Туториал по Unreal Engine. Часть 6: Анимация

Сегодня редко можно встретить игру без анимации, потому что она является важным аспектом передачи движения. Без анимации будет казаться, что персонаж не бежит, а скользит.

К счастью, Unreal позволяет быстро и удобно анимировать персонажей!

В этой части туториала вы научитесь следующему:

- Импортировать меш со скелетом
- Импортировать анимации
- Создавать Animation Blueprint для переходов между разными анимациями
- Выполнять плавные переходы анимаций

Приступаем к работе

Скачайте https://koenig-media.raywenderlich.com/uploads/2017/09/SkywardMuffinStarter.zip и распакуйте её. В корневом каталоге есть папка *Animation Assets*. В этой папке находятся персонаж и анимации, которые мы будем импортировать.

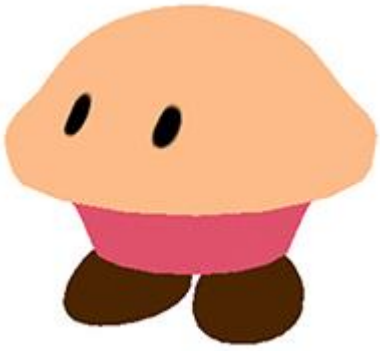
Откройте проект, перейдя в папку проекта и запустив *SkywardMuffin.uproject*.

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.

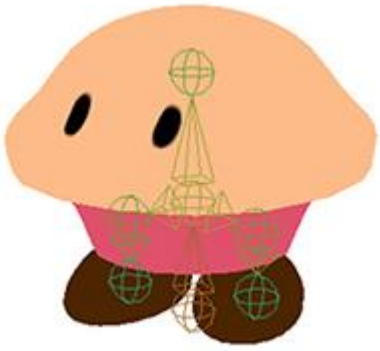
Нажмите на *Play*, чтобы запустить игру. Цель игры заключается в том, чтобы коснуться как можно большего числа облаков, не упав. Чтобы прыгнуть на первое облако, нажмите *левую клавишу мыши*.

Заголовок спойлера

Вместо простого красного круга давайте будем управлять этим милым маффином:

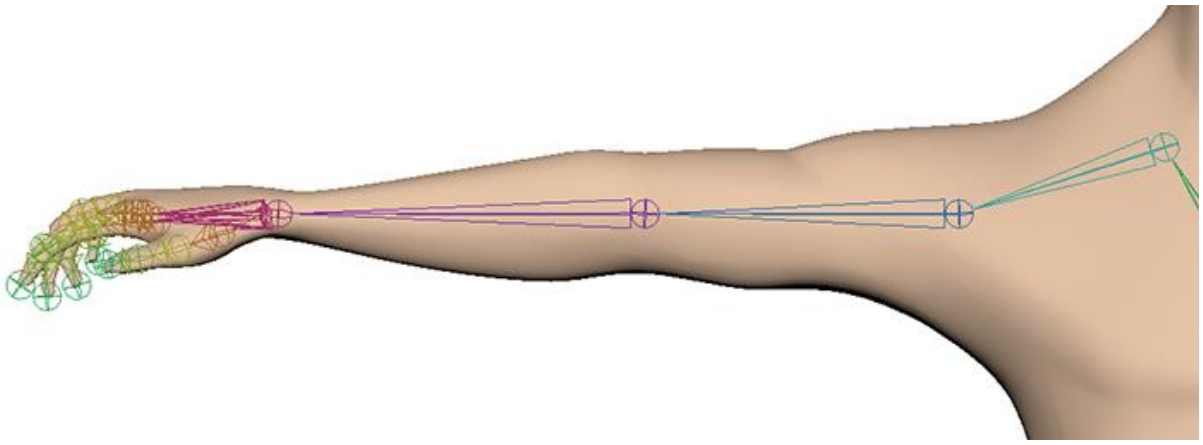


Этот маффин имеет *скелет*, который позволяет нам анимировать его.



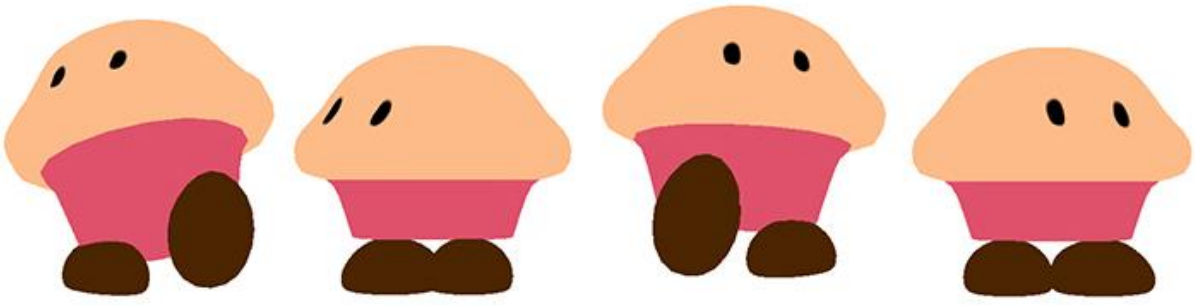
Что такое скелет?

В 3D-редакторах скелетом называется множество взаимосвязанных точек, называемых *шарнирами*. На рисунке ниже каждая сфера является шарниром.



Примечание: в Unreal используются взаимозаменяемые термины *joint* и *bone*.

Управляя этими шарнирами, можно создавать различные позы персонажа.



При переходе из одной позы в другую создаётся *анимация*.

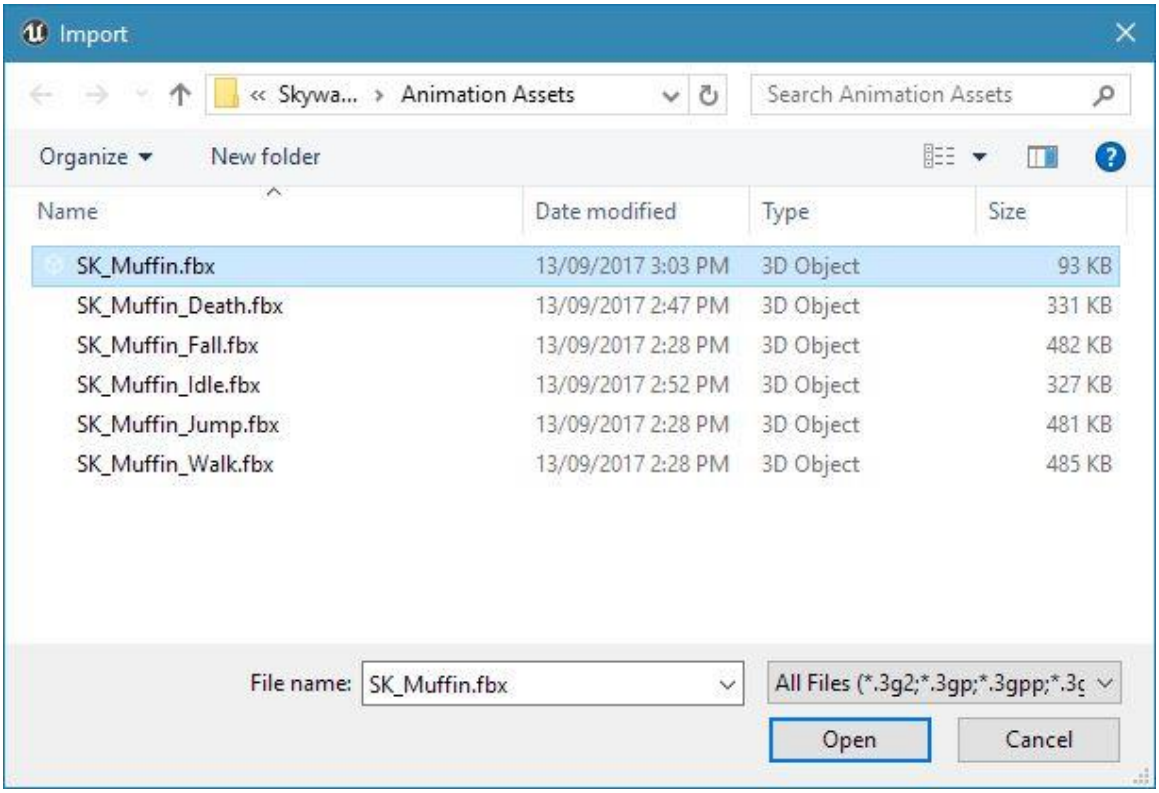
Если создать ещё больше поз между предыдущими, можно получить что-то вроде такого:

Заголовок спойлера

В Unreal любой меш со скелетом называется *Skeletal Mesh*. Давайте начнём с импорта Skeletal Mesh для маффина.

