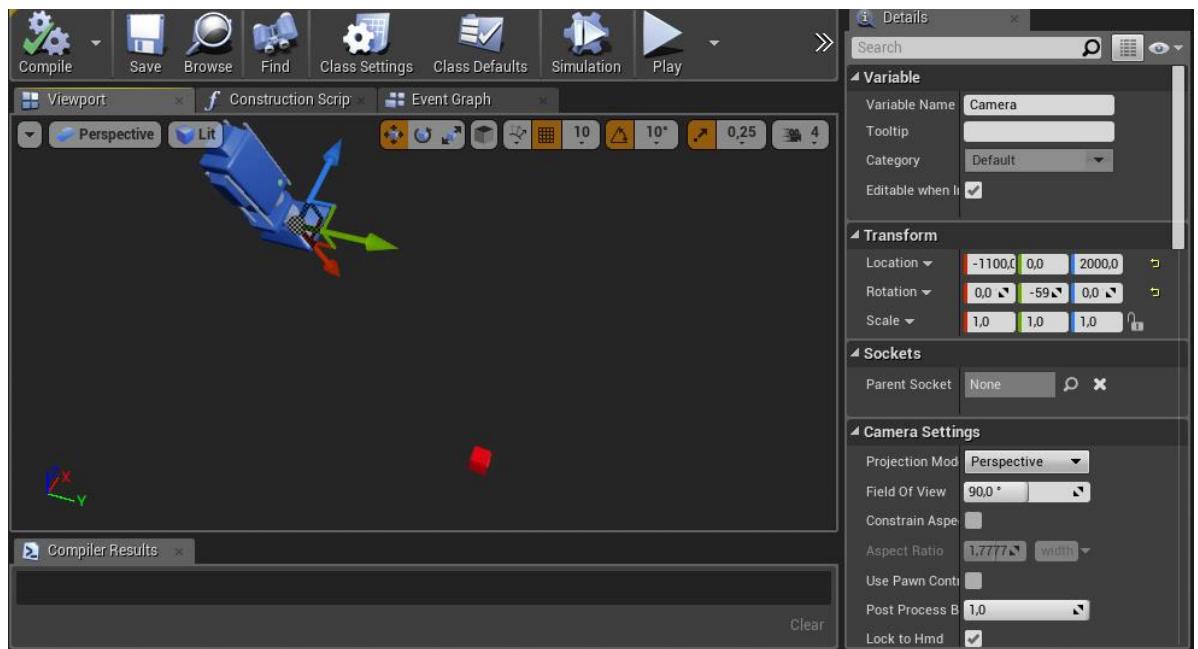


Рис. 2 Настройка камеры

## Отображение игрока

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*.

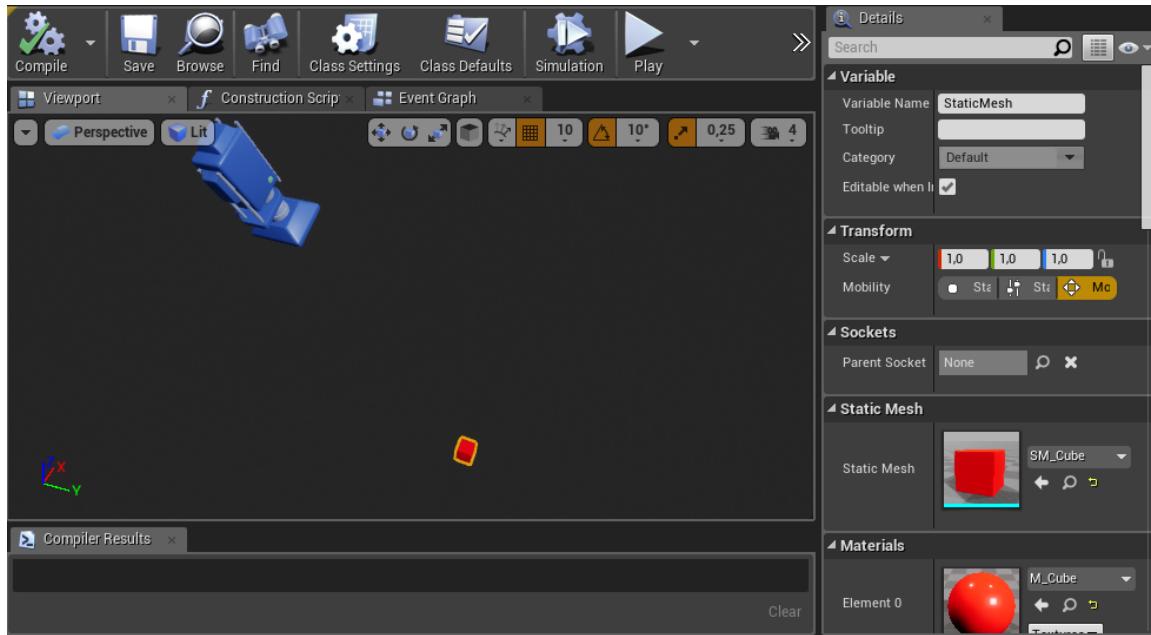


Рис. 3 Настройка игрока

## Создание Game Mode

Нажмите на кнопку *Add New* и выберите *Blueprint Class*. Выберите во всплывающем окне *Game Mode Base* и назовите его *GM\_Tutorial*

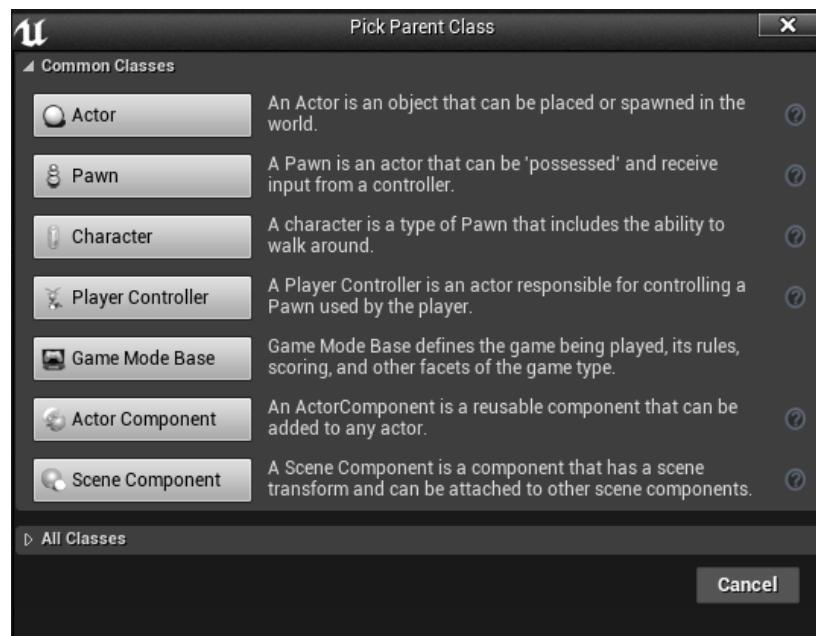


Рис. 4 Создание Game Mode

## Настройка Game Mode

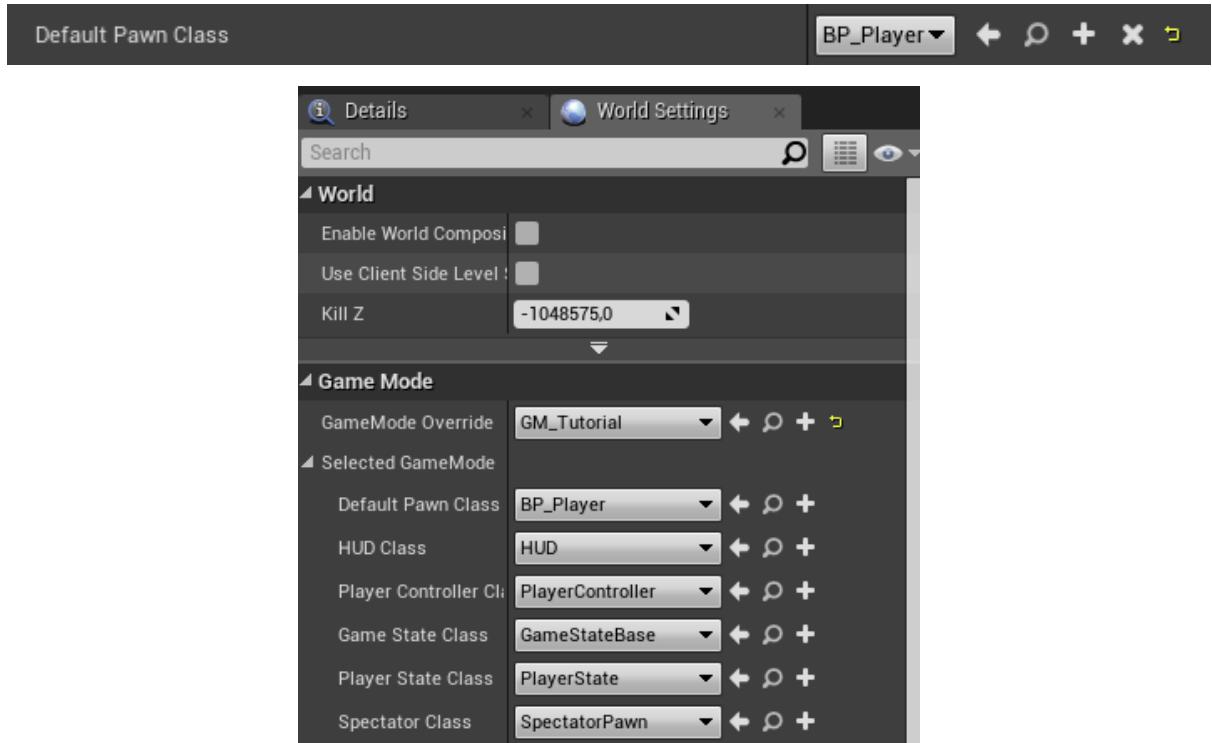


Рис. 5 Настройка Game Mode

## Привязка осей и действий

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

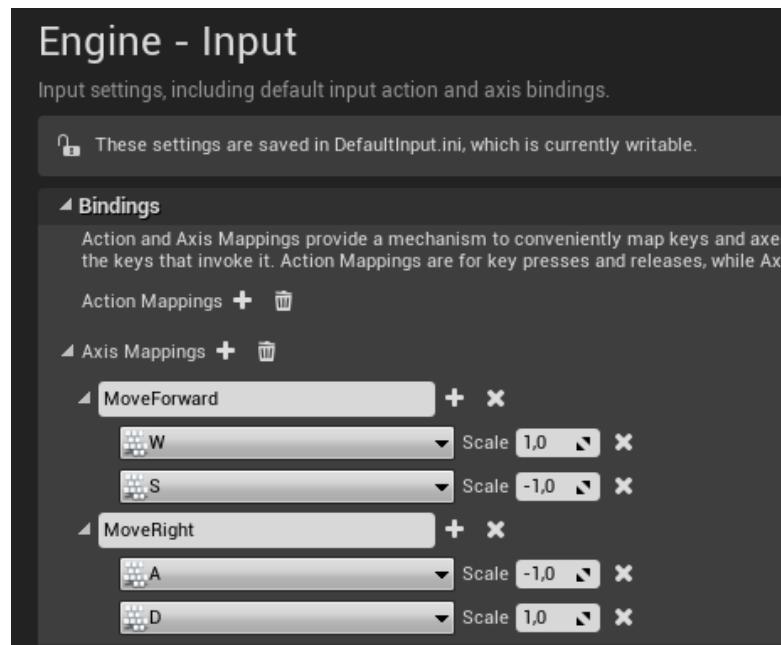


Рис. 6 Привязка осей

## Физика куба

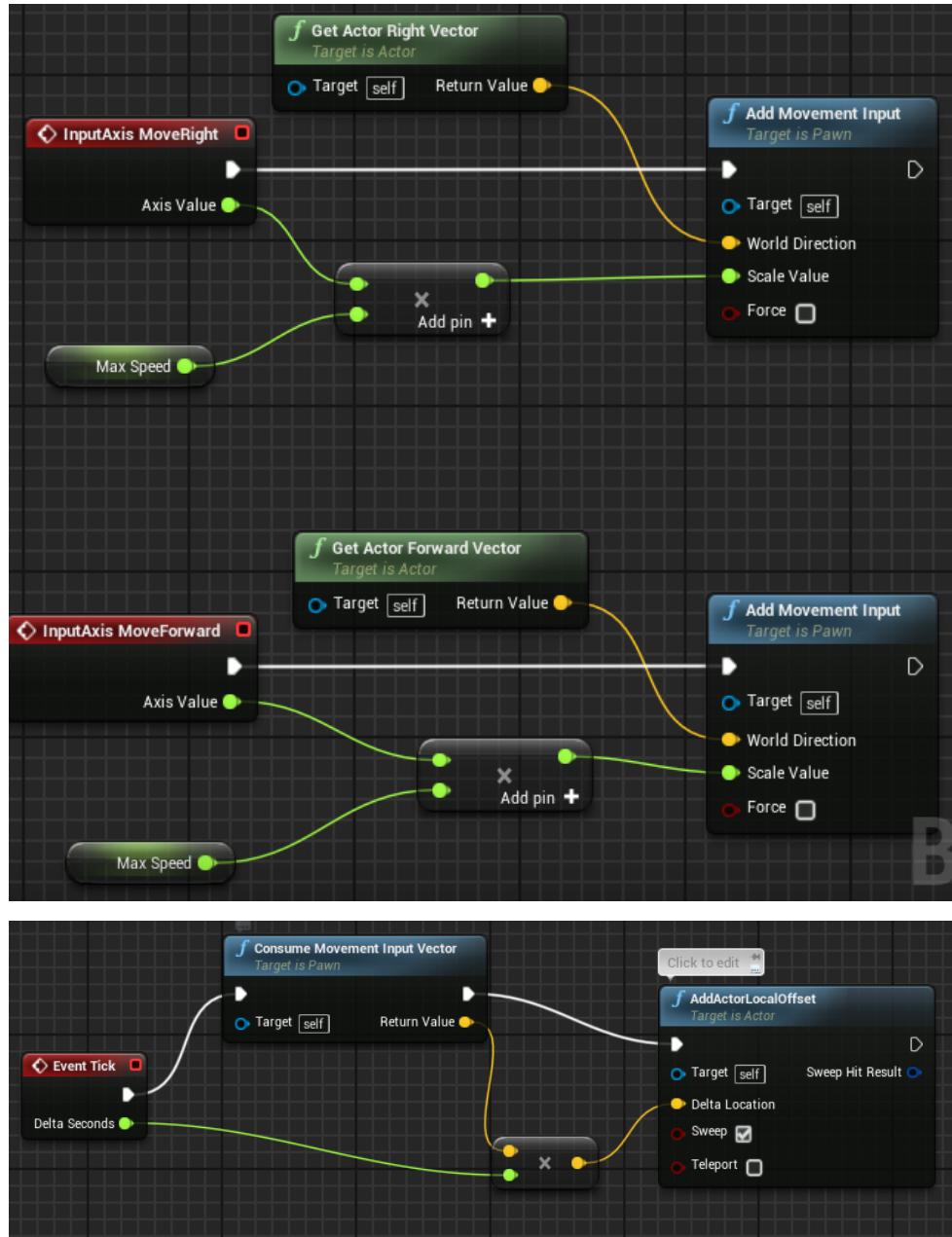


Рис. 7 Движение по оси X

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).