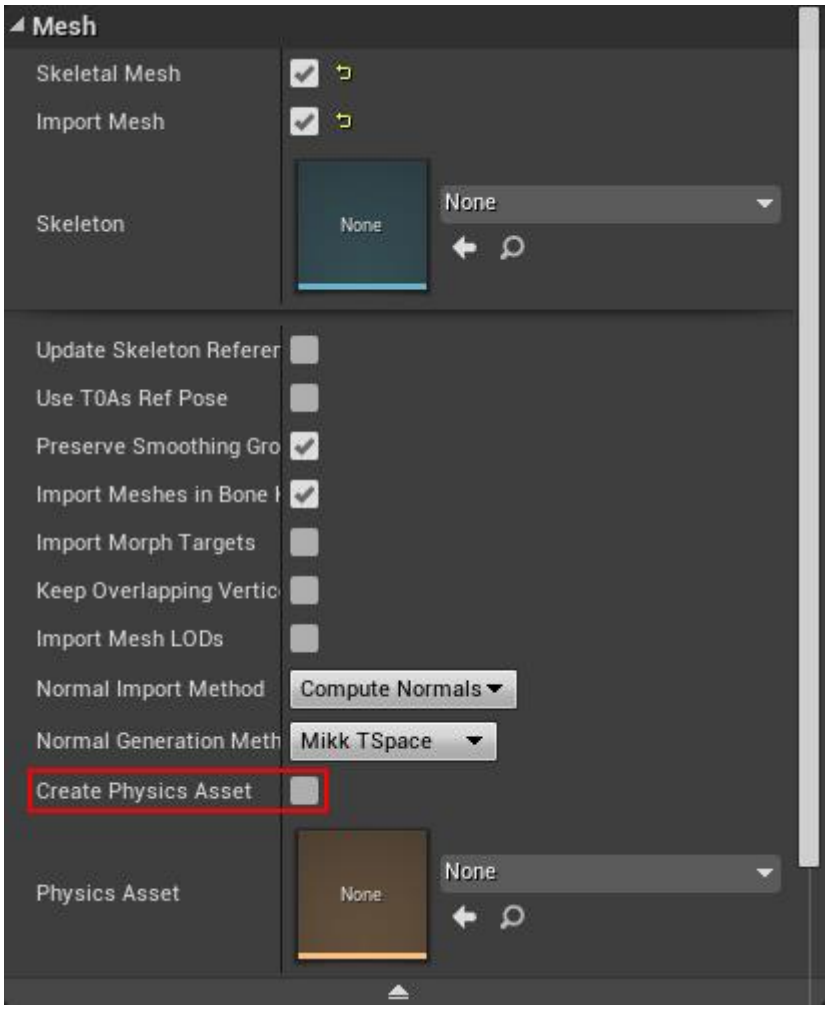
Импорт Skeletal Mesh

Перейдите в Content Browser и зайдите в *Characters\Muffin*. Нажмите на *Import* и перейдите в *SkywardMuffinStarter\Animation Assets*. Выберите *SK_Muffin.fbx* и нажмите *Open*.



В окне импорта перейдите в раздел *Mesh* и снимите флажок с опции *Create Physics Asset*.

Physics Asset помогает создать эффект рэгдолла. В этом туториале мы не будем его использовать, поэтому он нам не понадобится.



В проекте уже содержится материал и текстура маффина, поэтому импортировать их не нужно. *Снимите флажки* с опций *Import Materials* и *Import Textures*.

Все остальные настройки оставьте по умолчанию и намжите на *Import*. При этом создадутся следующие ассеты:

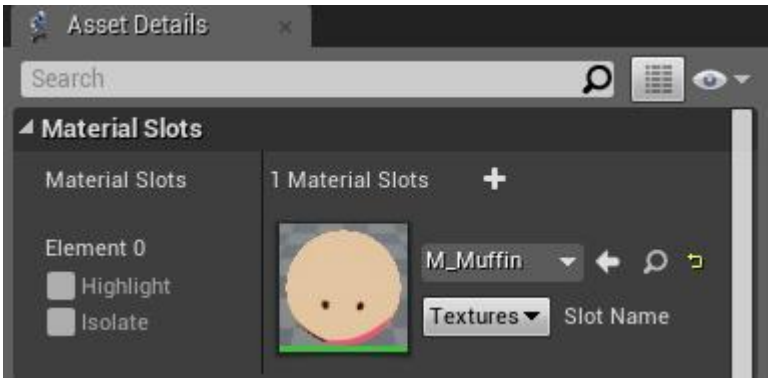
- *SK_Muffin:* ассет Skeletal Mesh. Это просто меш со ссылкой на ассет Skeleton.
- *SK_Muffin_Skeleton:* ассет Skeleton. Он содержит список шарниров и другую информацию, например, об их иерархии.

Импортировав маффин, мы готовы использовать его.

Использование Skeletal Mesh

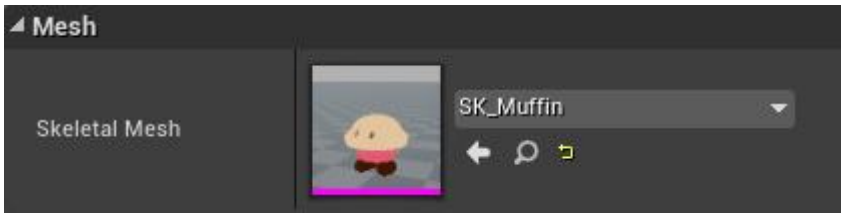
Прежде чем использовать новый Skeletal Mesh, нужно дать ему материал, чтобы он не был просто серым пятном. *Дважды щёлкните* на *SK_Muffin*, чтобы открыть его.

Перейдите в панель Asset Details и найдите раздел *Material Slots*. Назначьте материал *M_Muffin* и закройте *SK_Muffin*.



Теперь давайте используем *SK_Muffin* в качестве персонажа игрока. Вернитесь в Content Browser и *дважды щёлкните* на *BP_Muffin*, чтобы открыть его.

Перейдите к панели Components и выберите компонент *Mesh (Inherited)*. Переключитесь на панель Details и найдите раздел *Mesh*. Для свойства *Skeletal Mesh* выберите значение *SK_Muffin*.



Нажмите на *Compile* и вернитесь в основной редактор. Нажмите на *Play*, и вы сможете управлять в игре маффином!

Заголовок спойлера

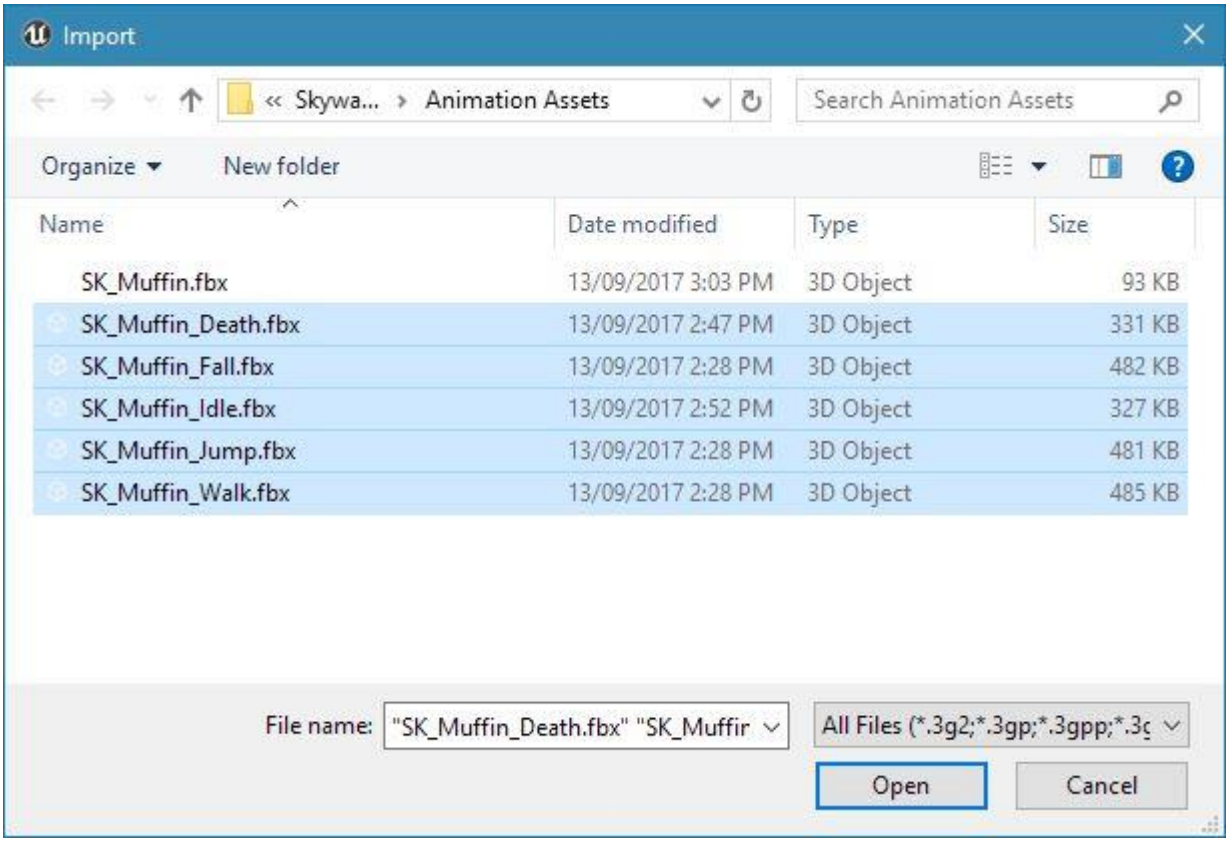
Игра уже выглядит намного лучше! Следующим шагом будет импорт анимаций, которые вдохнут в маффин жизнь.

Импорт анимаций

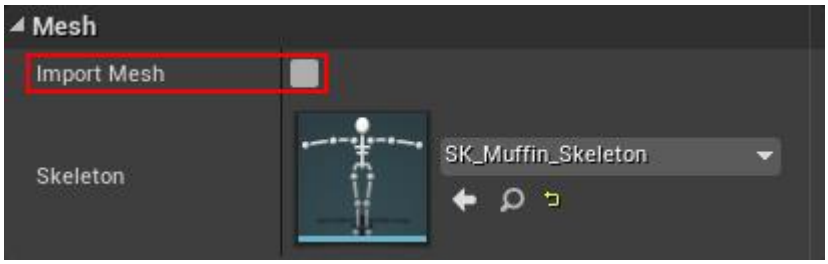
Перейдите в Content Browser и нажмите на *Import*. Перейдите в *SkywardMuffinStarter\Animation Assets*. Выберите следующие файлы:

- SK_Muffin_Death.fbx
- SK_Muffin_Fall.fbx
- SK_Muffin_Idle.fbx
- SK_Muffin_Jump.fbx
- SK_Muffin_Walk.fbx

Выбрав их, нажмите на *Open*.

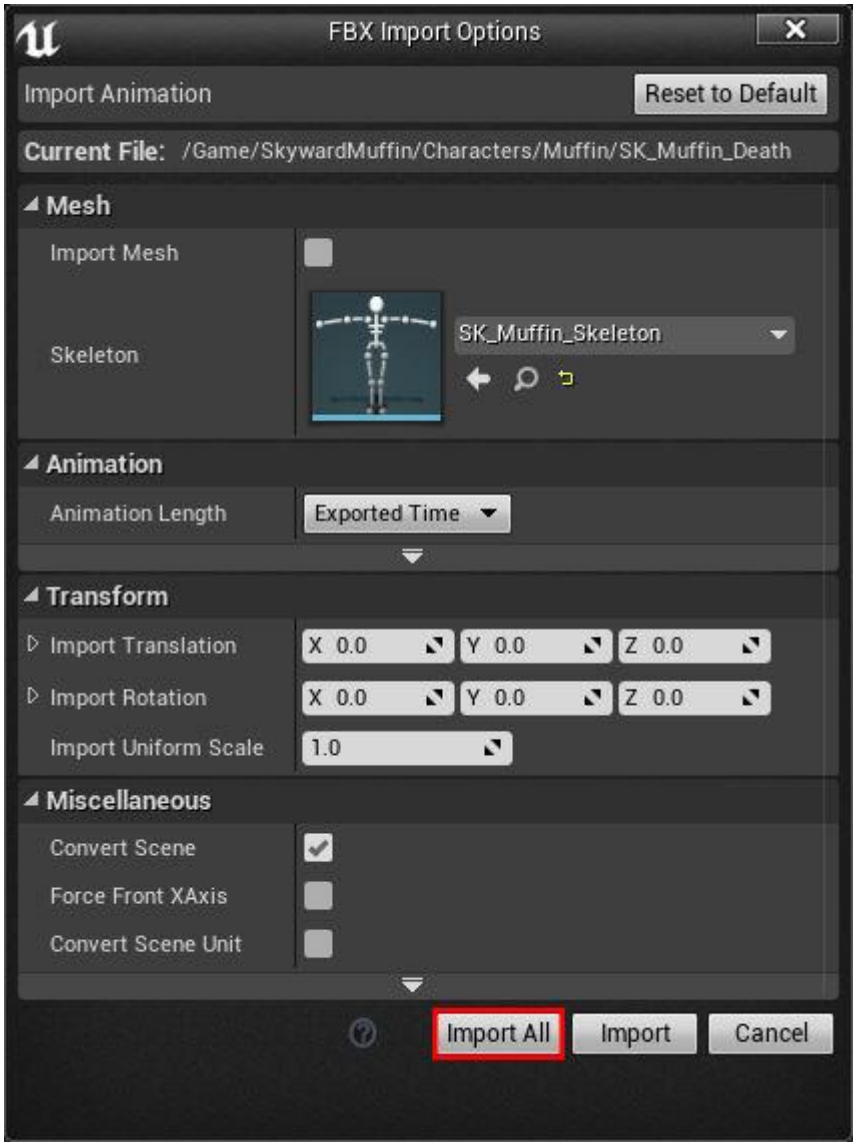


В окне импорта перейдите в раздел *Mesh* и *снимите флажок* с опции *Import Mesh*. Благодаря этому Skeletal Mesh не будет импортирован снова.



Теперь проверьте, что свойство *Skeleton* имеет значение *SK_Muffin_Skeleton*. Это определяет скелет, который будет использоваться в анимации.

Затем нажмите на *Import All*. Так вы импортируете все анимации с только что указанными настройками.



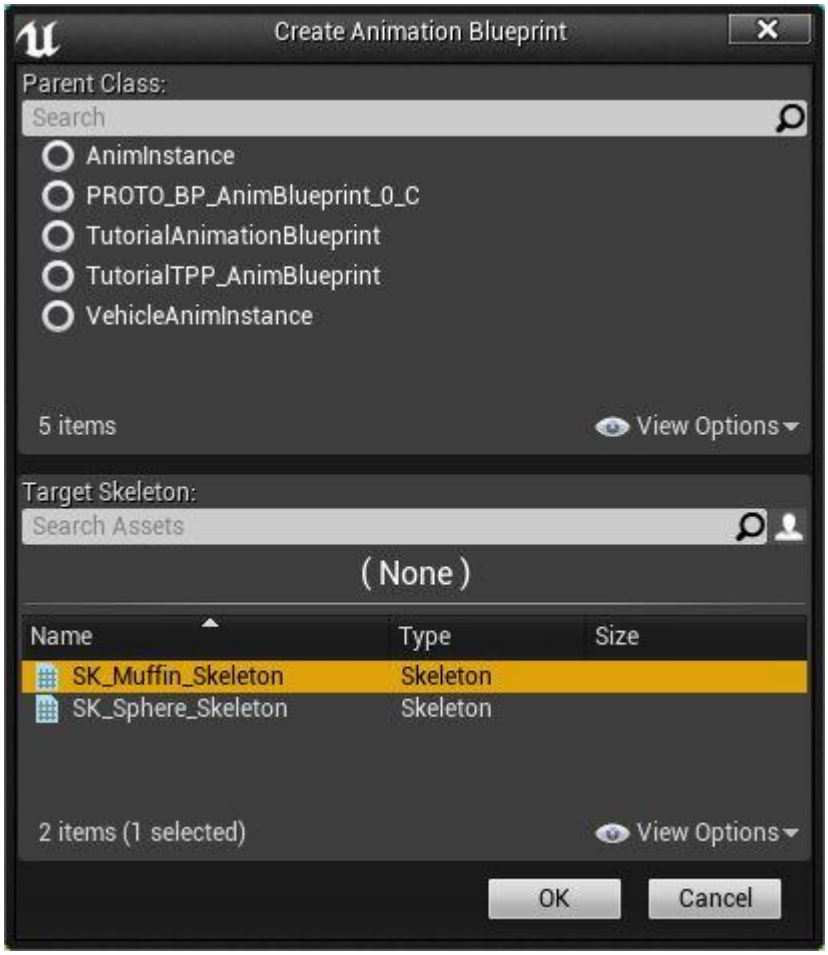
Теперь, когда у нас есть все анимации, нам нужен способ их воспроизведения. Можно воспользоваться для этого *Animation Blueprint*.

Создание Animation Blueprint

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

Чтобы создать его, перейдите в Content Browser и нажмите на кнопку *Add New*. Выберите *Animation\Animation Blueprint*.

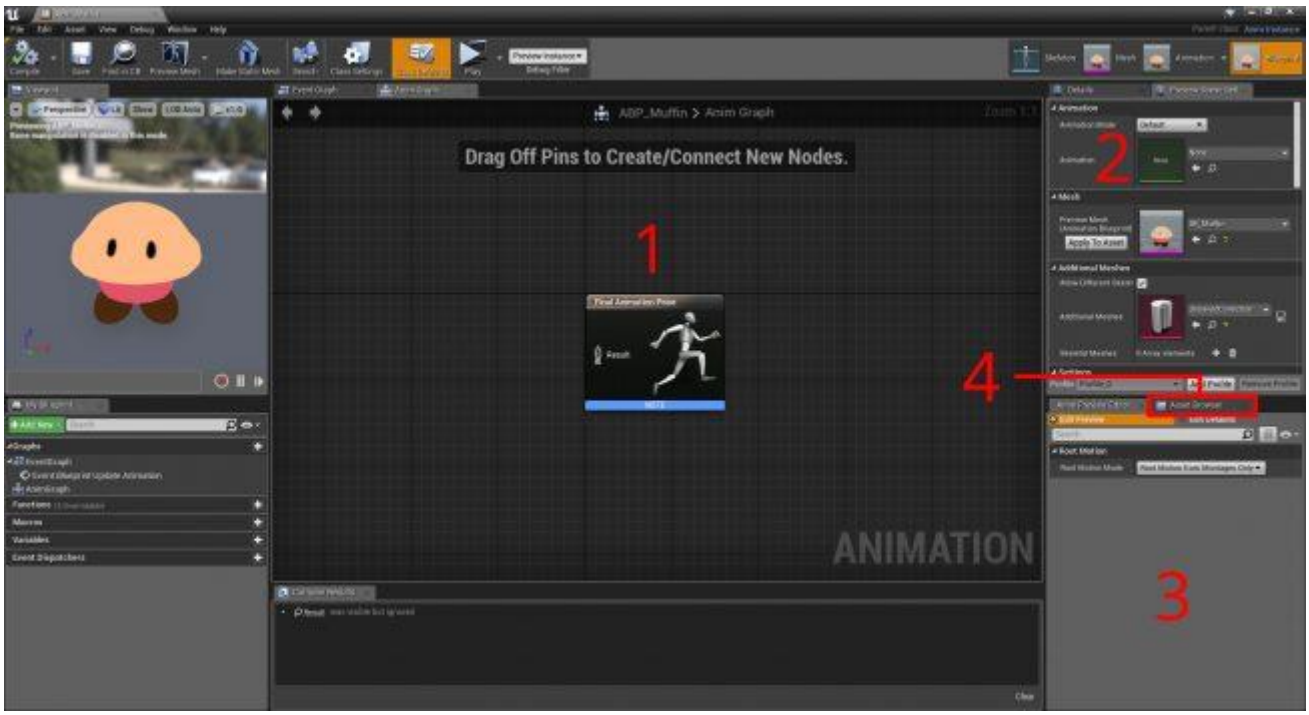
Во всплывающем окне найдите свойство *Target Skeleton* и выберите *SK_Muffin_Skeleton*. Затем нажмите на кнопку *OK* для создания Animation Blueprint.



Переименуйте ассет в *ABP_Muffin*. После этого *дважды щёлкните* на нём, чтобы открыть его в Animation Blueprint editor.

Animation Blueprint Editor

The Animation Blueprint editor похож на Blueprint editor, но в нём есть четыре дополнительных панели:



1. *Anim Graph:* это специальный граф для анимаций. Здесь воспроизводятся все анимации.
2. *Preview Scene Settings:* эта панель позволяет настраивать сцену предварительного просмотра во Viewport
3. *Anim Preview Editor:* создаваемые переменные также будут отображаться здесь. Используйте эту панель для предварительного просмотра того, как влияют ваши переменные на конечную анимацию.
4. *Asset Browser:* эта панель содержит список анимаций, которые может использовать текущий скелет.