## Физика Банана

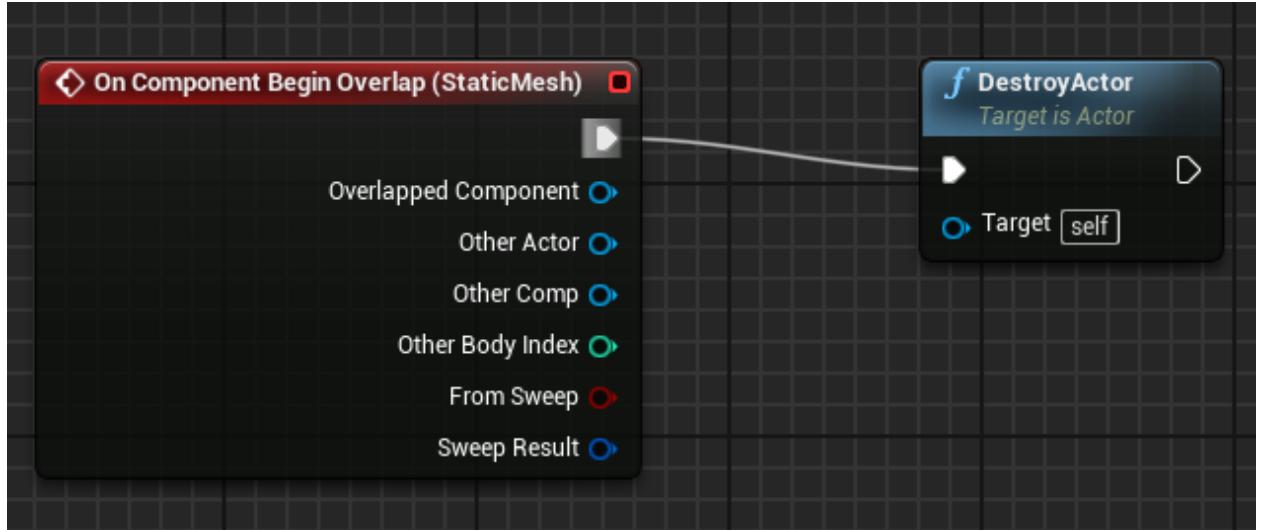


Рис. 8 Уничтожение банана

Если коллизии начинают взаимодействовать, то банан самоуничтожается

### 3. Материалы

#### Регулировка яркости

HueShift – меняет оттенок текстуры

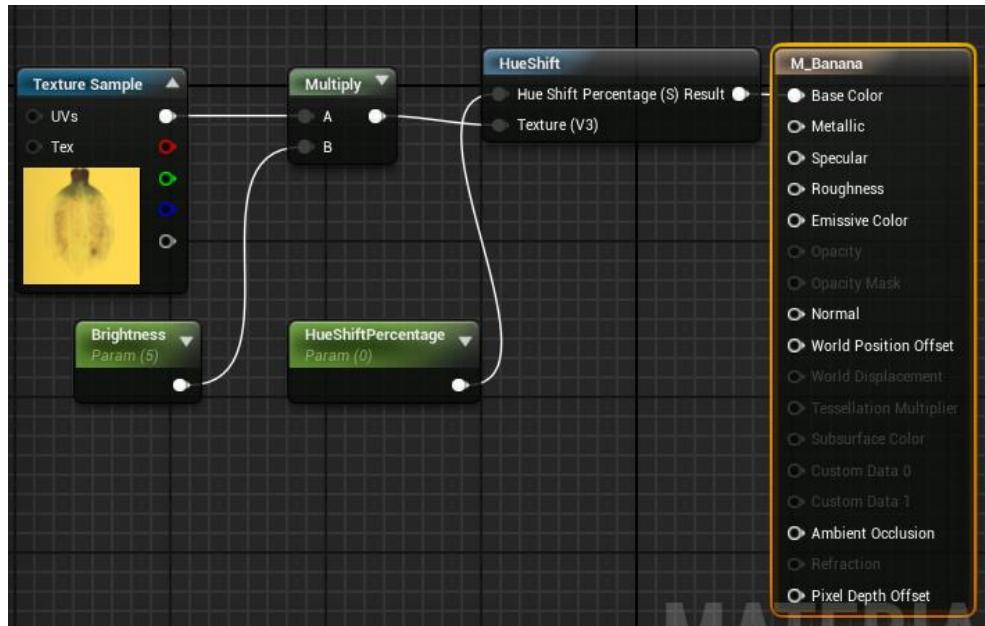


Рис. 9 Настройка яркости

#### Создание экземпляра материала

Меняя параметры *Brightness* и *HueShiftPercentage*, можно добиваться разных цветов

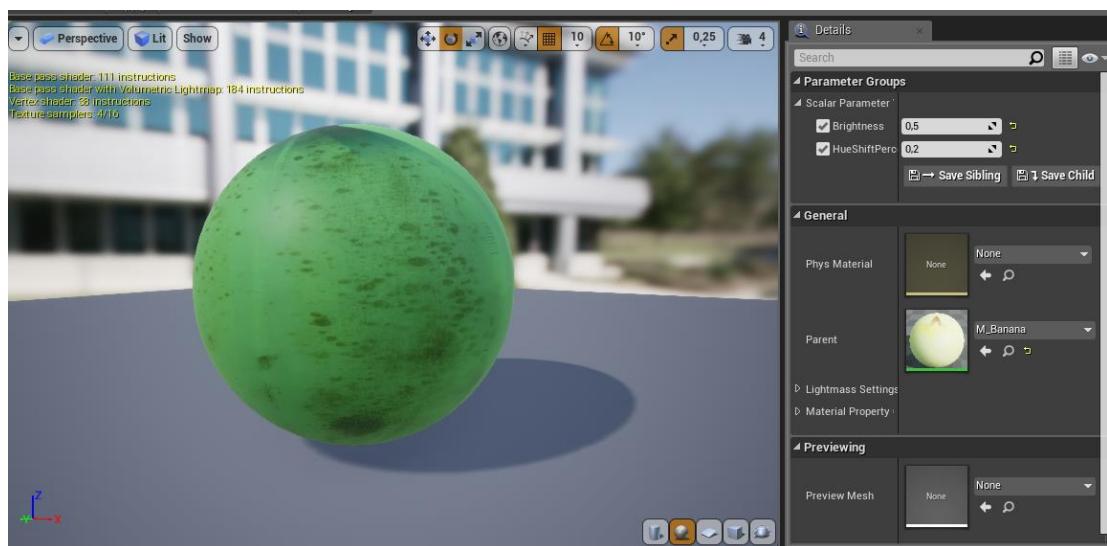


Рис. 10 Настройка цвета

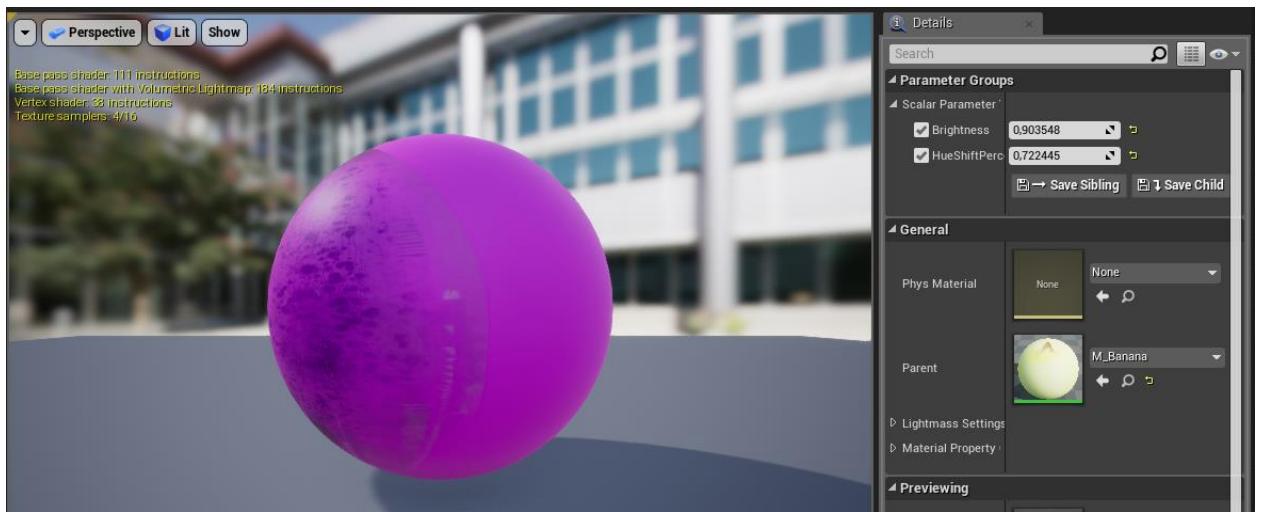


Рис. 11 Пример смены цвета

### Динамическое изменение материалов

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*.

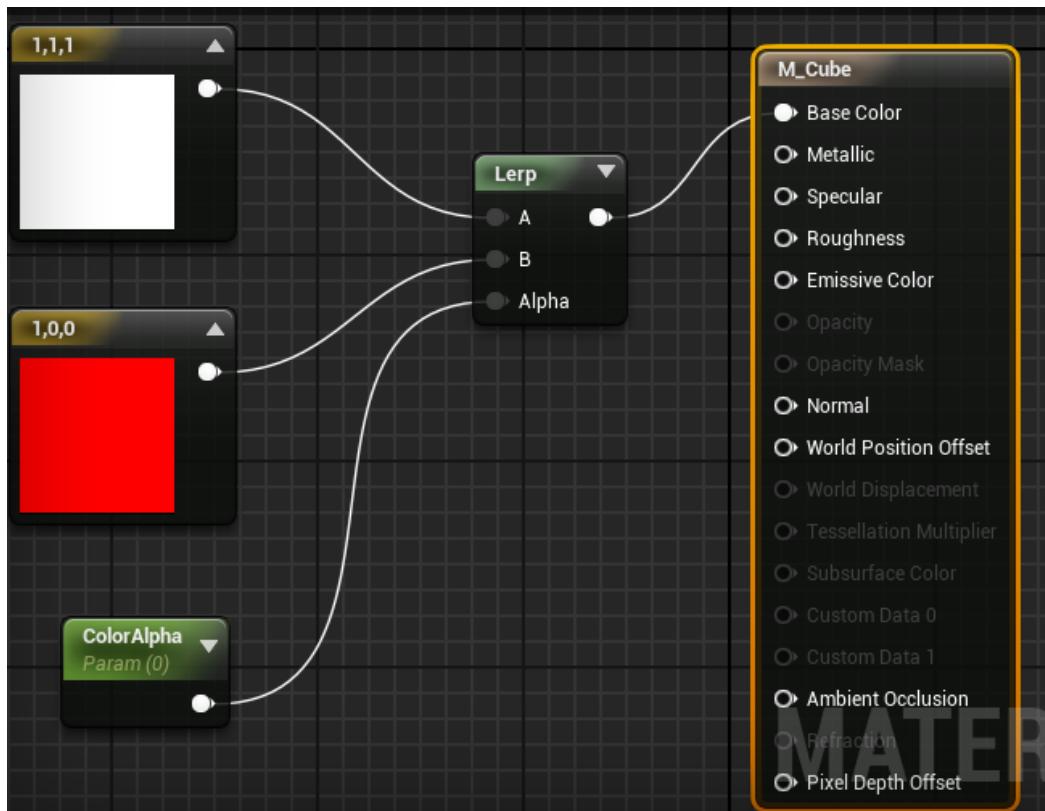


Рис. 12 Динамическое изменение

## Создание счётчика бананов и обновление материала

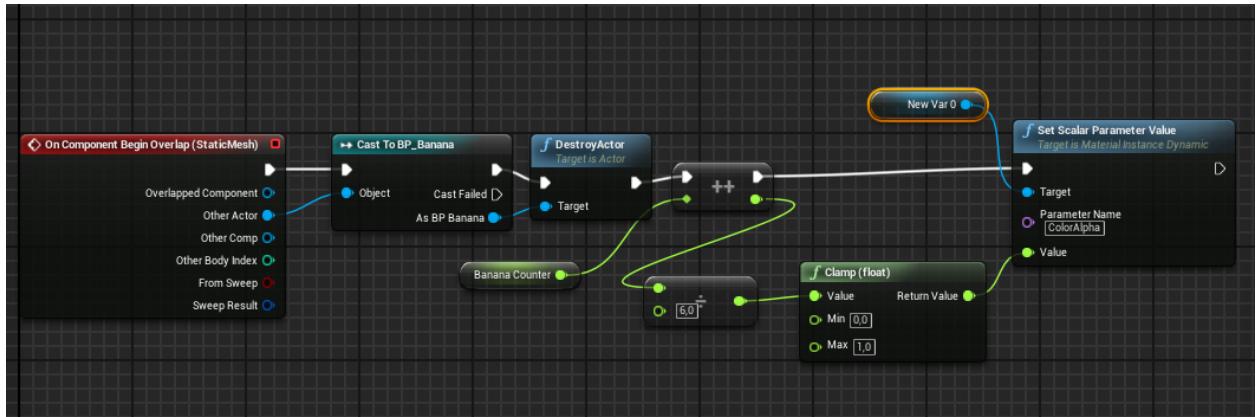


Рис. 13 Реализация динамического изменения цвета

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 нормализовать 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 нормализованной и ограниченной в интервале версией *BananaCounter*

## 4. UI (User Interface)

### Создание виджета

Перейдите в Content Browser и найдите папку *UI*. Нажмите на кнопку *Add New* и выберите *User Interface\Widget Blueprint*. Переименуйте новый ассет в *WBP\_HUD*.

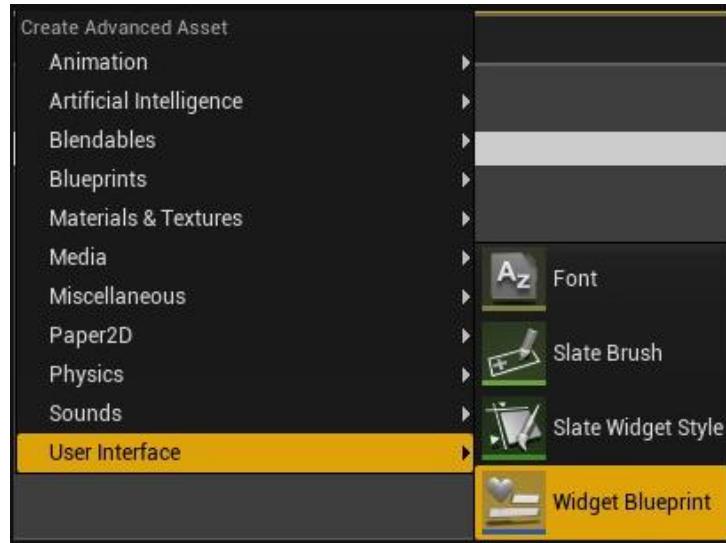


Рис. 14 Создание виджета

### Создание Текса и картинки

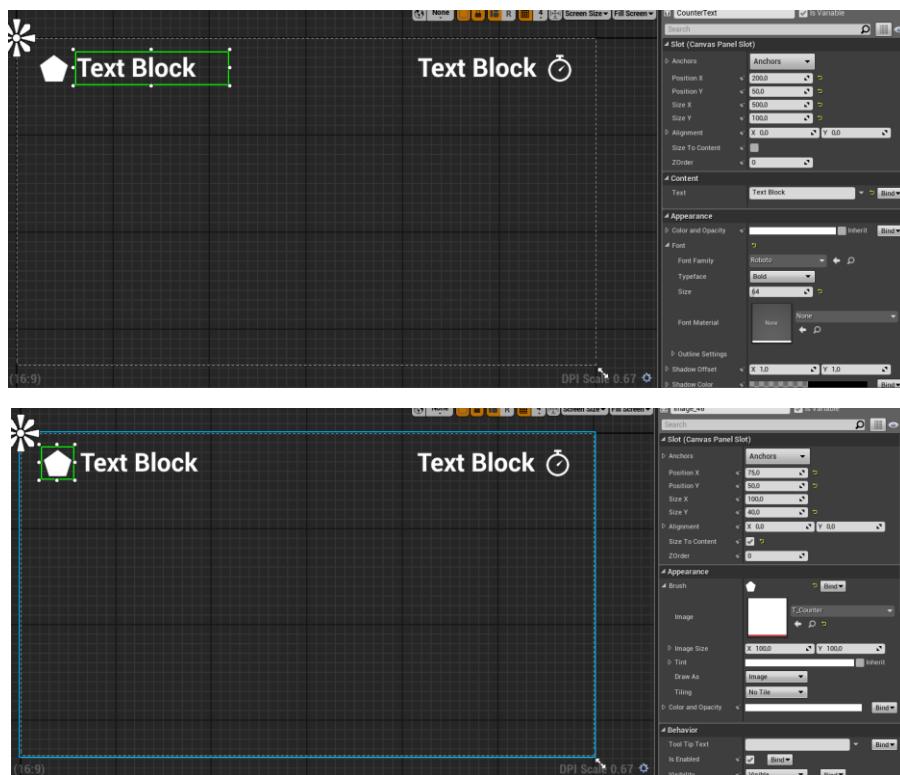


Рис. 15 Текс и картинка

## Создание Переменные-ссылки

Благодаря ссылкам можно отслеживать коробку с мячом. Таким образом, нам не придётся проверять каждую коробку.