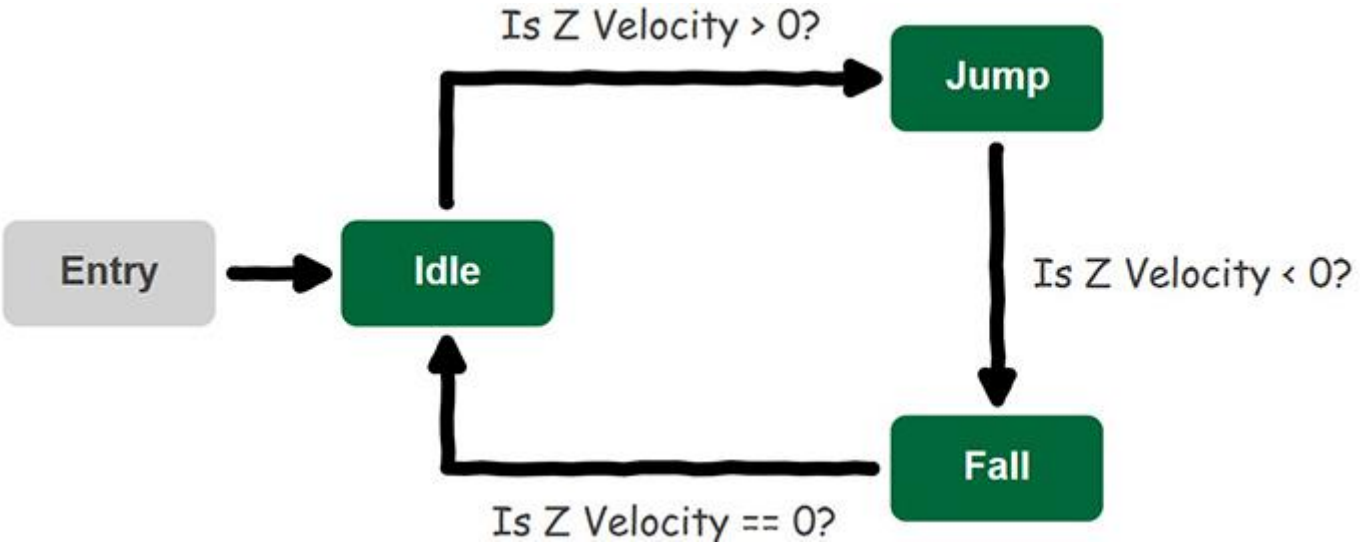
Для задания ситуаций, в которых должна воспроизводиться каждая анимация, можно использовать конечный автомат (*State Machine*).

Что такое конечный автомат?

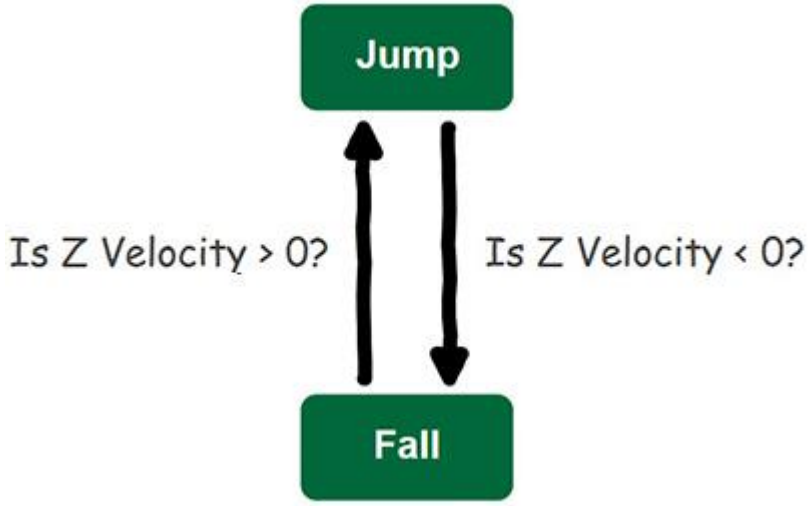
Конечный автомат — это набор *состояний* и *правил*. В нашем туториале можно считать состояние анимацией.

Конечные автоматы одновременно могут находиться только в одном состоянии. Для перехода в следующее состояние должны выполняться определённые условия, задаваемые правилами.

Ниже представлен простой пример конечного автомата. В нём показаны состояния прыжка и правила перехода в каждое состояние.



Состояния могут иметь и двустороннюю взаимосвязь. В представленном ниже примере состояния Jump и Fall могут переходить друг в друга.

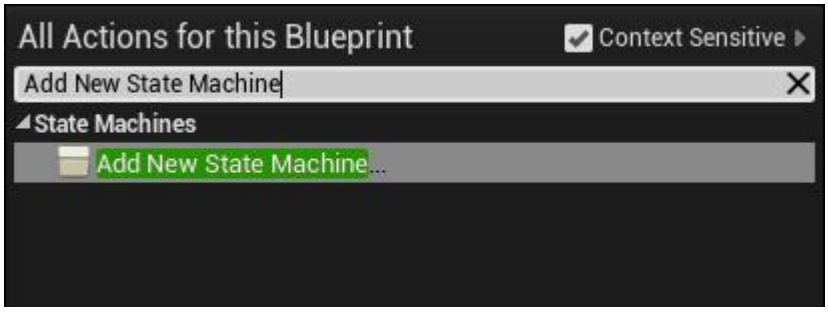


Без этой двусторонней связи персонаж не смог бы выполнять двойной прыжок, потому что он мог бы переходить в состояние Jump только из состояния Idle.

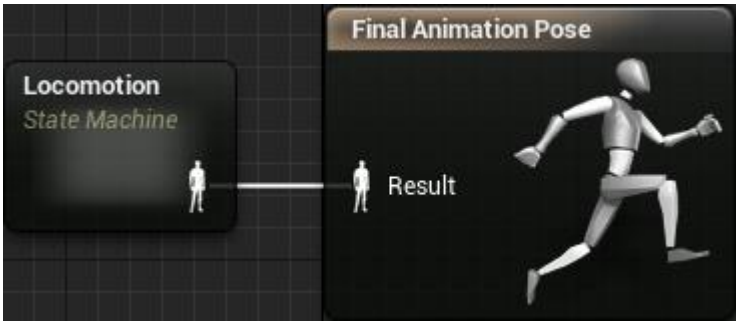
Ну, довольно о конечных автоматах, давайте уже займёмся их созданием.

Создание конечного автомата

Перейдите в Anim Graph и *нажмите правой клавишей мыши* на пустой области. В меню выберите *Add New State Machine*.



Это добавит в граф нод State Machine. Переименуйте State Machine в *Locomotion*. Затем соедините конечный автомат *Locomotion* с нодом *Final Animation Pose*.



Теперь конечный автомат *Locomotion* будет определять конечную анимацию маффина.

Дважды щёлкните на конечном автомате *Locomotion*, чтобы открыть его. Внутри вы увидите нод *Entry*.

Соединённое с этим нодом состояние является состоянием *по умолчанию*. В нашем туториале состоянием по умолчанию будет анимация ожидания. Создайте это состояние, *нажав правой клавишей мыши* на пустой области графа. В меню выберите *Add State* и переименуйте его в *Idle*.

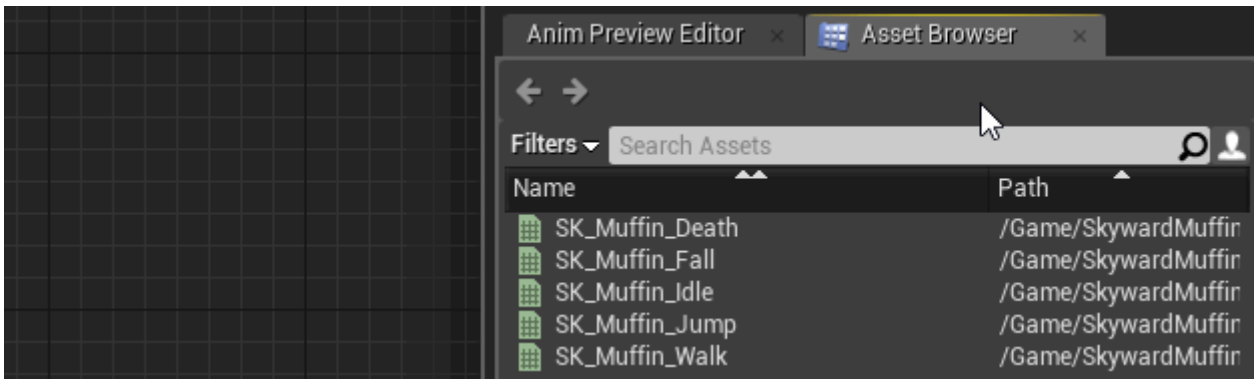
Теперь нужно соединить нод *Entry* с состоянием *Idle*. *Перетащите* контакт *Entry* к *серой* области состояния *Idle*. Отпустите *левую клавишу мыши*, чтобы соединить их.

При создании состояния через контекстное меню с ним не будет связана никакая анимация. Давайте это исправим.

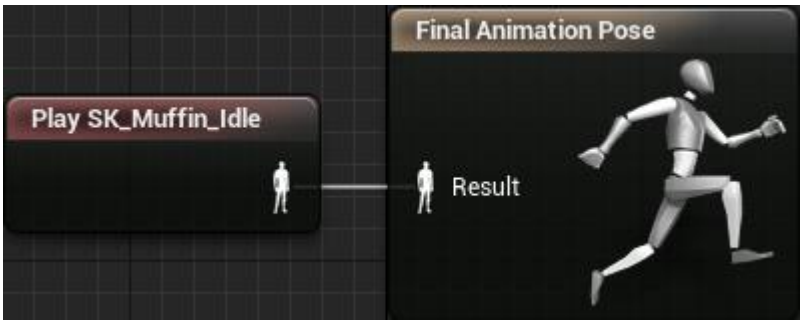
Привязывание анимации к состоянию

Дважды щёлкните на состоянии *Idle*, чтобы открыть его.