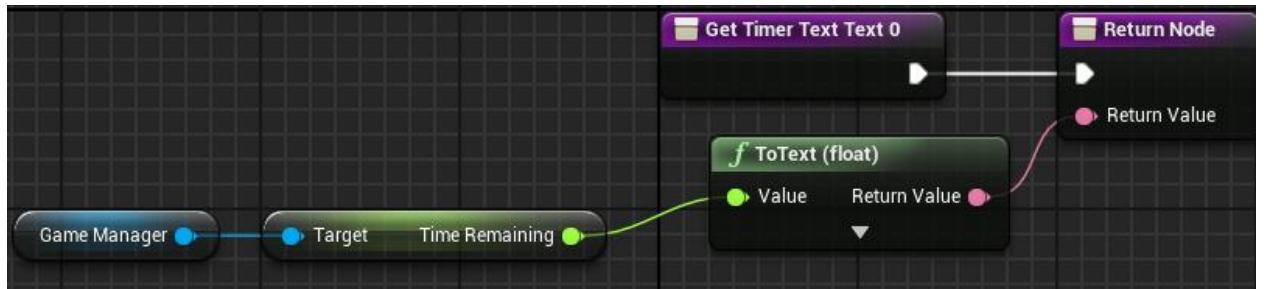


Рис. 16 Переменная-ссылка

## Задание переменной ссылки

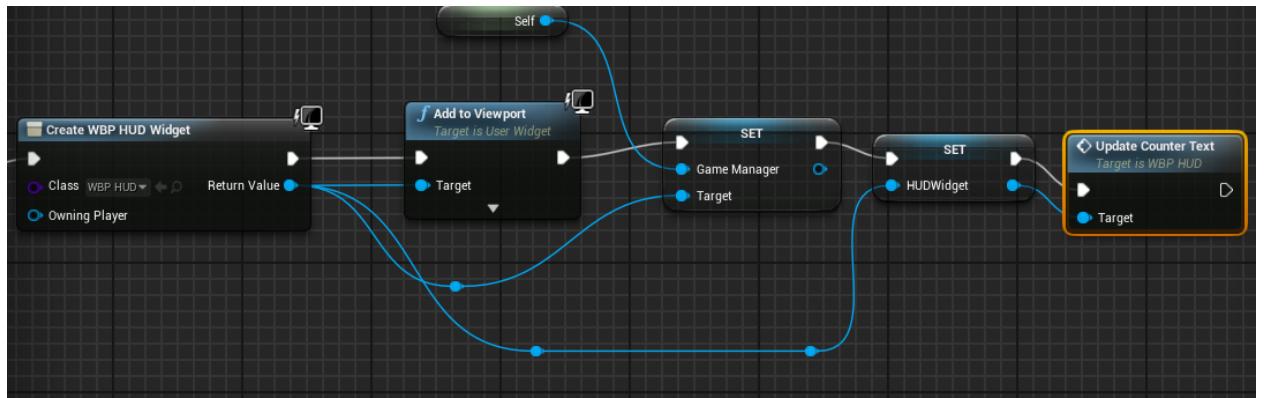


Рис. 17 Задание переменной

## Создание функции обновления

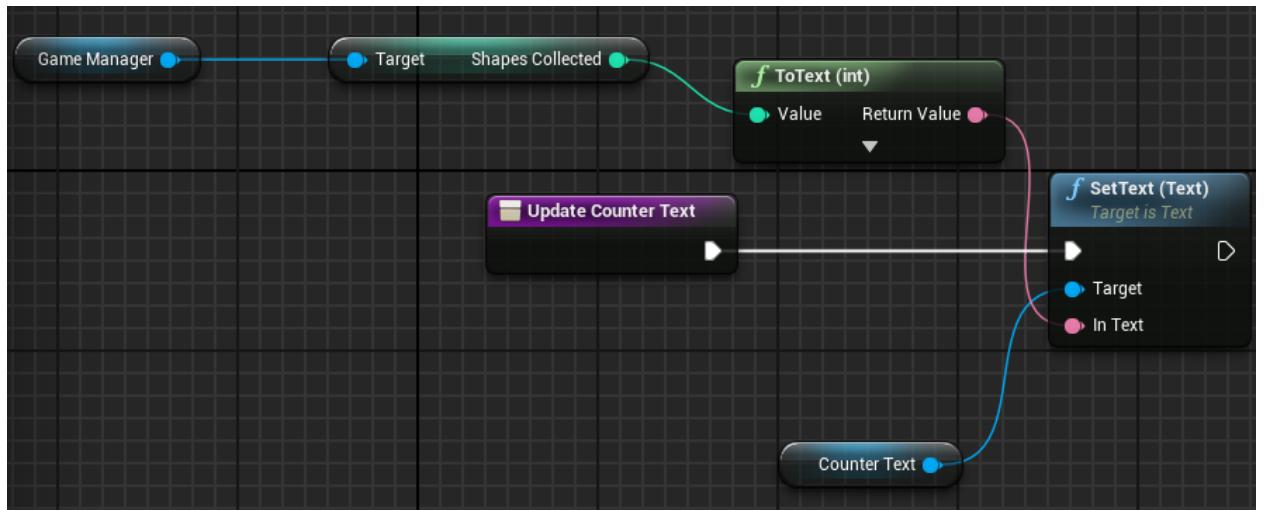


Рис. 18 Функция обновления

Порядок событий:

1. При вызове *UpdateCounterText* функция получает переменную *ShapesCollected* от *BP\_GameManager*
2. Нод *ToText (int)* преобразует значение *ShapesCollected* в тип *Text*
3. *SetText (Text)* задаёт текст *CounterText* из значения *ToText (int)*

### Вызов функции обновления

Теперь при выполнении *IncrementShapesCollected* она будет увеличивать *ShapesCollected*, а затем вызывать *UpdateCounterText*. Эта функция затем обновит *CounterText* значением *ShapesCollected*.

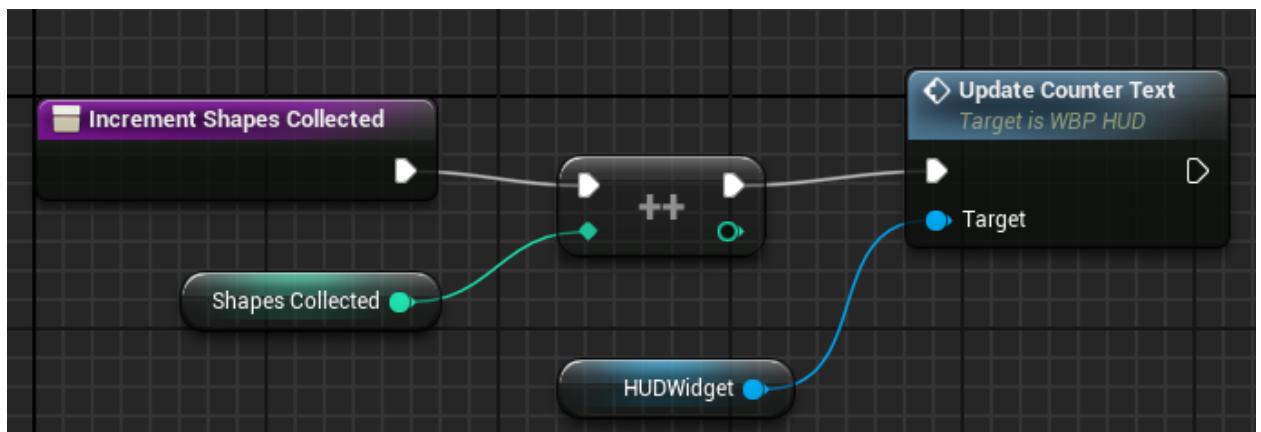


Рис. 19 Вызов функции

## 5. Как создать простую игру

### Движение игрока вперёд

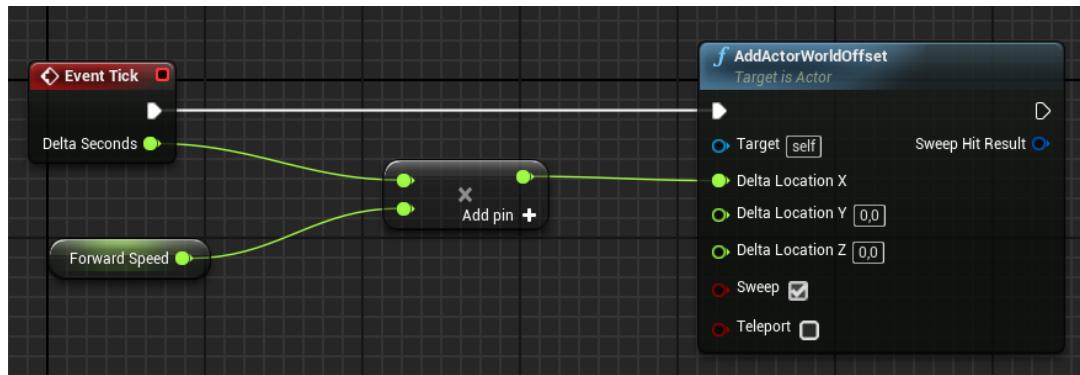


Рис. 20 Движение

Подведём итог:

1. В каждом кадре игра будет умножать *ForwardSpeed* и *Delta Seconds*, чтобы получать независимый от частоты кадров результат
2. *AddActorWorldOffset* будет использовать этот результат для перемещения игрока вдоль оси *X*
3. Поскольку *Sweep* включён, игрок будет прерывать движение вперёд, когда его что-то блокирует

### Создание системы спауна туннелей

Перейдите в Content Browser и зайдите в папку *Blueprints*. Создайте новый *Blueprint Class* с родительским классом *Actor*. Назовите его *BP\_TunnelSpawner* и откройте его

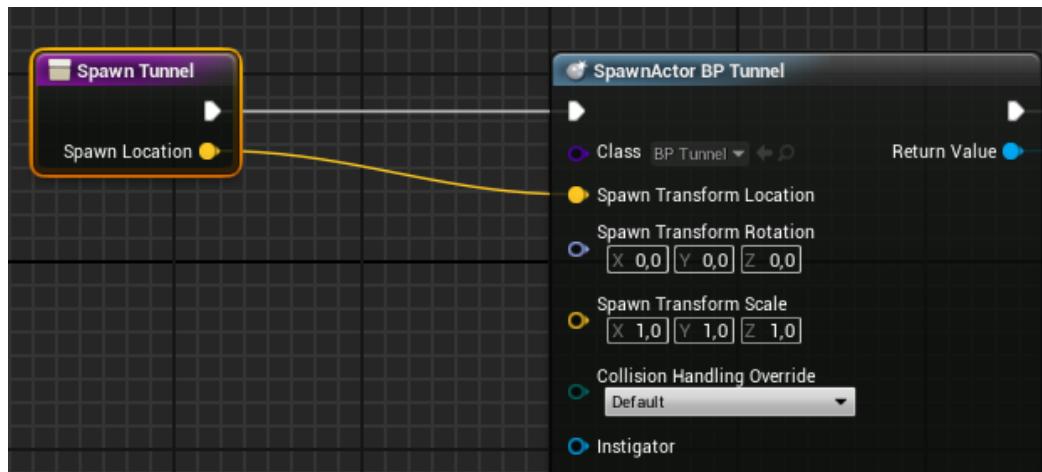


Рис. 21 Система спауна

## Система Спауна тунелей

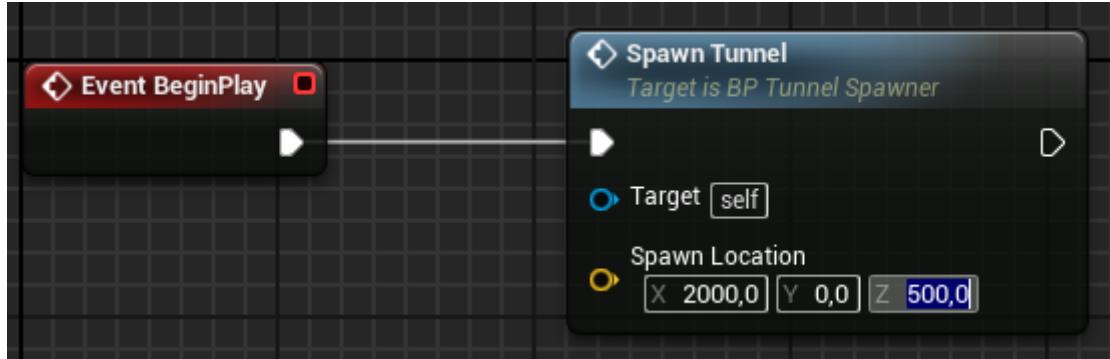


Рис. 22 Событие спауна

## Настройка Blueprint туннеля

### Создание зоны триггера

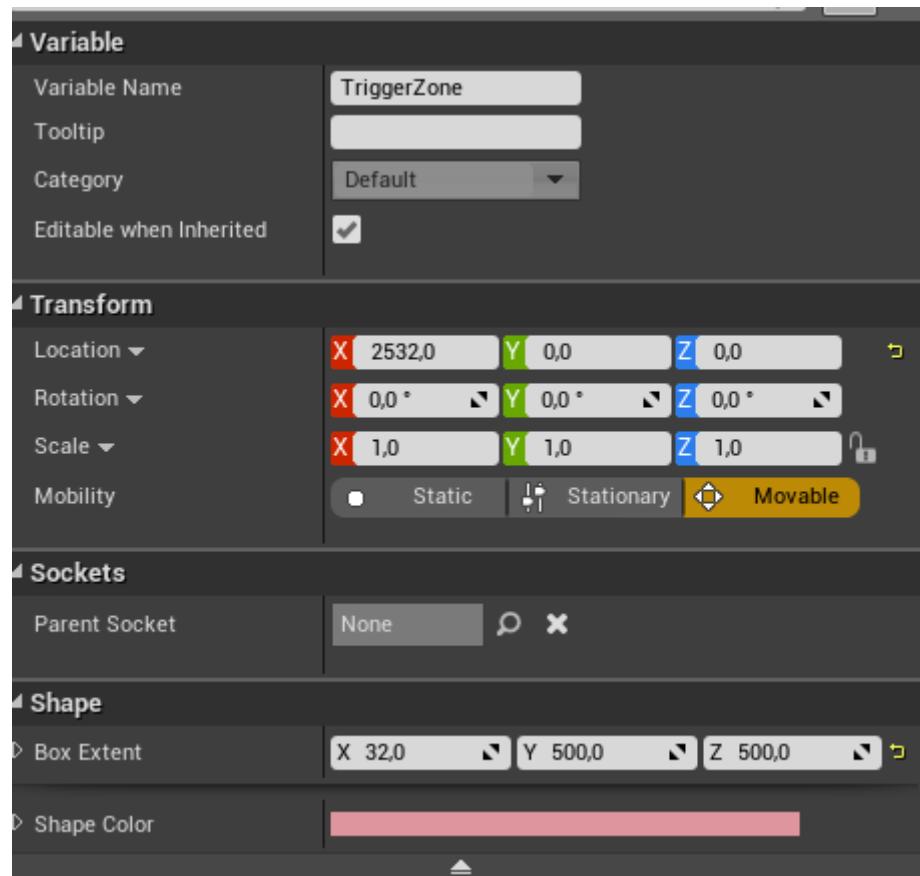


Рис. 23 зона триггера

## Создание точки спавна

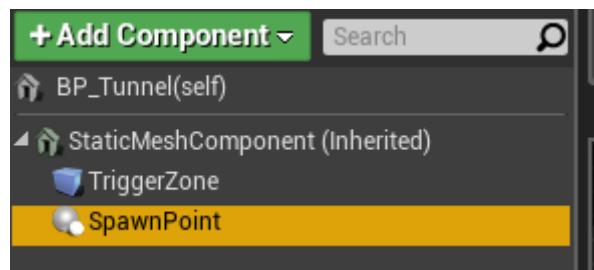


Рис. 24 точка спавна

## Создание туннеля в точке спавна

Эта схема будет получать самый новый туннель и местоположение его компонента *SpawnPoint*, после чего спаунить новый туннель в этом местоположении.

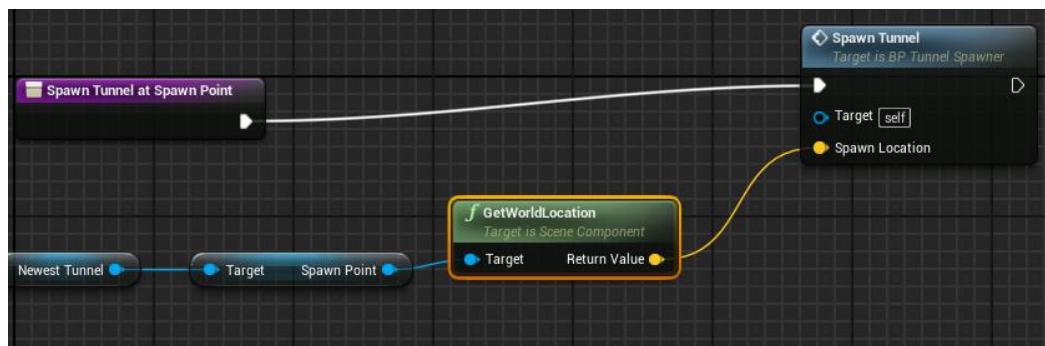


Рис. 25 ТунNELь

## Создание ссылки на спаунер

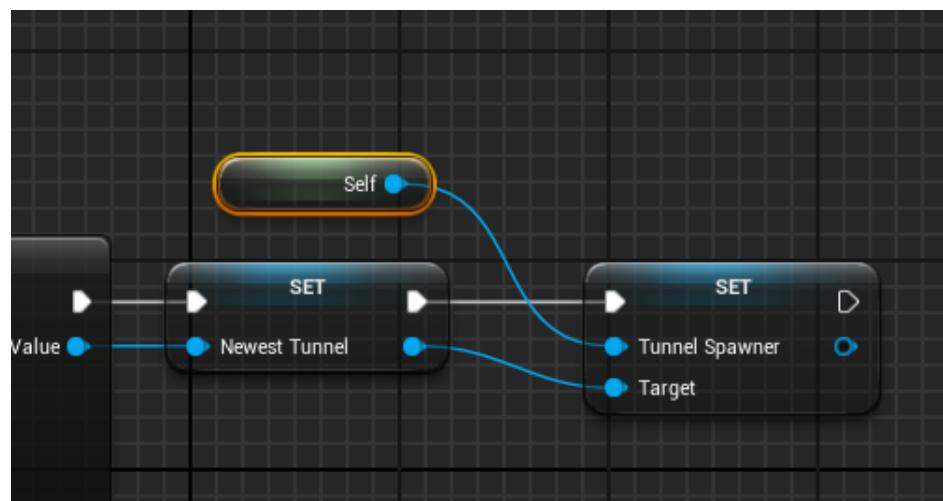


Рис. 26 ссылка спауна

## Скрипting зоны триггера

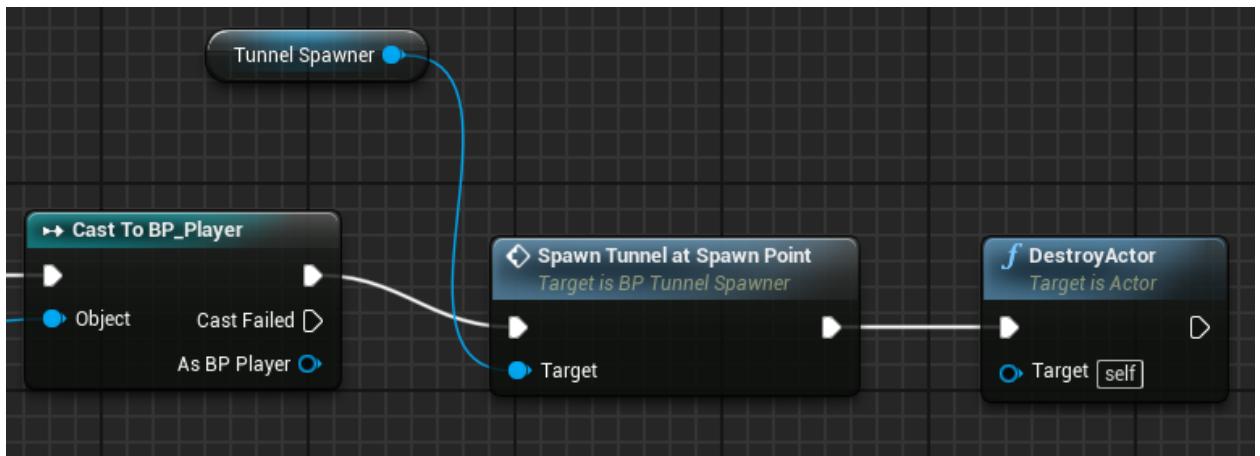


Рис. 27 Скрипт триггера

Давайте посмотрим, что здесь происходит пошагово:

1. Когда *Actor* касается *TriggerZone*, выполняется под *On Component Begin Overlap (TriggerZone)*
2. Под *Cast to BP\_Player* проверяет, является ли касающийся актор игроком
3. Если это игрок, то *BP\_TunnelSpawner* создаст новый туннель. Его местоположение будет находиться в компоненте *SpawnPoint* последнего созданного туннеля.
4. Поскольку старый туннель уже не нужен, игра удаляет его с помощью нода *DestroyActor*

## Создание нескольких туннелей

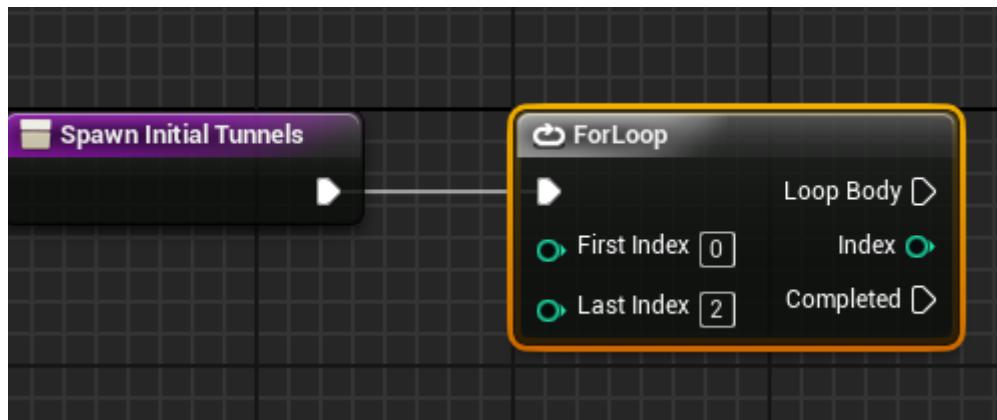


Рис. 28 Туннели

## Создание первого туннеля

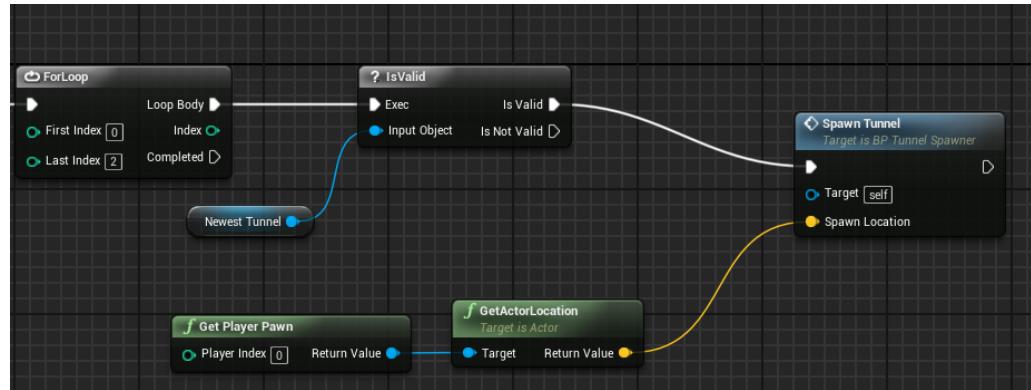


Рис. 29 Создание туннеля

## Создание следующего туннеля

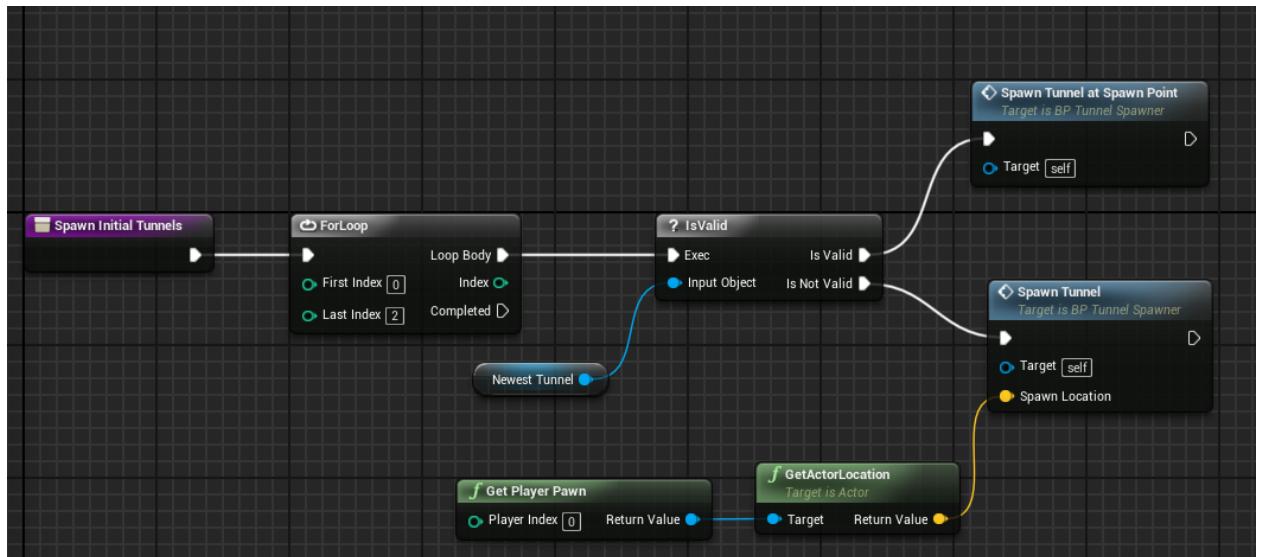


Рис. 30 Создание туннелей

Подведём итог:

1. Нод *ForLoop* выполняется три раза
2. В первом цикле он спаунит туннель в точке расположения игрока
3. В последующих циклах он спаунит туннель в точке *SpawnPoint* самого нового туннеля

## Создание препятствий

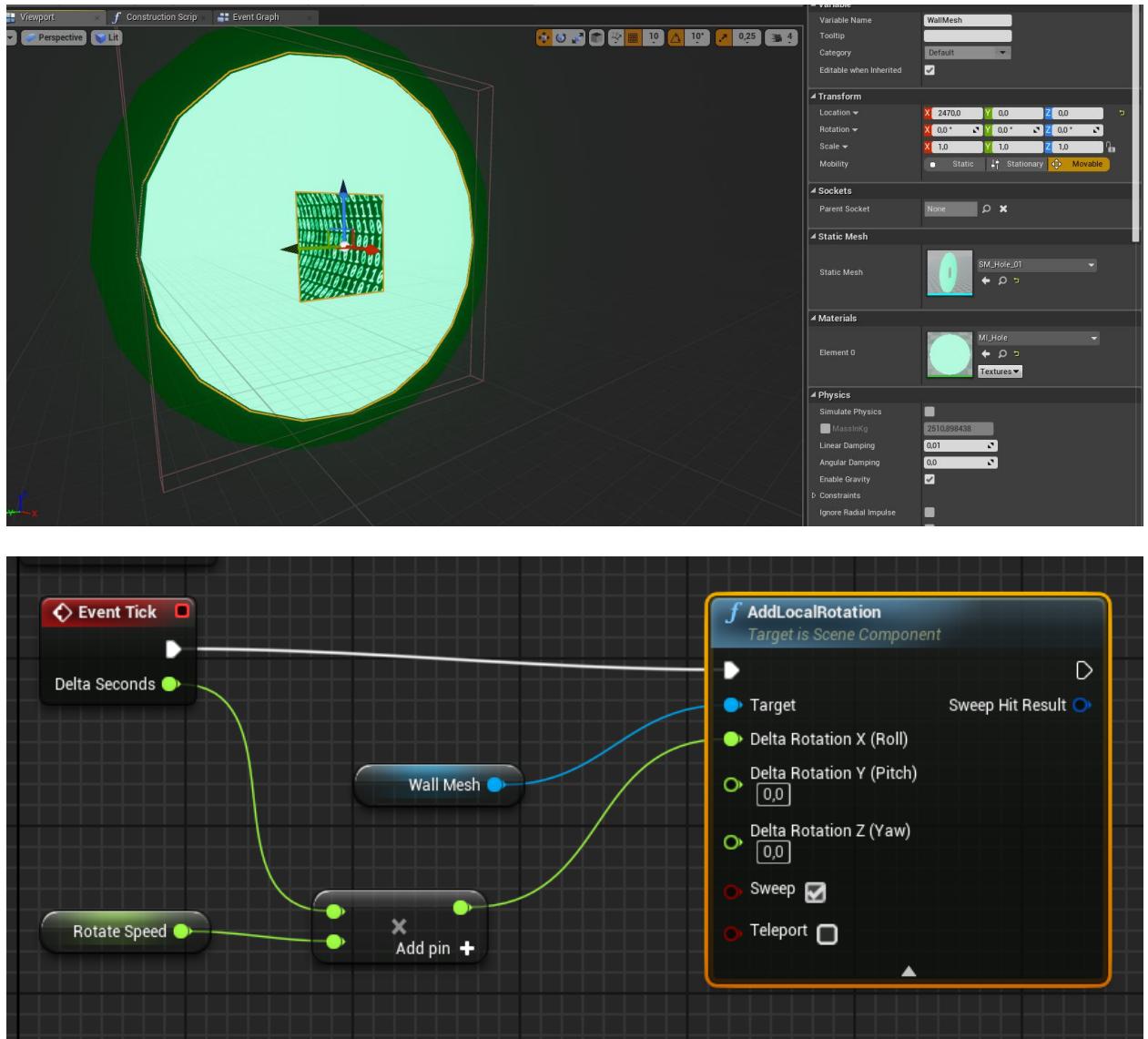


Рис. 31 Реализация препятствий

Чтобы сделать игру интереснее, стены должны ещё и вращаться. Добавим новую переменную *Float* и назовём её *RotateSpeed*. Зададим *Default Value* значение *30*.

## Создание вариации стен

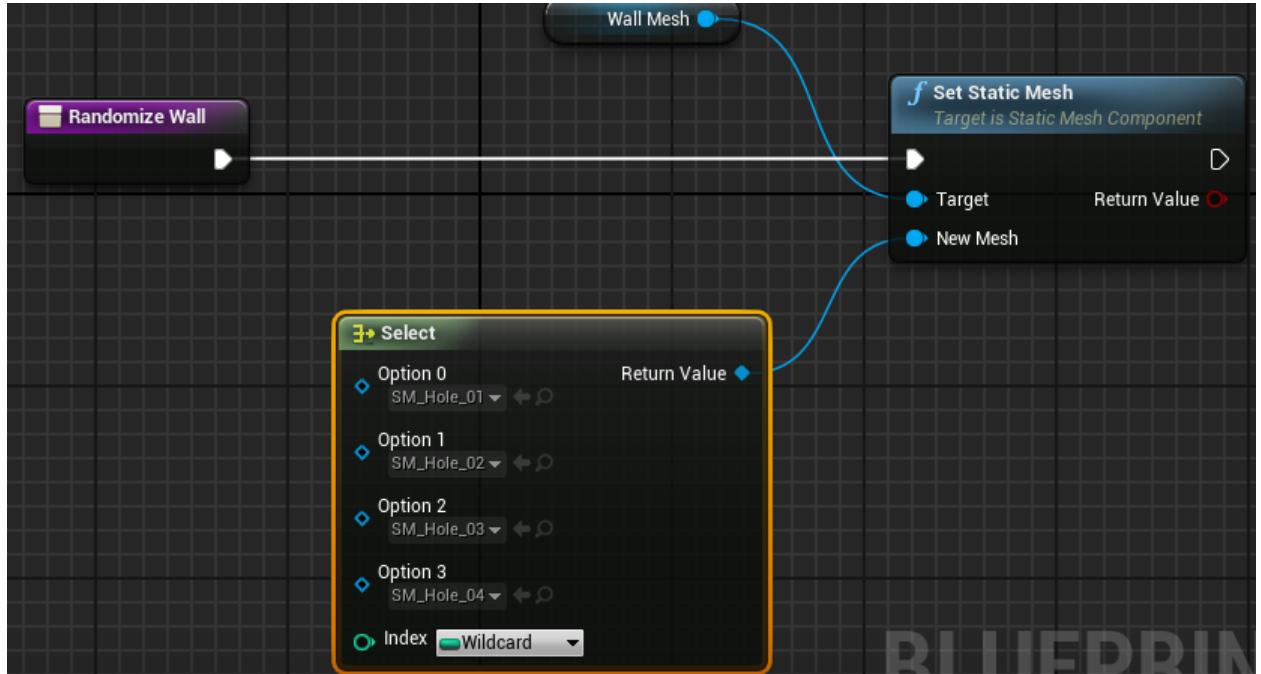


Рис. 32 Выбор стен

## Рандомизация стен

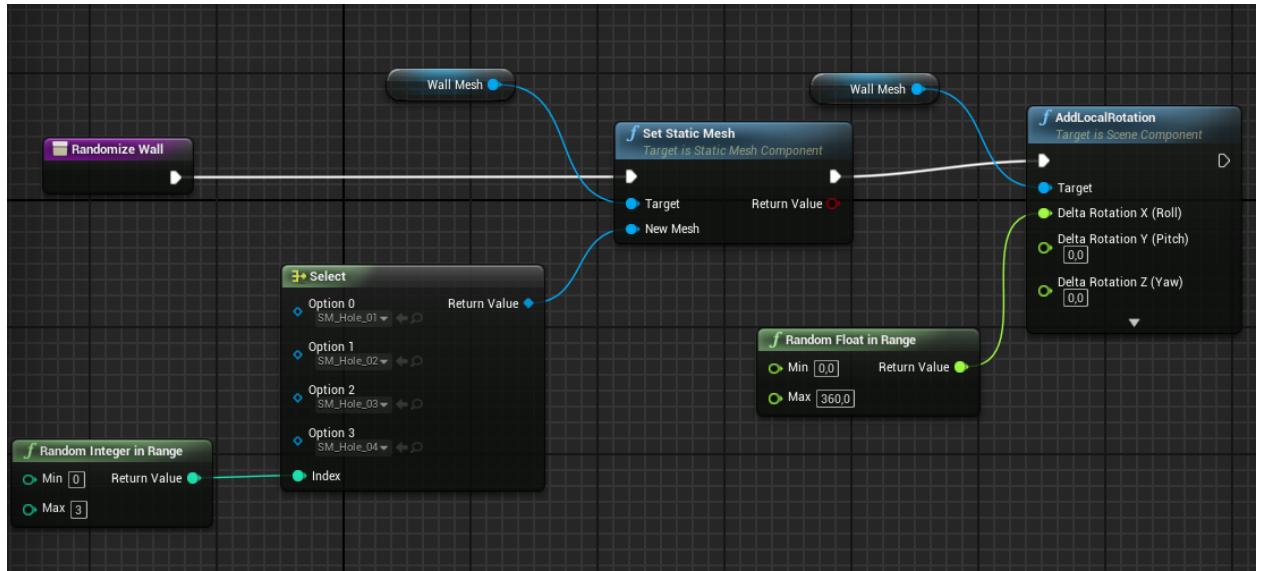


Рис. 33 Рандомизация

Подведём итог:

1. Нод *Select* передаёт список мешей
2. С помощью нода *Random Integer in Range* выбирается случайный меш
3. Нод *Set Static Mesh* задаёт *WallMesh* выбранный меш
4. Нод *AddLocalRotation* добавляет смещение случайного поворота *WallMesh*

## Обработка коллизий со стенами

### Остановка при смерти

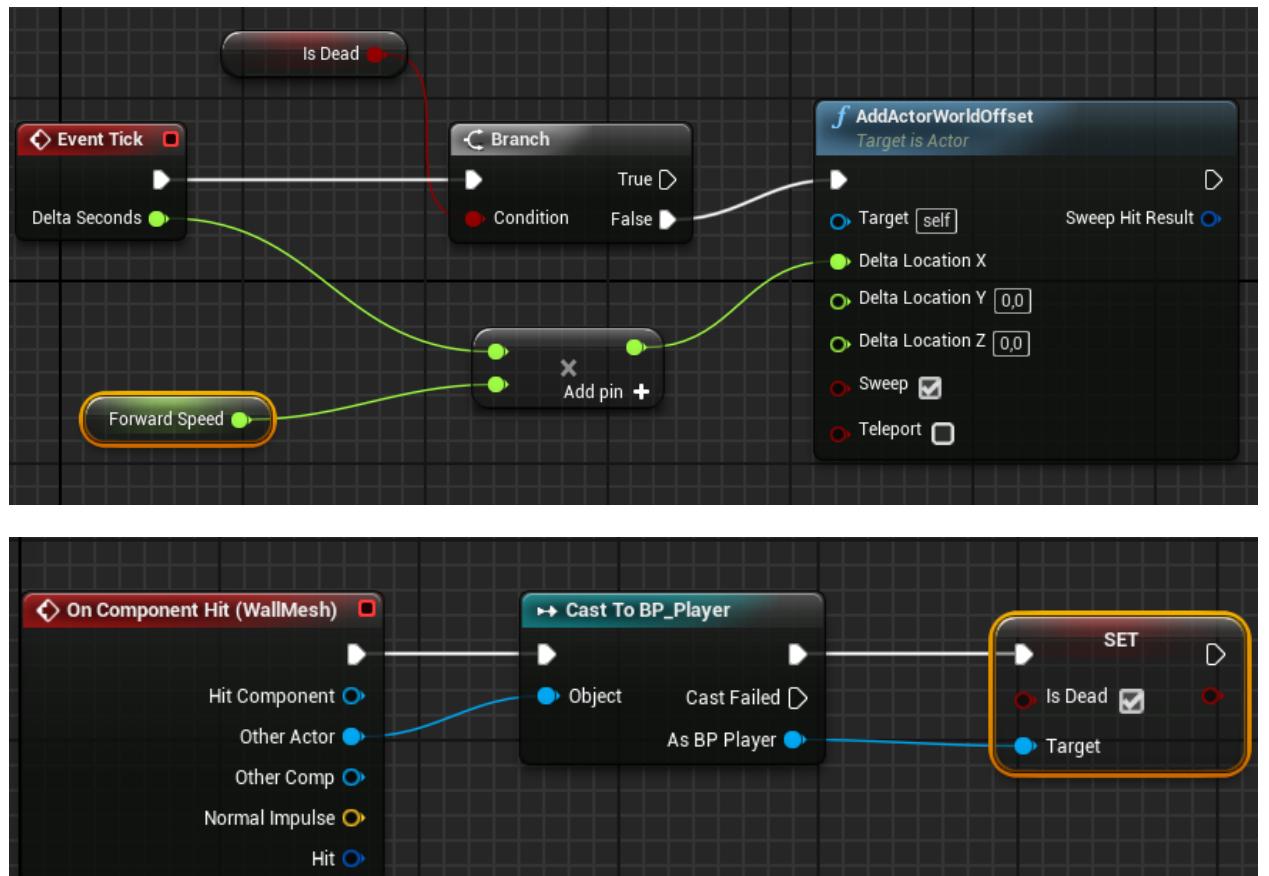


Рис. 34-35 Действия при смерти

## Отображение кнопки перезагрузки

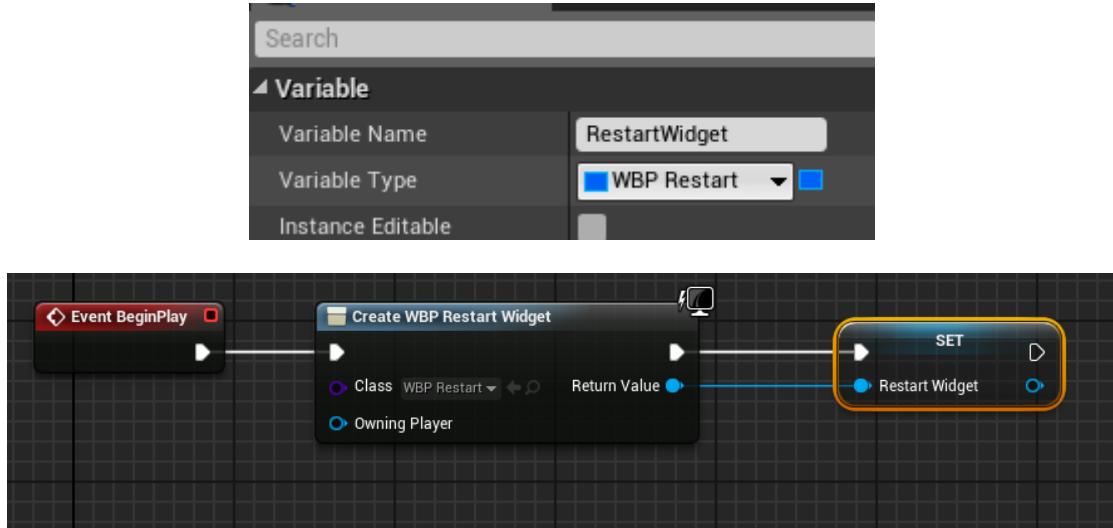


Рис.36-37 Кнопка перезагрузки

Теперь при спауне игрока мы будем создавать экземпляр *WBP\_Restart*. Следующим шагом будет создание функции, отображающей этот экземпляр.

## Создание функции отображения

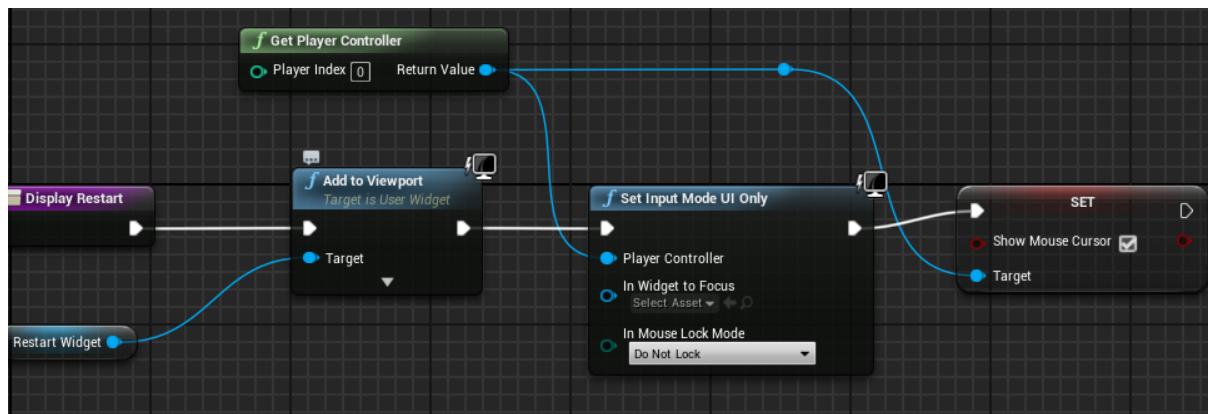


Рис. 38 Отображение

Подведём итог:

1. *Add to Viewport* отображает *RestartWidget* на экране
2. *Set Input Mode UI Only* позволяет игроку взаимодействовать только с UI. Мы делаем так, чтобы игрок не мог перемещаться, пока он мёртв.
3. Как можно понять из названия, *Set Show Mouse Cursor* просто отображает курсор мыши

## Вызов функции отображения

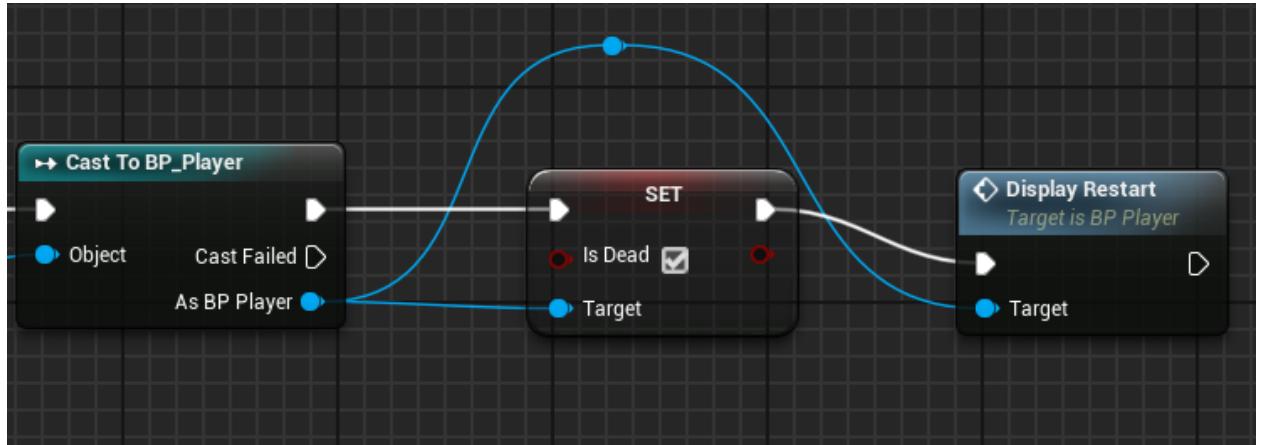


Рис. 39 Вызов

## Перезапуск игры

### Сброс игрока

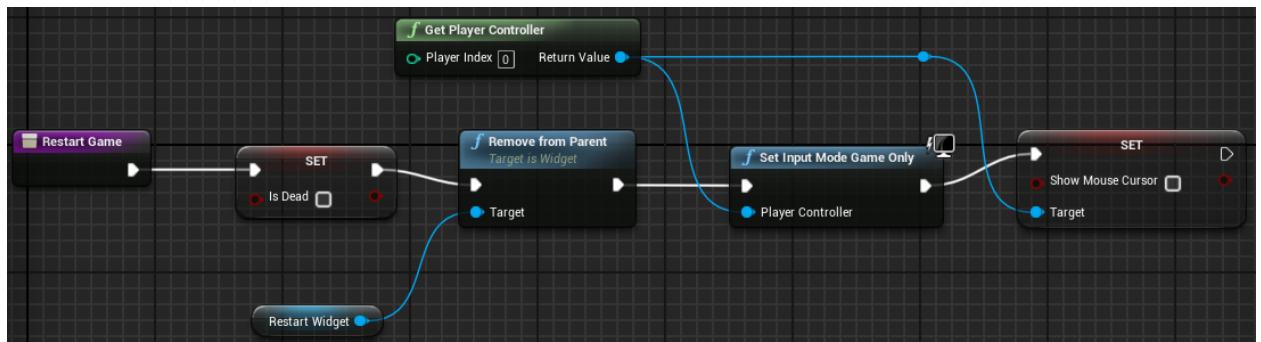


Рис. 40 Рестарт

Подведём итог:

1. Set *Is Dead* присваивает *IsDead* значение *false*. Это снова включает возможность движения вперёд.
2. Remove From Parent удаляет с экрана *RestartWidget*
3. Set Input Mode Game Only снова включает возможность управления в игре, чтобы игрок мог перемещаться
4. Set Show Mouse Cursor скрывает курсор мыши

## Повторное создание туннеля

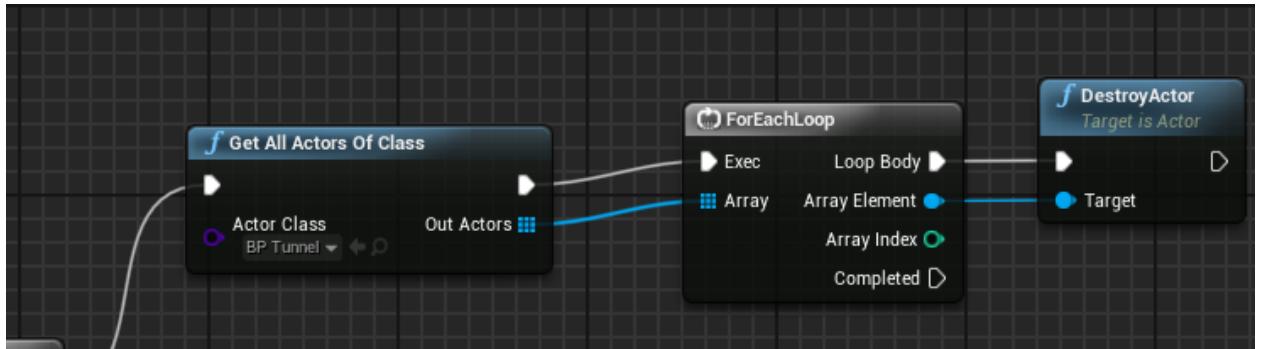


Рис. 41 Новые тунNELи

## Обработка нажатий на кнопку

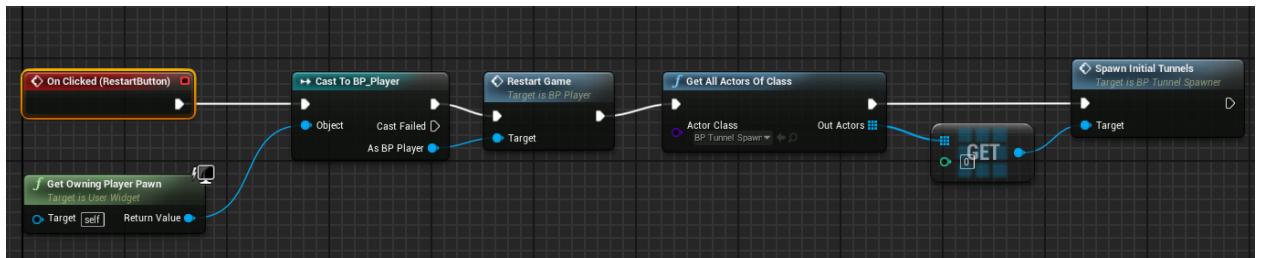


Рис. 42 Нажатие

Подведём итог:

1. *Get Owning Player Pawn* возвращает Pawn, которым в текущий момент управляет игрок
2. *Cast to BP\_Player* проверяет, принадлежит ли Pawn к классу *BP\_Player*
3. Если это так, то он вызывает функцию *RestartGame*. Эта функция сбрасывает игрока и скрывает кнопку перезапуска.
4. *Get All Actors of Class* и *Get* возвращают *BP\_TunnelSpawner* и вызывает *SpawnInitialTunnels*. Эта функция удаляет все существующие тунNELи и спаунит новые.