Чтобы привязать анимацию, перейдите в Asset Browser и *перетащите* анимацию *SK_Muffin_Idle*. Отпустите *левую клавишу мыши* на пустой области графа, чтобы добавить её.



Затем присоедините нод *Play SK_Muffin_Idle* к ноду *Final Animation Pose*.



Для использования Animation Blueprint нам нужно обновить *BP_Muffin*.

Использование Animation Blueprint

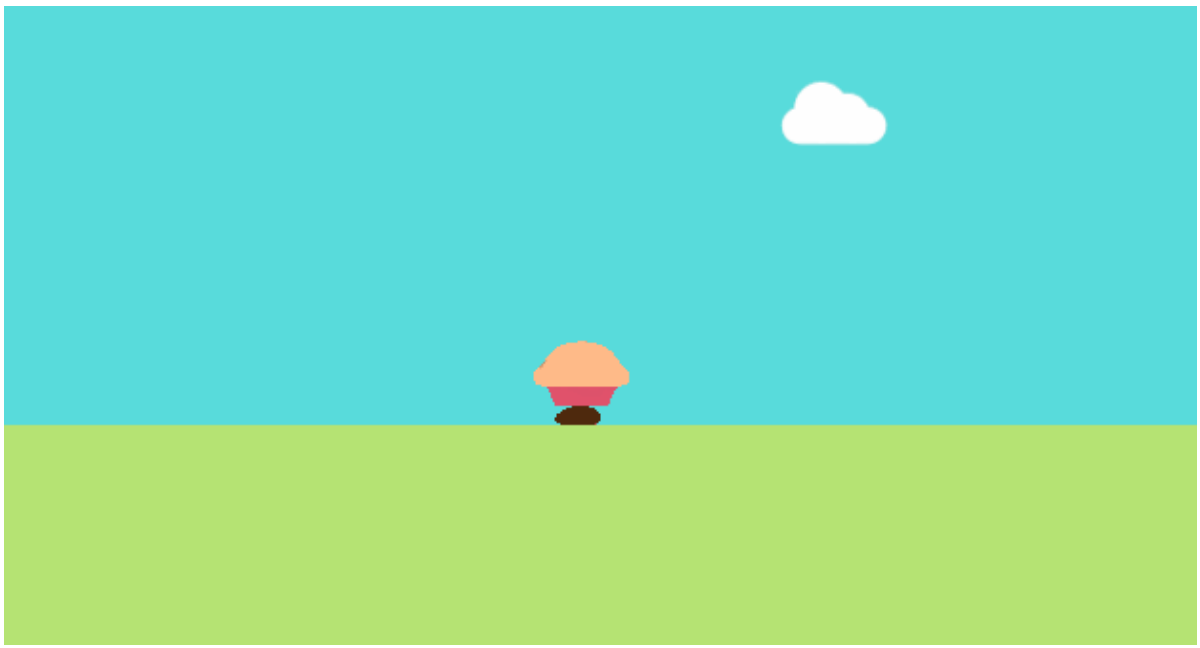
Нажмите на *Compile* и переключитесь на *BP_Muffin*.

Перейдите в панель Components и выберите компонент *Mesh (Inherited)*. Перейдите в панель и найдите раздел *Animation*.

Выберите для *Animation Mode* значение *Use Animation Blueprint*. Для *Anim Class* выберите значение *ABP_Muffin*.

Теперь Skeletal Mesh в качестве своего Animation Blueprint будет использовать *ABP_Muffin*.

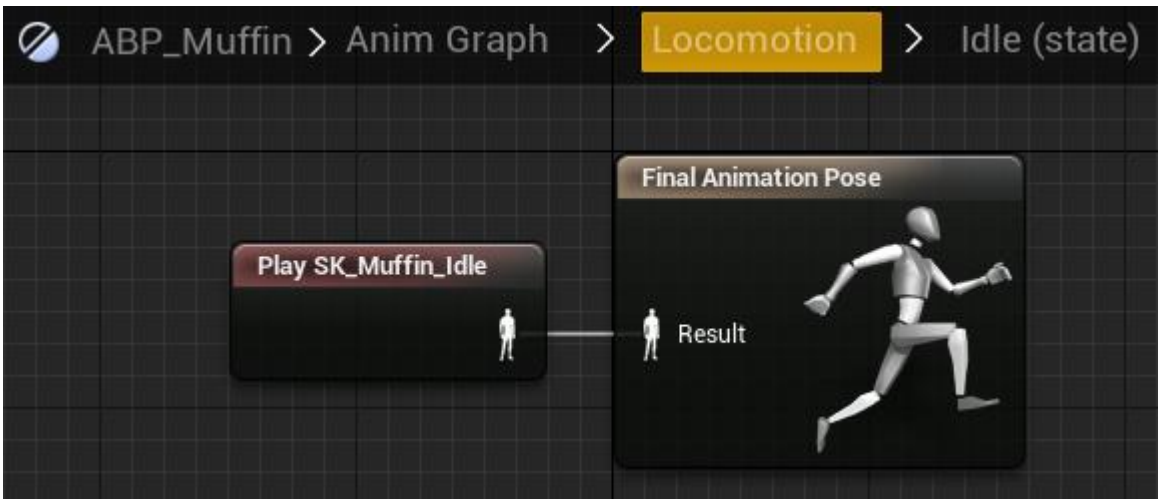
Нажмите на *Compile* и закройте *BP_Muffin*. Перейдите в основной редактор и нажмите на *Play*, чтобы проверить Animation Blueprint. Поскольку *Idle* является состоянием по умолчанию, маффин сразу же использует анимацию ожидания.



В следующем разделе мы создадим состояния для прыжков и падения.

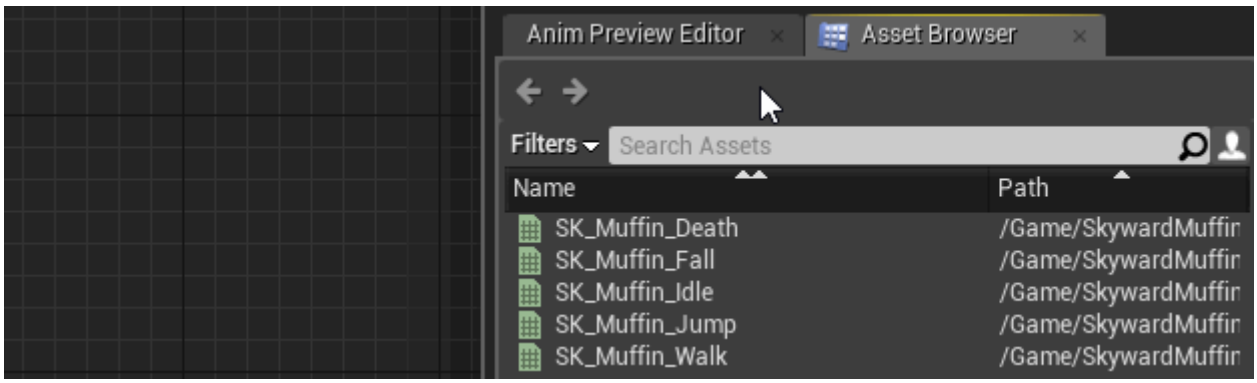
Создание состояний прыжка и падения

Вернитесь к *ABP_Muffin* и переключитесь к графу конечного автомата *Locomotion*. Это можно сделать, нажав на слово *Locomotion*, расположенное в верхней части графа.



Вместо создания состояния и привязки к нему анимации можно создать состояние с уже привязанной анимацией. Давайте сделаем так для состояния прыжка.

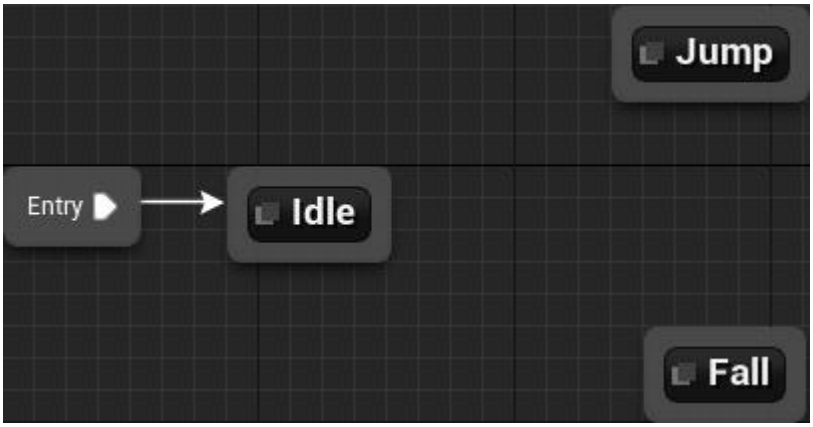
Перейдите в Asset Browser и *перетащите* анимацию *SK_Muffin_Jump*. Отпустите *левую клавишу мыши* на пустой области графа. Это создаст состояние с уже привязанной к нему анимацией.



Переименуйте состояние в *Jump*.

Повторите процесс с анимацией *SK_Muffin_Fall* и переименуйте состояние в *Fall*.

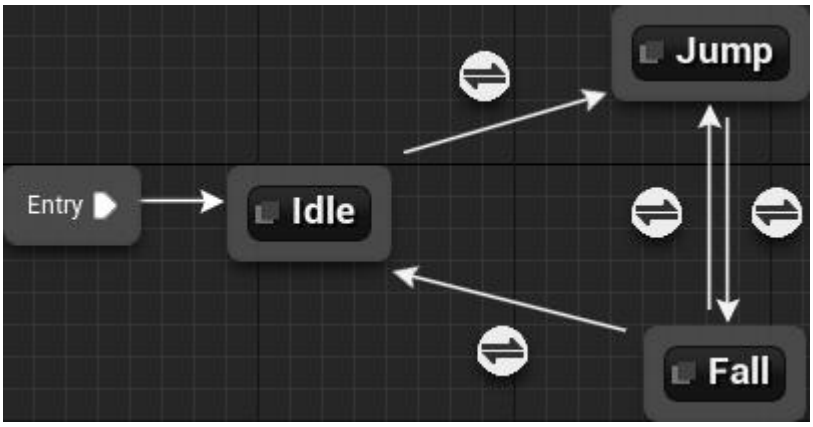
Теперь у нас есть три состояния: *Idle*, *Jump* и *Fall*.



Теперь мы свяжем состояния друг с другом. Это можно сделать, *перетащив серую* область состояния, *из* которого нужно выполнить переход. Отпустите *левую клавишу мыши* на *серой* области *целевого* состояния для создания перехода.

Создайте следующие переходы:

- Из *Idle* в *Jump*
- Из *Jump* в *Fall*
- Из *Fall* в *Jump*
- Из *Fall* в *Idle*

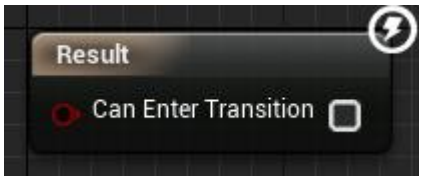


Теперь, когда у нас есть переходы, нужно задать условия, при которых они происходят. Это можно сделать с помощью правил переходов (*Transition Rules*).

Transition Rules

Этот значок обозначает правило перехода (Transition Rule):

Каждое правило перехода содержит нод *Result* с единственным булевым входом.



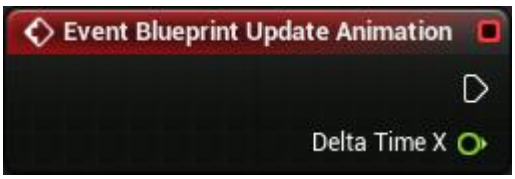
Если этот вход равен *true*, то происходит переход.

Далее нужно создать переменные, сообщающие нам, прыгает или падает игрок. Мы применим эти переменные в правилах переходов.

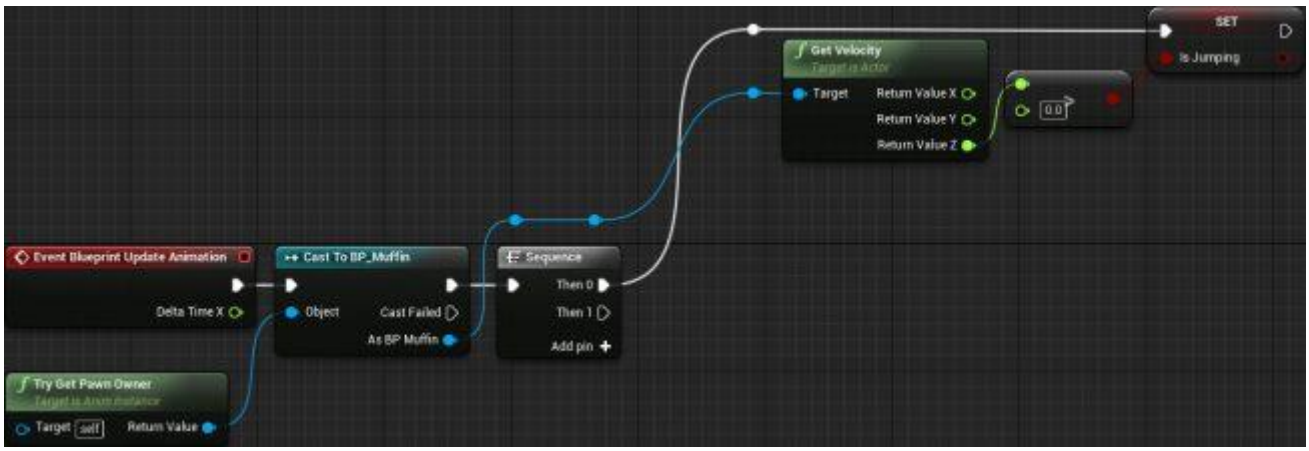
Определяем, прыгает персонаж или падает

Создайте две *булевы* переменные с названиями *IsJumping* и *IsFalling*.

Во-первых, мы зададим значение *IsJumping*. Переключитесь на Event Graph и найдите нод *Event Blueprint Update Animation*. Этот нод работает подобно ноду *Event Tick*.



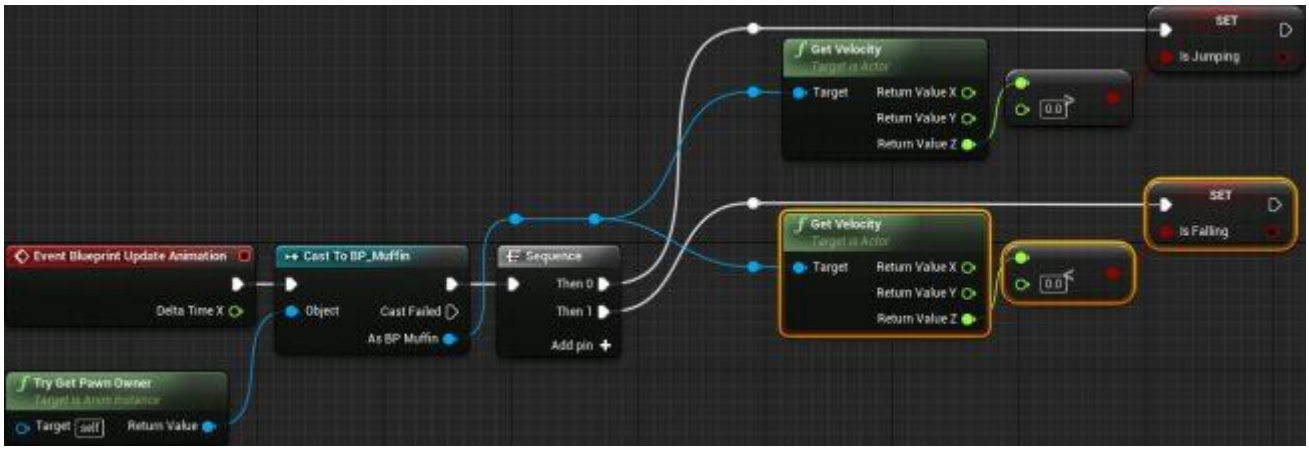
Чтобы проверить, прыгает ли персонаж, создадим следующую схему:



Так мы проверим, больше ли нуля *скорость* персонажа по *оси Z*. Если это так, то персонаж прыгает, а *IsJumping* должна принимать значение *true*.

Примечание: приведите её к классу, который будет использовать Animation Blueprint. Это очень важно для того, чтобы можно было предварительно просматривать переменные с помощью Anim Preview Editor.

Чтобы проверить, падает ли персонаж, нам достаточно выполнить противоположную проверку. Добавьте выделенные ноды:

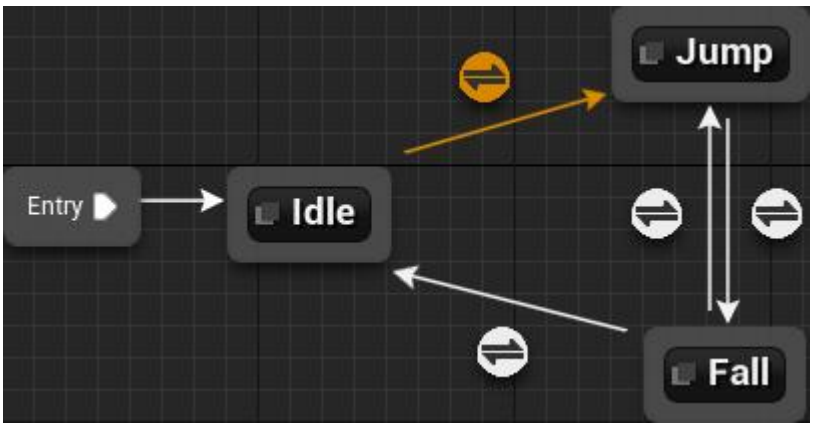


Теперь *IsFalling* будет равна *true*, когда *скорость по Z* персонажа меньше нуля.

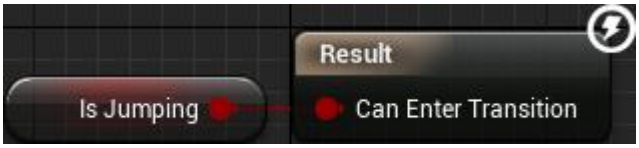
Настало время использовать эти переменные для задания правил переходов.

Определение правил переходов

Во-первых, нам нужно задать правило перехода *Idle to Jump*. Переключитесь обратно к конечному автомату *Locomotion*. *Дважды щёлкните* на правиле перехода *Idle to Jump*, чтобы открыть его.



Создайте нод *IsJumping* и соедините его с нодом *Result*.



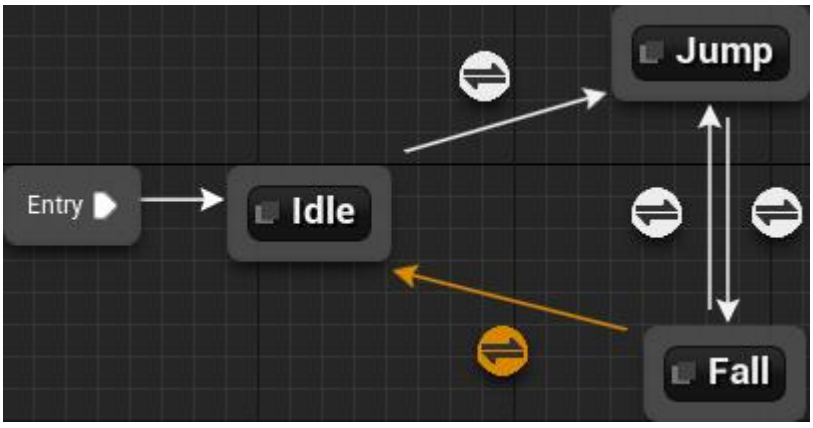
Теперь состояние *Idle* может переходить в состояние *Jump*, когда *IsJumping* равно *true*.