## 6. Анимация

### Импорт анимации



Рис. 43 Импорт

### Использование Skeletal Mesh

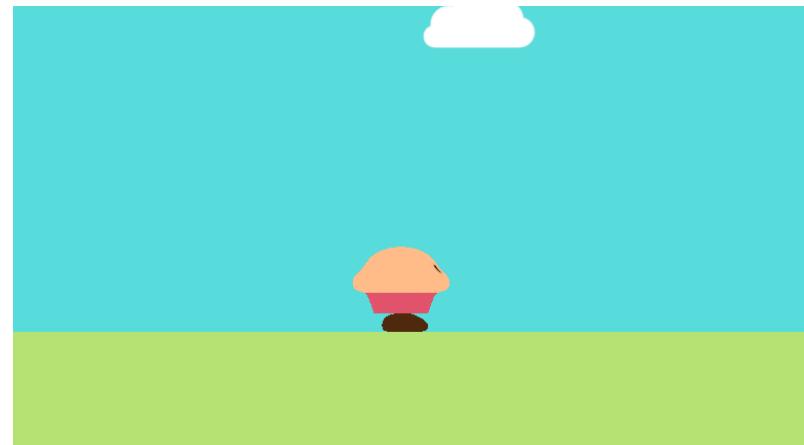


Рис. 44-45 Скелет

## Создание Animation Blueprint

Animation Blueprint похож на обычный Blueprint. Однако в нём также есть граф, предназначенный только для задач анимации.