Повторите процесс для правил переходов *Jump to Fall* и *Fall to Jump*. Используйте следующие переменные:

- *Из Jump в Fall:* IsFalling
- *Из Fall в Jump:* IsJumping

Теперь состояния *Jump* и *Fall* могут переходить друг в друга.

Осталось назначить ещё одно правило перехода. Откройте правило перехода *Fall to Idle*.



Для перехода в состояние *Idle* игрок не должен прыгать или падать. Чтобы выполнить эту

проверку, можно использовать нод *NOR*. Этот нод возвращает *true* только когда оба входа имеют значения *false*.

Создайте нод *NOR* и соедините с ним нод *IsJumping* и *IsFalling*. После этого соедините нод *NOR* с нодом *Result*.

Теперь состояние *Fall* может переходить в состояние *Idle*, когда *IsJumping* и *IsFalling* равны *false*.

Нажмите на *Compile* и вернитесь в основной редактор. Нажмите на *Play* для проверки переходов.

Заголовок спойлера

Примечание: также вы можете проверить переходы, изменяя переменные в Anim Preview Editor.

Пока маффин при движении по земле только скользит, потому что мы ещё не применили анимацию ходьбы!

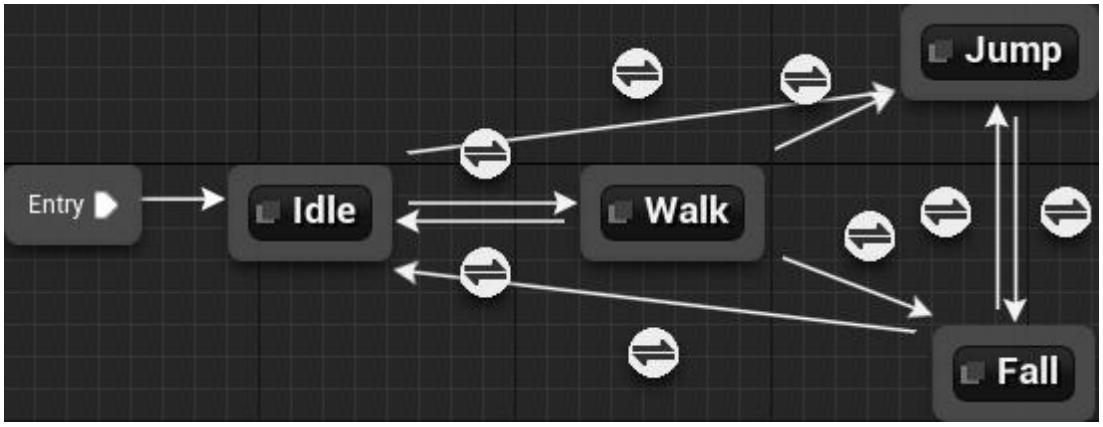
Вместо создания нового состояния для ходьбы можно смешать его с анимацией ожидания с помощью *Blend Space*.

Что такое Blend Space?

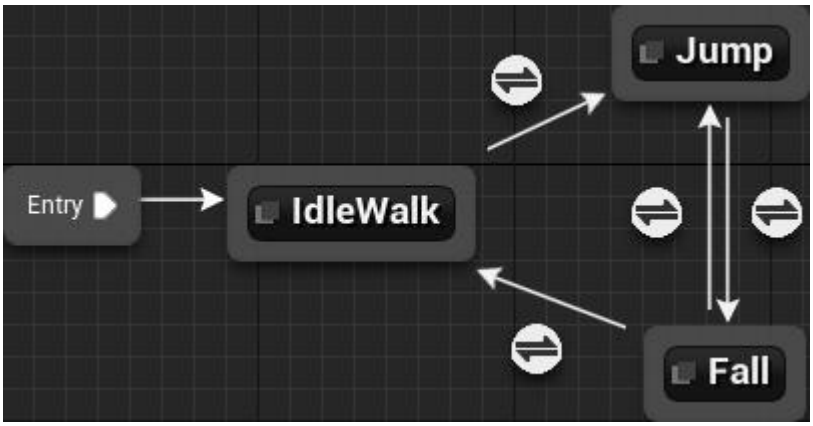
Blend Space — это тип анимационного ресурса. Он выполняет интерполяцию между разными анимациями на основании входных значений. В этом туториале мы будем использовать в качестве входных данных *скорость* игрока.

Заголовок спойлера

Blend Space также помогают упростить конечные автоматы. Вот как будет выглядеть конечный автомат *Locomotion*, если не использовать для ходьбы Blend Space:



Благодаря Blend Space достаточно просто заменить анимацию ожидания.



Теперь, когда вы познакомились с магией Blend Space, настало время создать его.

Создание Blend Space

Перейдите в Content Browser и нажмите на *Add New*. Выберите *Animation\Blend Space 1D*.

Примечание: разница между *Blend Space* и *Blend Space 1D* заключается в том, что у первого есть *два* входа. Последнее имеет только *один* вход.

Во всплывающем окне выберите *SK_Muffin_Skeleton*.

Переименуйте ассет в *BS_IdleWalk* и *дважды щёлкните* на нём, чтобы открыть его в Animation editor.

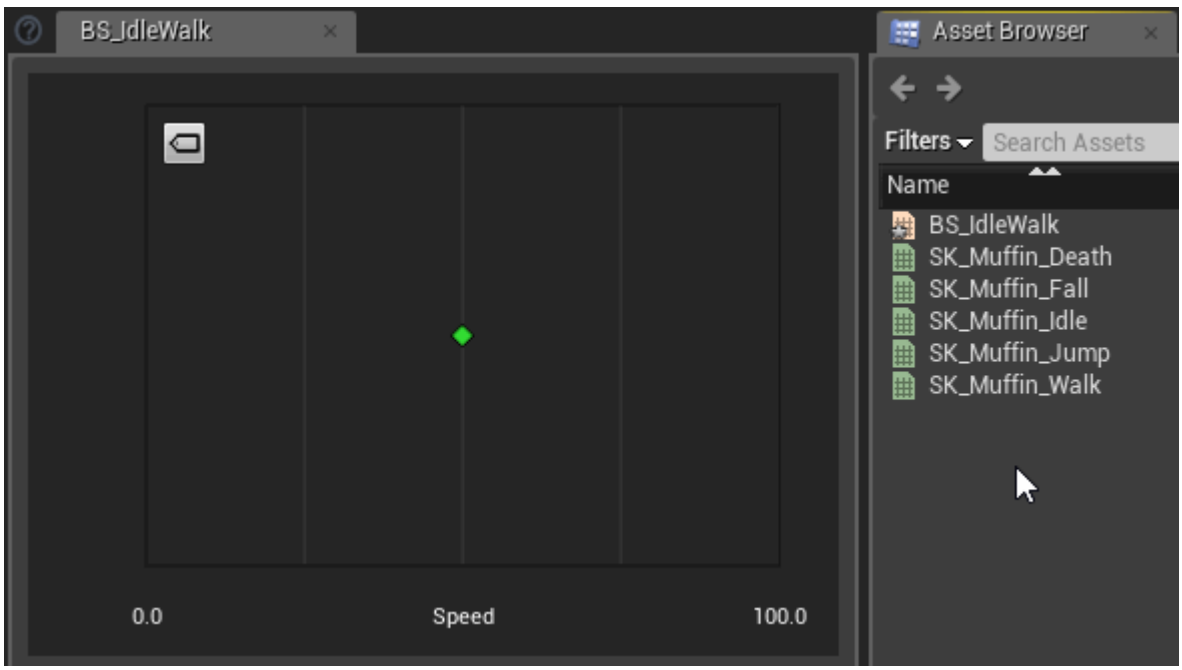
Когда откроется Blend Space, вы увидите панель в нижней части экрана. Это Blend Space editor, в котором мы будем добавлять анимации.

Давайте добавим анимации в Blend Space.

Добавление анимаций в Blend Space

Во-первых, нужно изменить имя значеняи оси (входа). Перейдите в панель Asset Details и найдите раздел *Axis Settings*. Измените свойство *Horizontal Axis\Name* на *Speed*.

Теперь мы добавим анимации. Перейдите в Asset Browser и *перетащите* анимацию *SK_Muffin_Idle*. Переместите её *влево* на сетке Blend Space, чтобы она привязалась к значению *0.0*. Отпустите *левую клавишу мыши*, чтобы добавить анимацию.



Примечание: для отображения названий анимаций нажмите на значок *метки* в *верхнем левом* углу сетки Blend Space.

После этого добавьте анимацию *SK_Muffin_Walk* в значение *100.0*.

Теперь Blend Space будет смешивать анимации ожидания и ходьбы в зависимости от входного значения. Если входное значение равно *0*, то будет воспроизводиться только анимация ожидания. Если входное значение равно *100*, то будет воспроизводиться только анимация ходьбы. Все промежуточные значения будут смешиваться.

Примечание: эти значения выбраны произвольно. Например, можно изменить максимальное значение, сделав его равным 500. В этом случае анимация ходьбы будет воспроизводиться только при более высоких скоростях.

Можно изменять значения раздела *Axis Settings* в панели Asset Details.

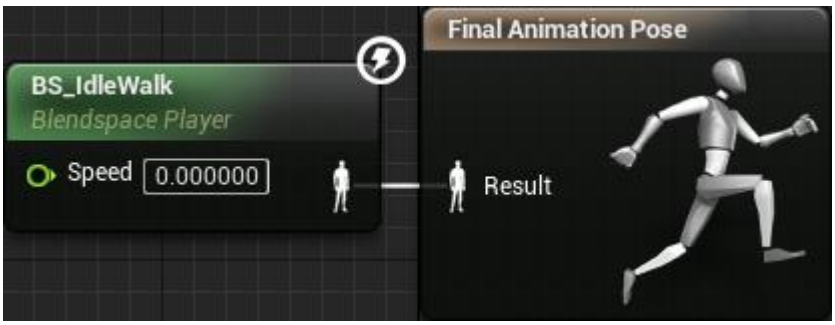
Настало время использовать Blend Space.