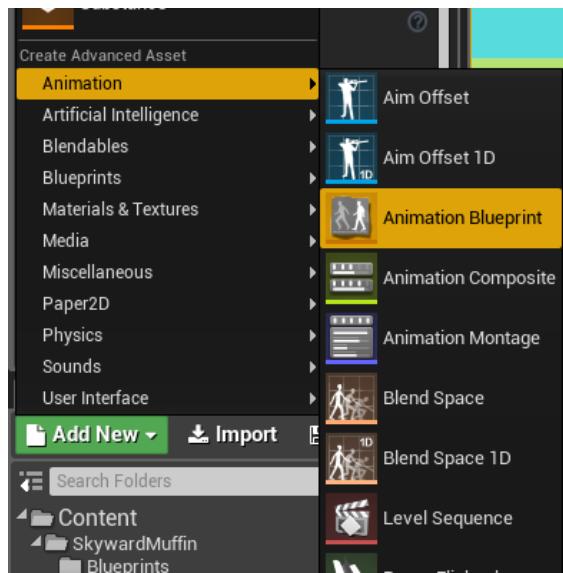


Рис. 46 Создание

## Создание конечного автомата

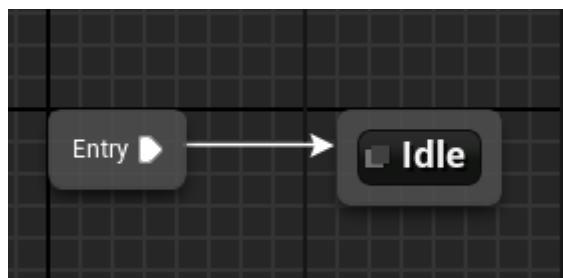


Рис. 47 Конечный автомат

## Привязывание анимации к состоянию

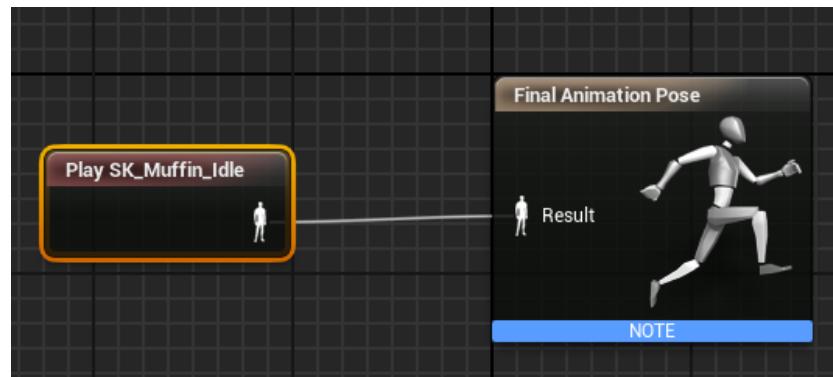


Рис. 48 Привязка анимации

## Использование Animation Blueprint

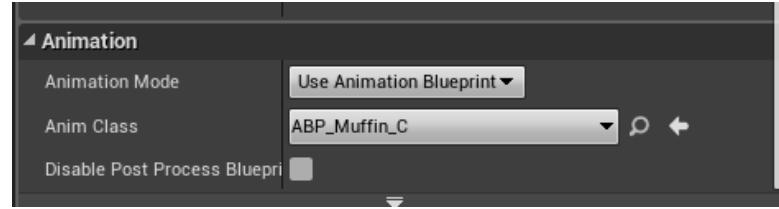


Рис. 49 Использование

## Создание анимации прыжка и падения

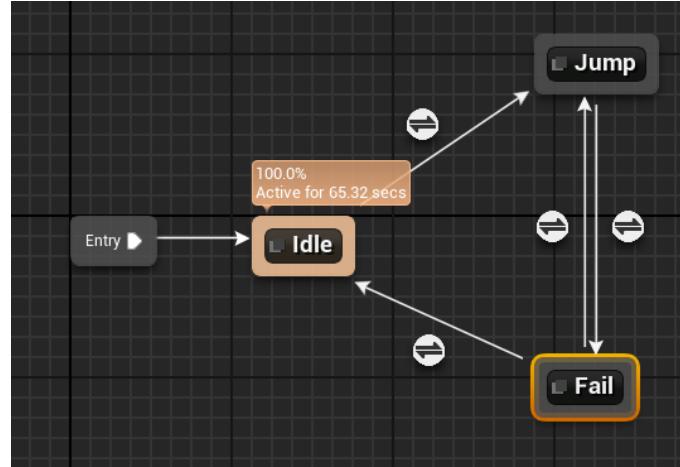


Рис. 50 Анимация

Определяем прыгает персонаж или падает

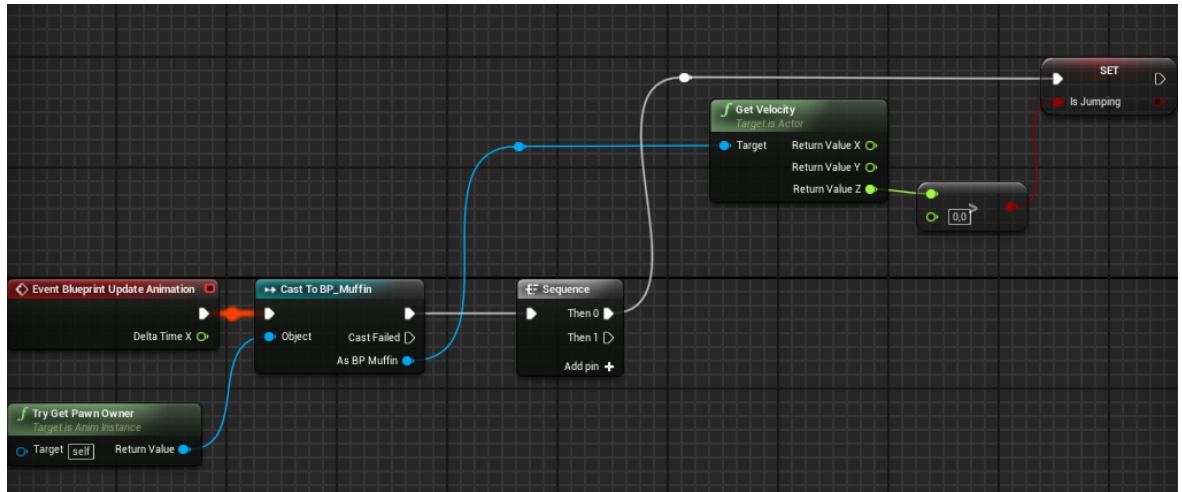


Рис. 51 Условия анимации

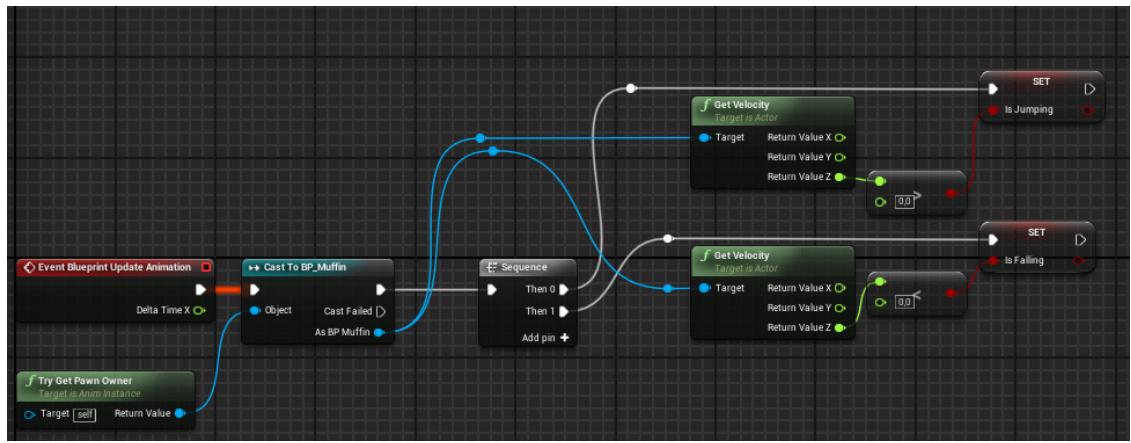


Рис. 52 Условия анимации

### Определение правил перехода

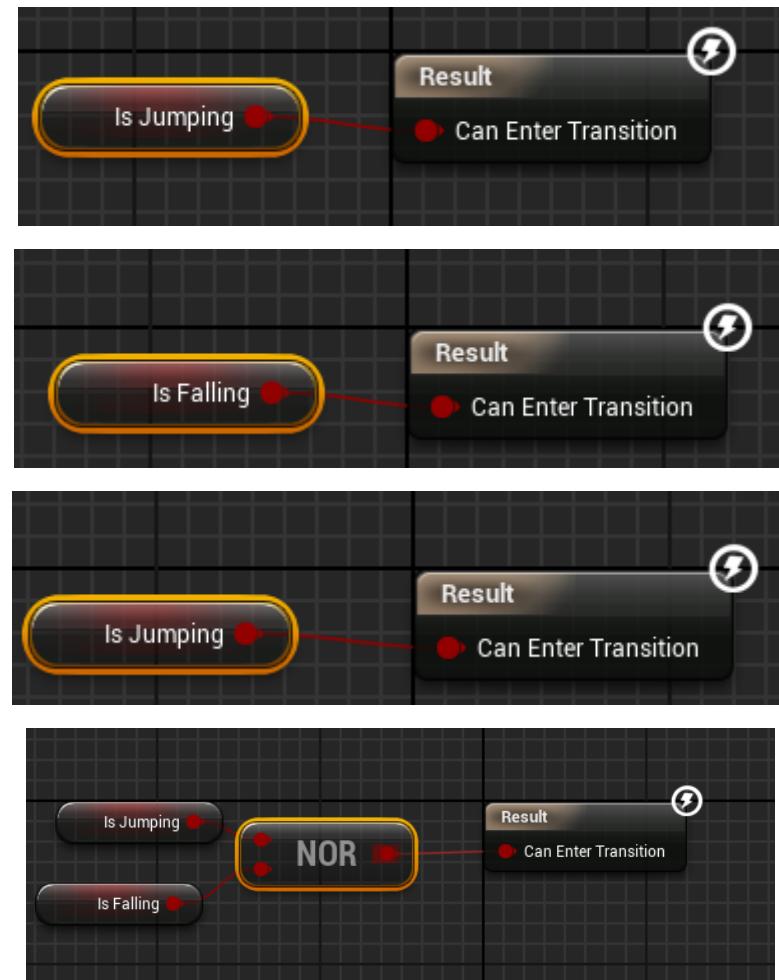


Рис. 53-56 Правила перехода

## Создание Blend Space

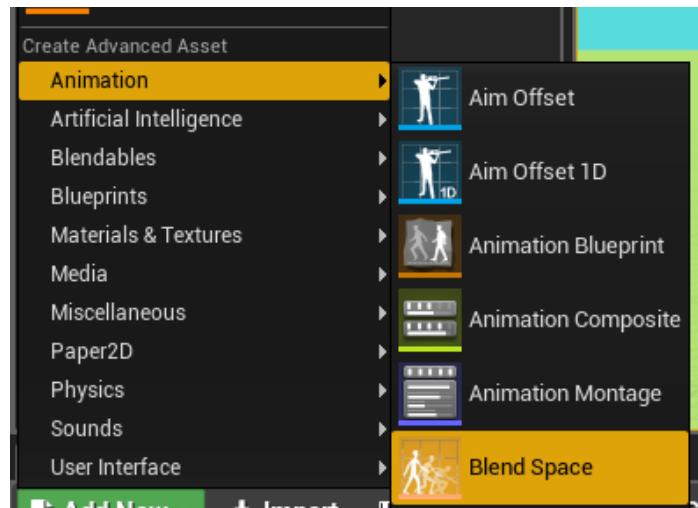


Рис. 57 Создание

## Добавление анимации в Blend Space

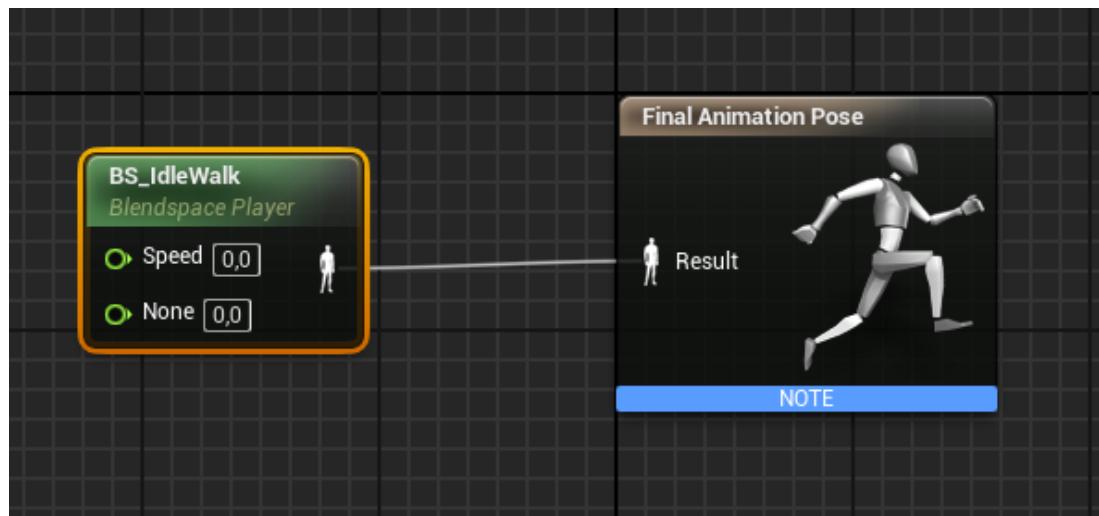


Рис. 58 Анимация

## Получение скорости игрока

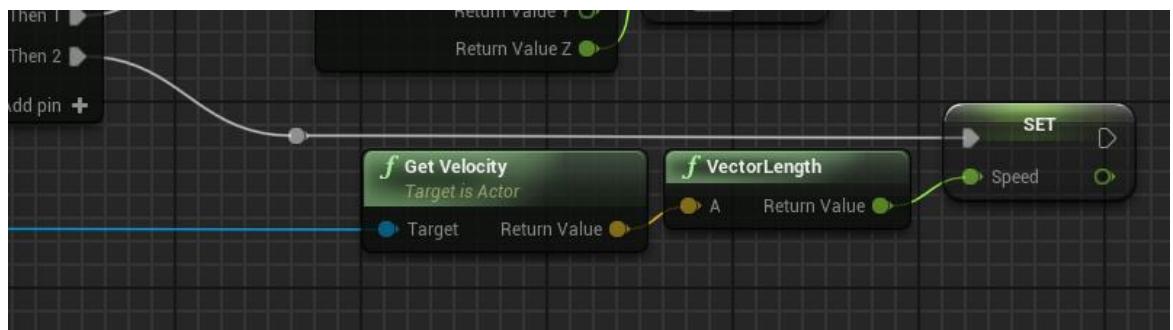


Рис. 59 Скорость игрока

## Использование анимации смерти

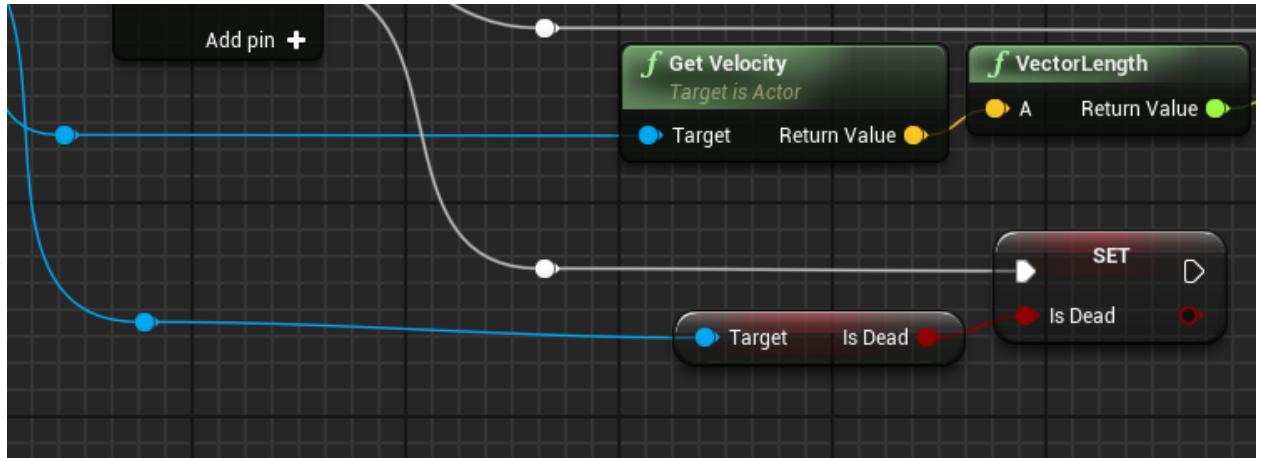


Рис. 60 Анимация смерти

## Использование узла Blend Poses by bool

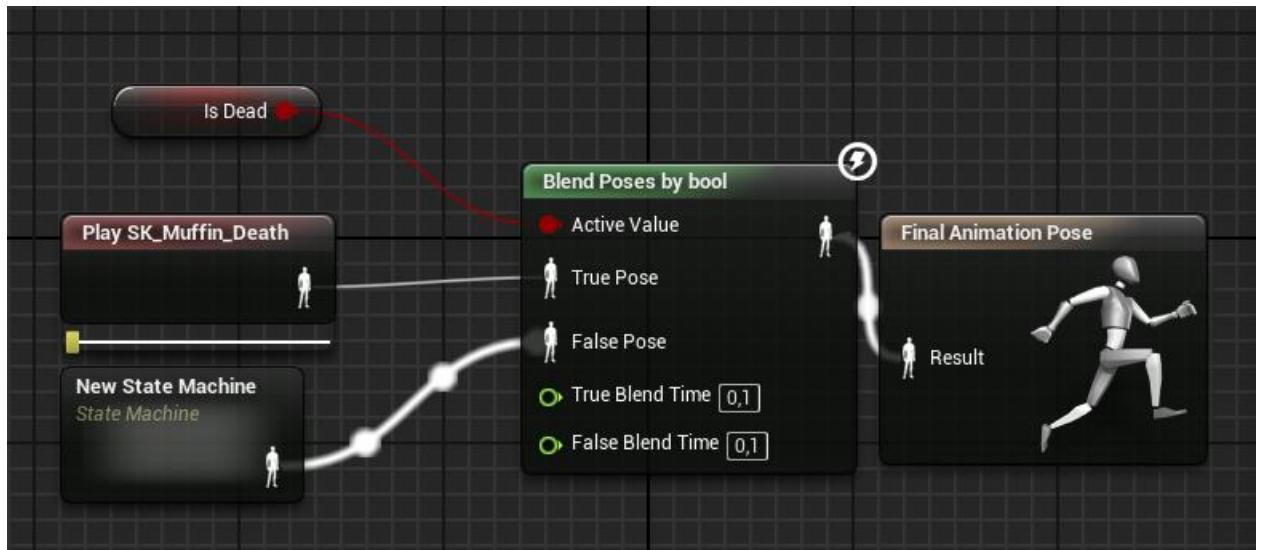


Рис. 61 Полная анимация

Теперь если переменная *IsDead* будет равна *true*, то начнёт воспроизводиться анимация смерти. Если *IsDead* равно *false*, то будет воспроизводиться текущая анимация конечного автомата *Locomotion*.

## 7. Звук

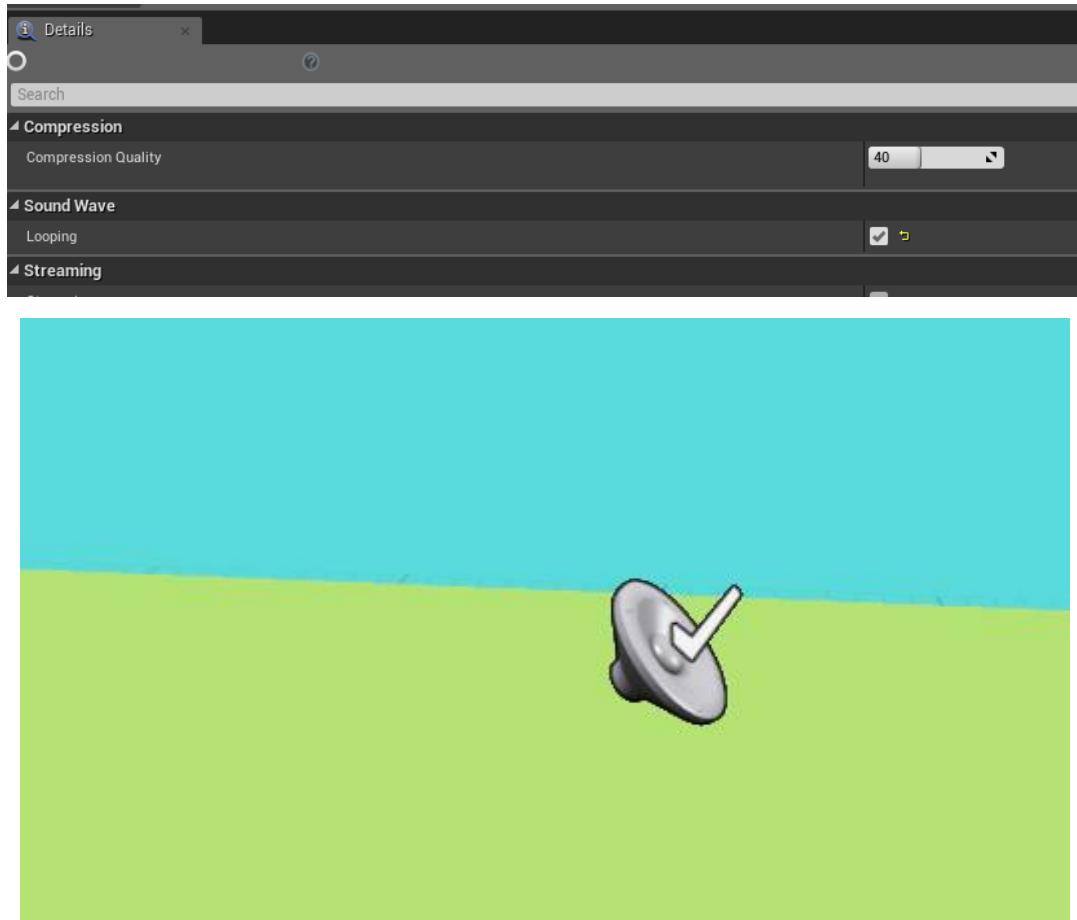


Рис. 62-63 Звук на сцене

## Animation Notify

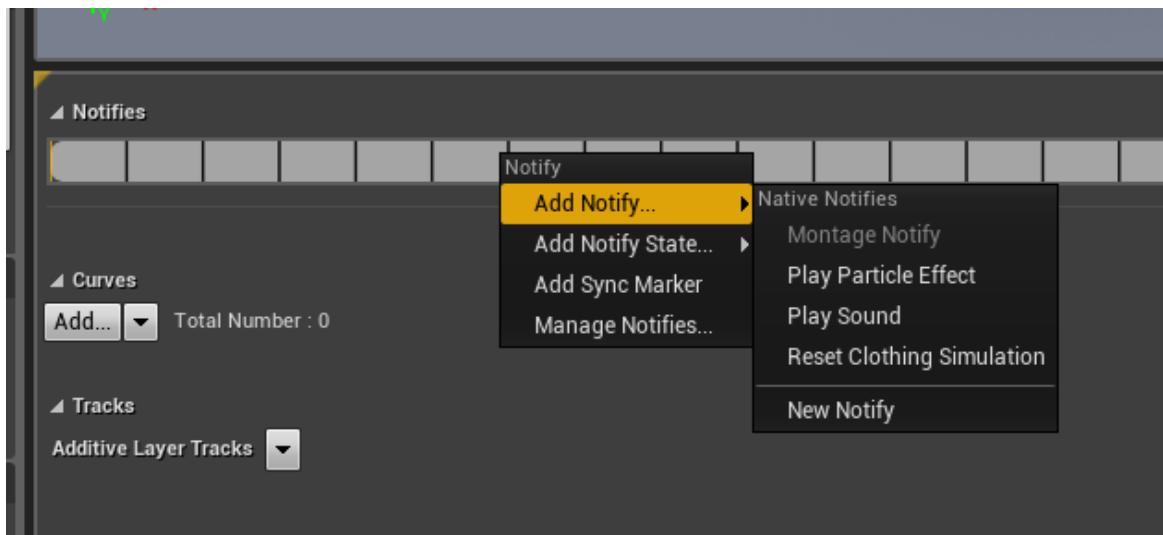


Рис. 64 Создание

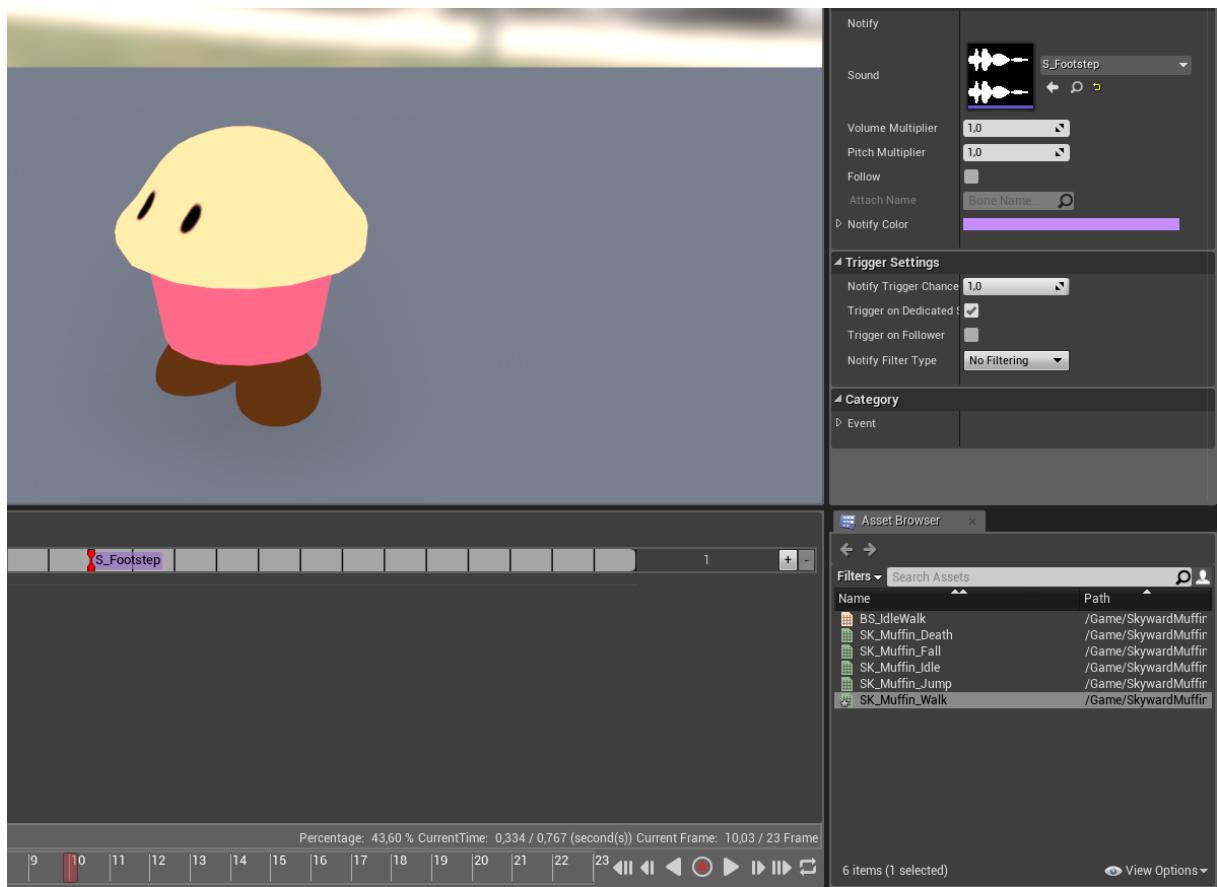


Рис. 65 Перетаскивание

## Sound Cue

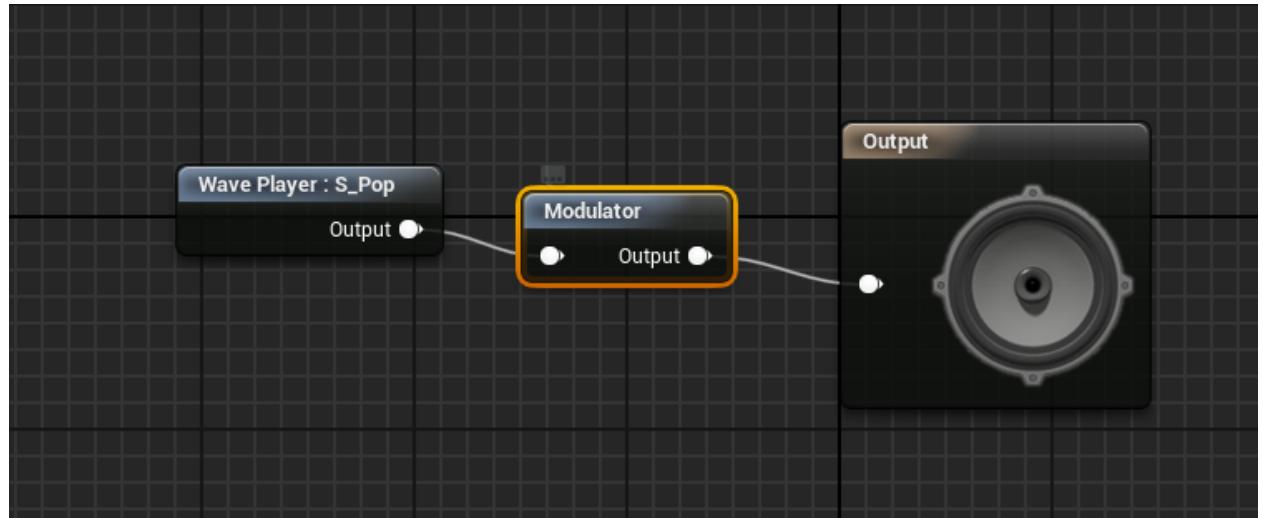


Рис. 66 Модулятор

## Воспроизведение Sound Cue

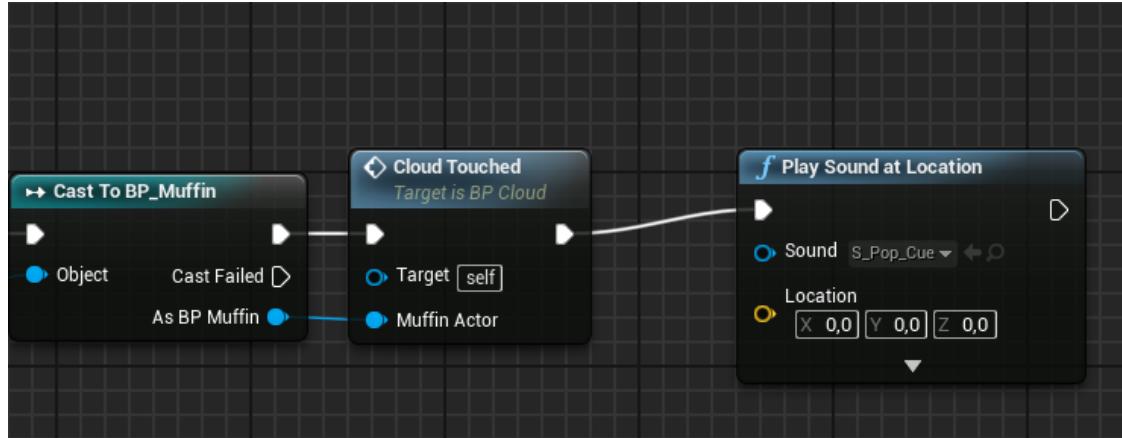


Рис. 67 Воспроизведение

## Воспроизведение звука в 3-Д пространстве

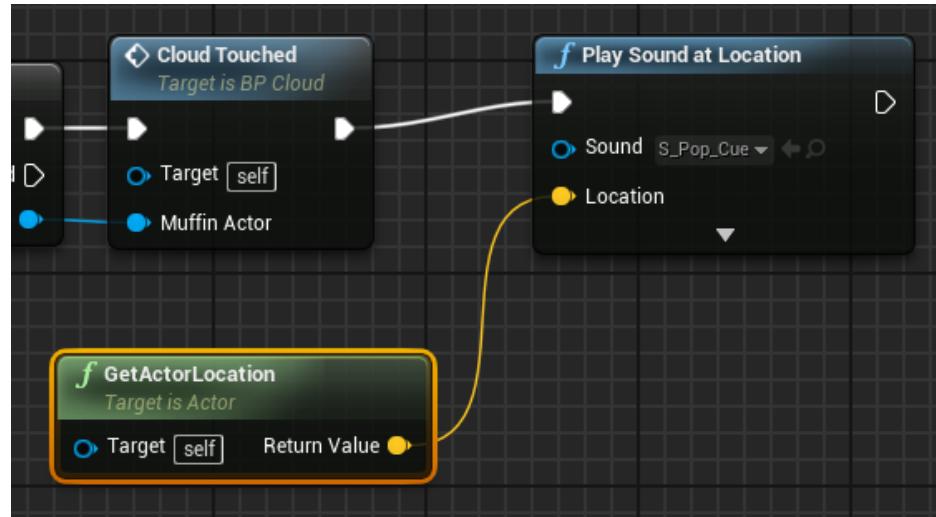


Рис. 68 Эффект 3-D

## Добавление звука дождя

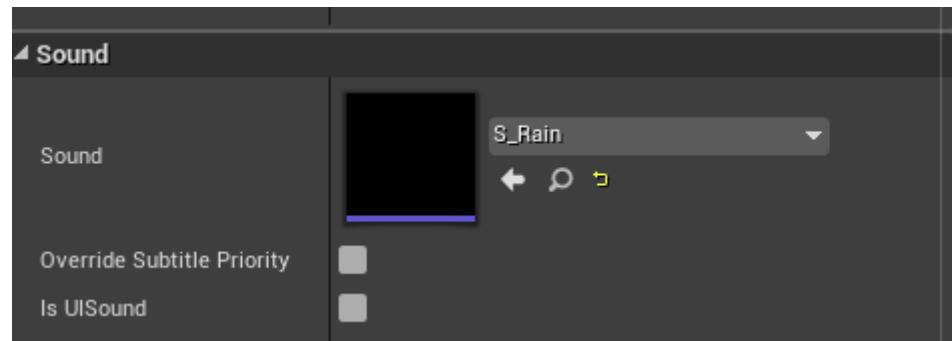


Рис. 69 Добавление звука

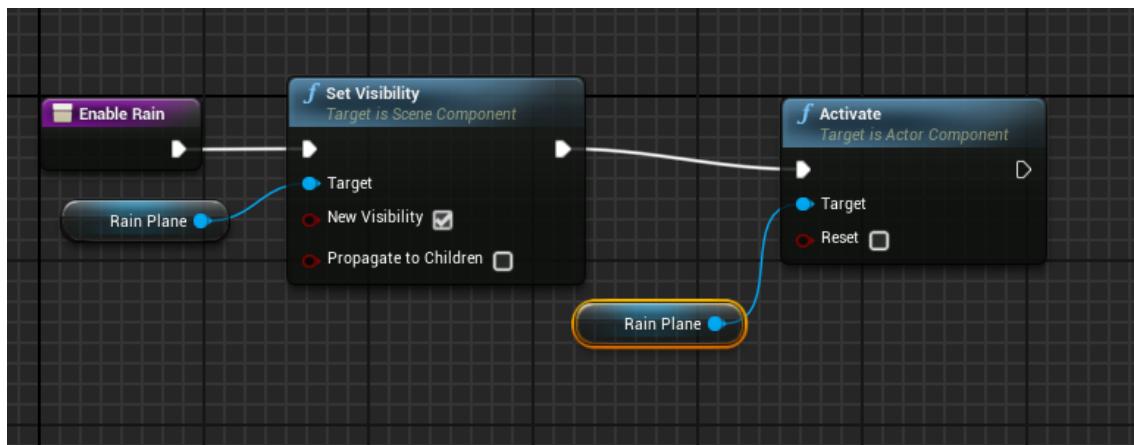


Рис. 70 активация добавления

## Настройка затухания

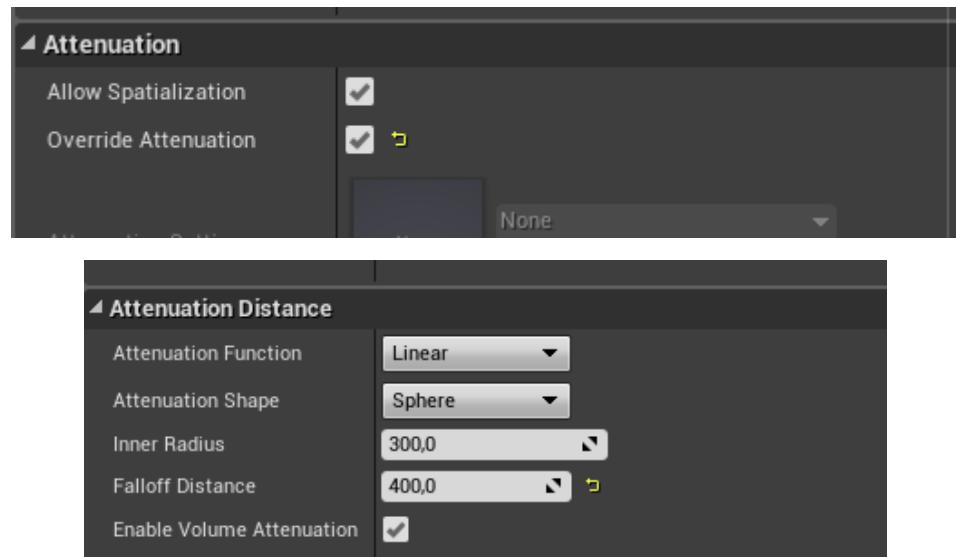


Рис. 71-72 Затухание

## Плавное снижение громкости звука

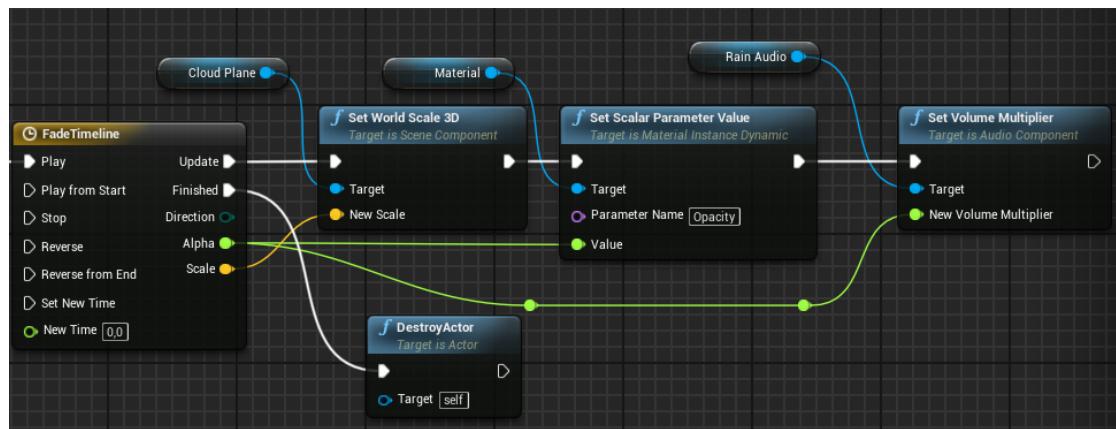


Рис. 73 Снижение громкости

## Создание классов звуков

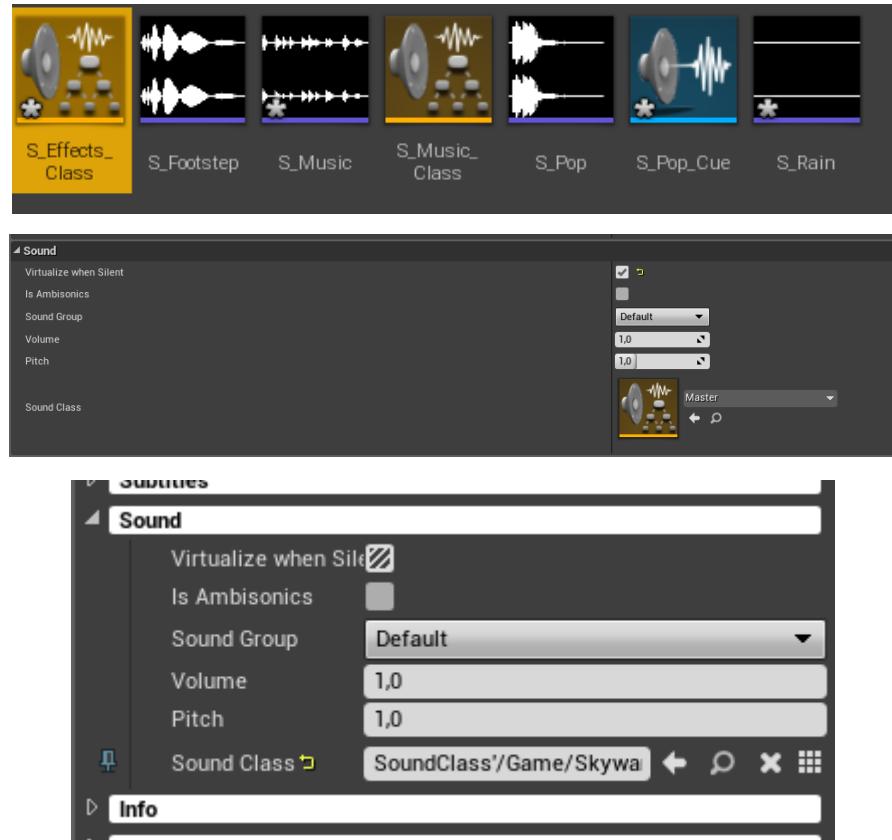


Рис.74-76 Создание и реализация классов

## Создание и настройка Sound Mix

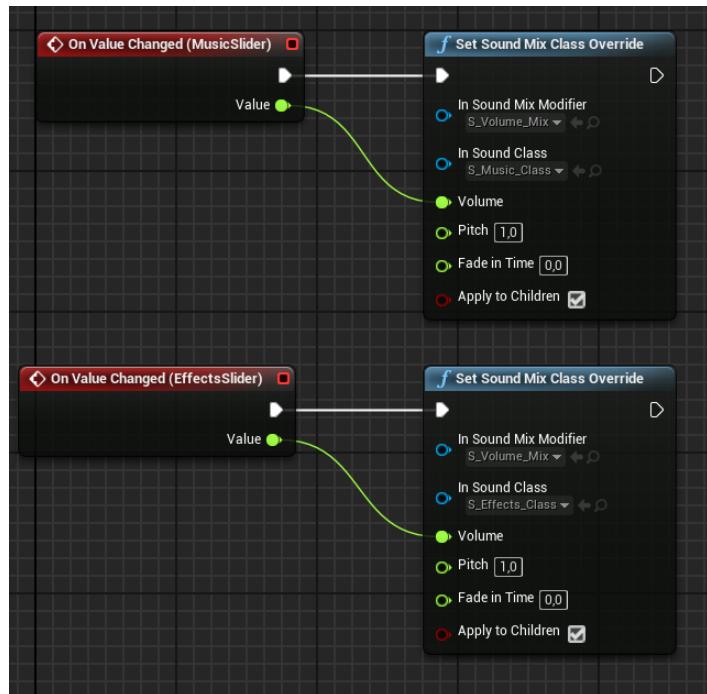


Рис. 77 Создание

## Активация Sound Mix

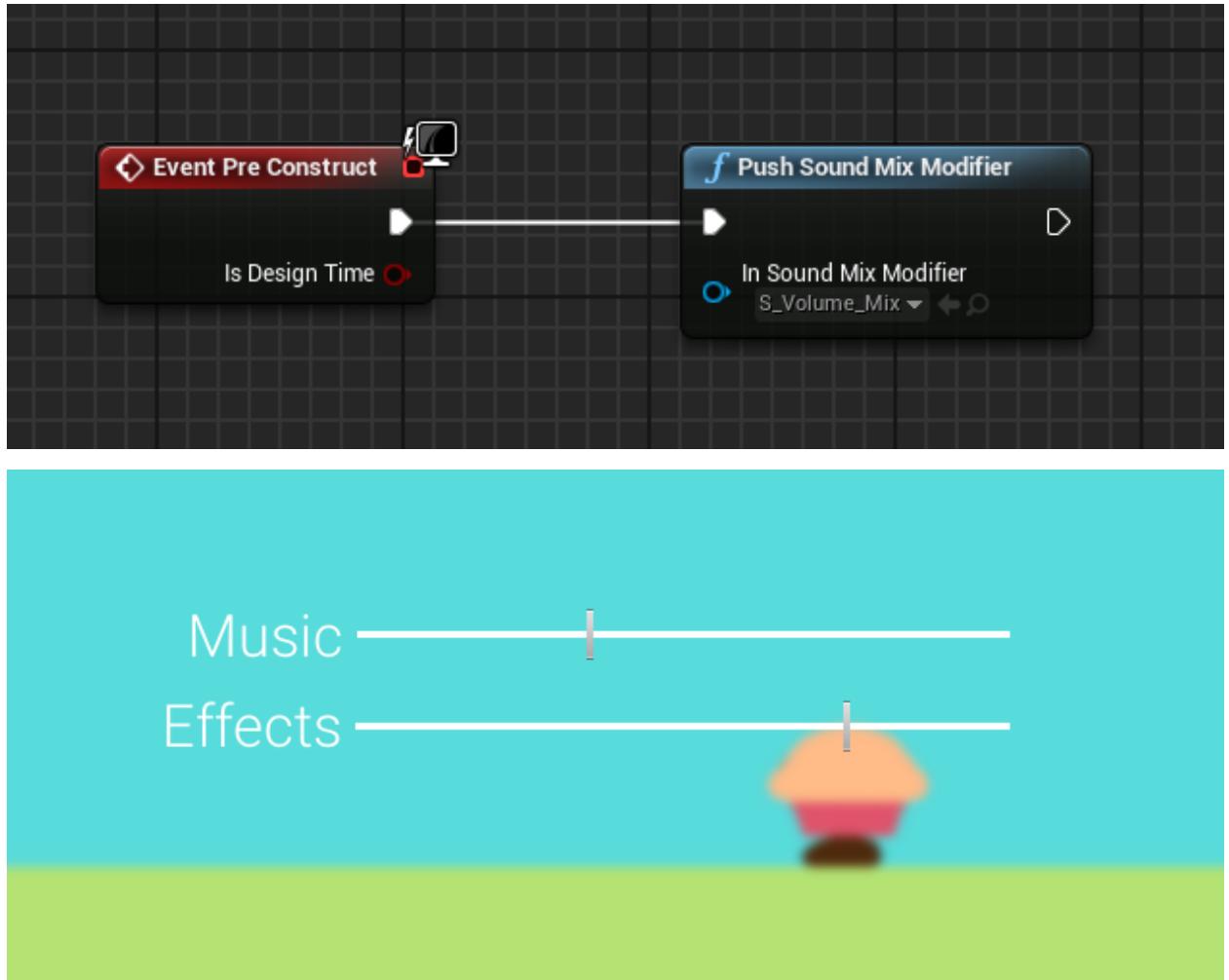


Рис.78- 79 Активация