Применяем Blend Space

Закройте *BS_IdleWalk* и откройте *ABP_Muffin*. Переключитесь на конечный автомат *Locomotion* и откройте состояние *Idle*.

Во-первых, удалите нод *Play SK_Muffin_Idle*.

Затем добавьте Blend Space *BS_IdleWalk* с помощью перетаскивания. Соедините нод *BS_IdleWalk* с нодом *Final Animation Pose*.



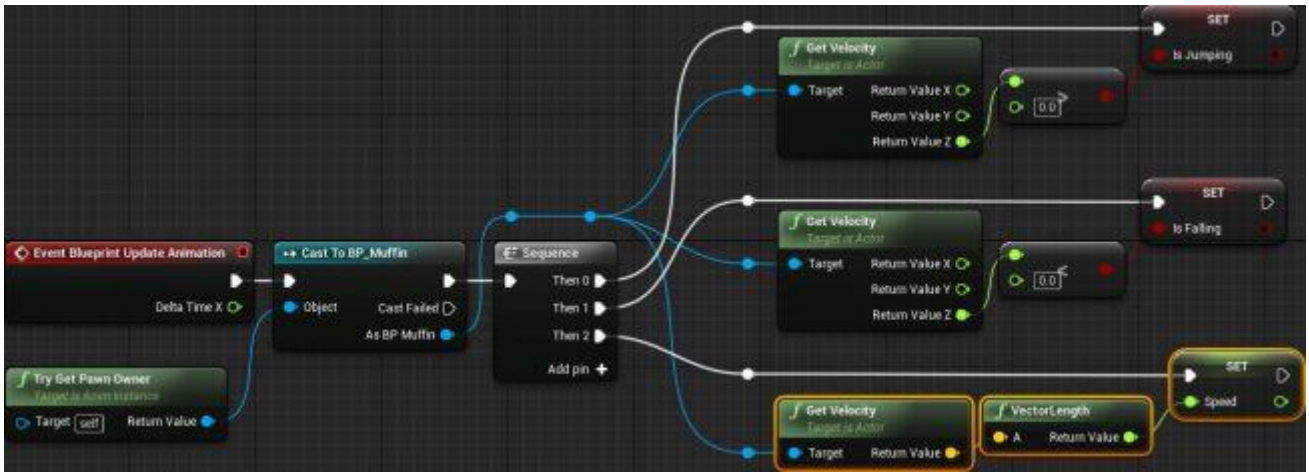
Теперь *BS_IdleWalk* будет воспроизводиться автоматически, потому что является состоянием по умолчанию. Однако оно будет отображать только анимацию ожидания. Так происходит потому, что вход *Speed* остаётся равным *0*.

Чтобы исправить это, нужно передать ему скорость игрока.

Получение скорости игрока

Создайте новую переменную типа *float* с именем *Speed*. Затем переключитесь на Event Graph.

Добавьте новый контакт к ноду *Sequence*, а затем добавьте к нему выделенные ноды:



Эта схема будет постоянно приравнивать переменную *Speed* к значению скорости игрока.

Переключитесь обратно к графу состояния *Idle*. Соедините переменную *Speed* со входом *Speed* нода *BS_IdleWalk*.

Теперь сможет *BS_IdleWalk* выполнять смешивание между анимациями ожидания и ходьбы.

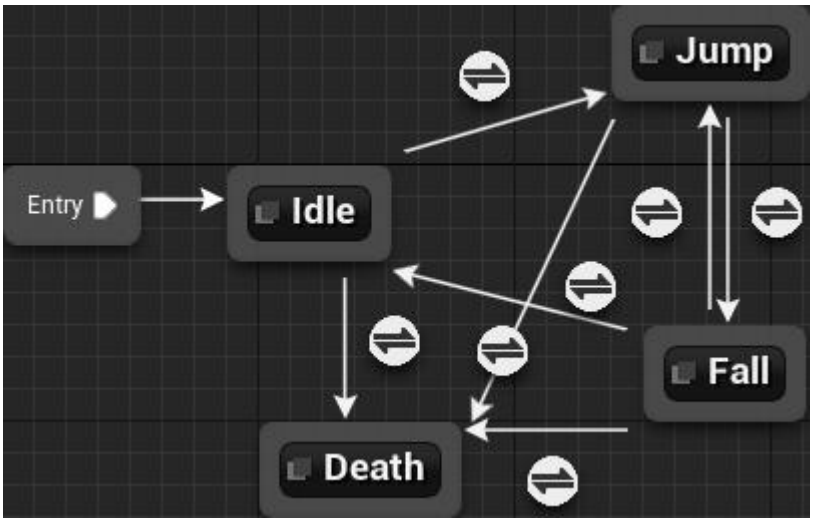
Нажмите на *Compile* и вернитесь в основной редактор. Нажмите на *Play*, чтобы протестировать Blend Space.

Заголовок спойлера

Есть ещё одна анимация, которую нам нужно использовать: анимация смерти!

Использование анимации смерти

В этой игре можно умереть только в состоянии *Idle* (на земле). Однако давайте представим, что игрок может умереть в любом состоянии. Первой мыслью будет создать состояние *Death* и соединить с ним все состояния. Так можно сделать, но это быстро приведёт к запутанному графу.



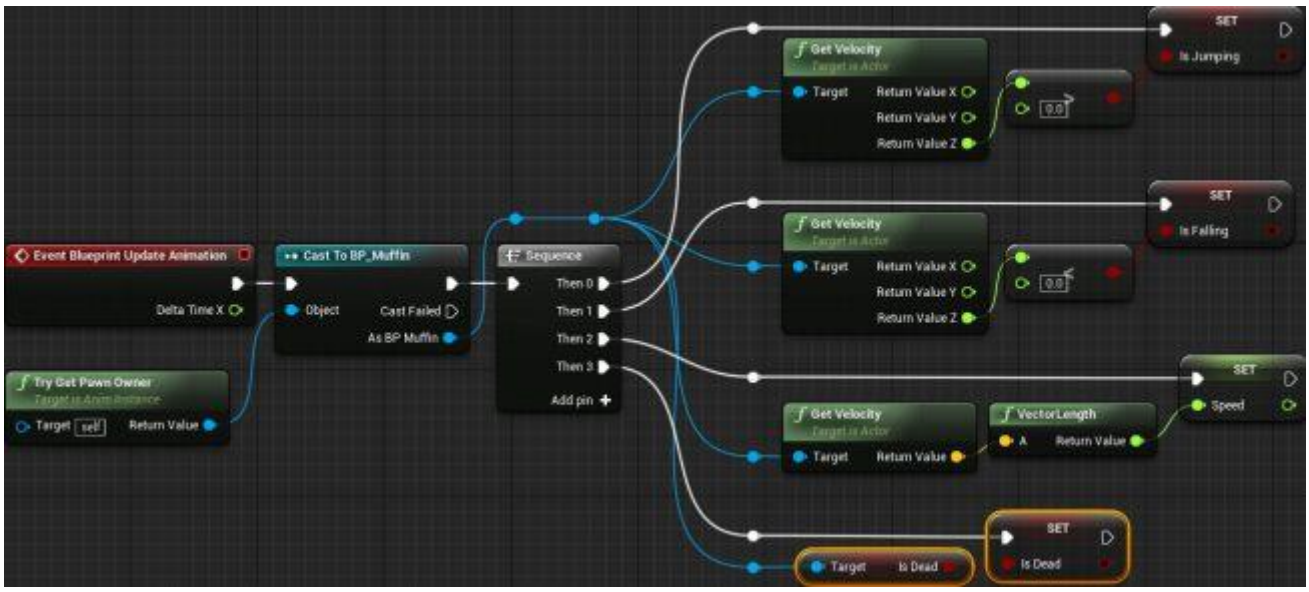
Решением может стать использование нода *Blend Poses by bool*. Этот нод может переключаться между двумя анимациями в зависимости от значения входного булева значения.

Прежде чем создать этот нод, нам нужна переменная, содержащая статус смерти игрока.

Проверяем, умер ли игрок

Вернитесь назад к *ABP_Muffin* и создайте переменную типа *boolean* под названием *IsDead*. Затем переключитесь на Event Graph.

Добавьте новый контакт к ноду *Sequence*, а затем добавьте к нему выделенные ноды:

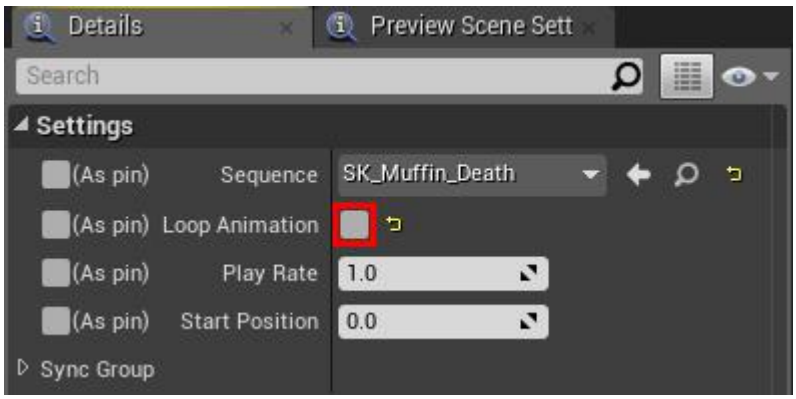


Так мы зададим переменную *IsDead*, зависящую от состояния смерти игрока.

Затем мы воспользуемся нодом *Blend Poses by bool*.

Использование нода Blend Poses by Bool

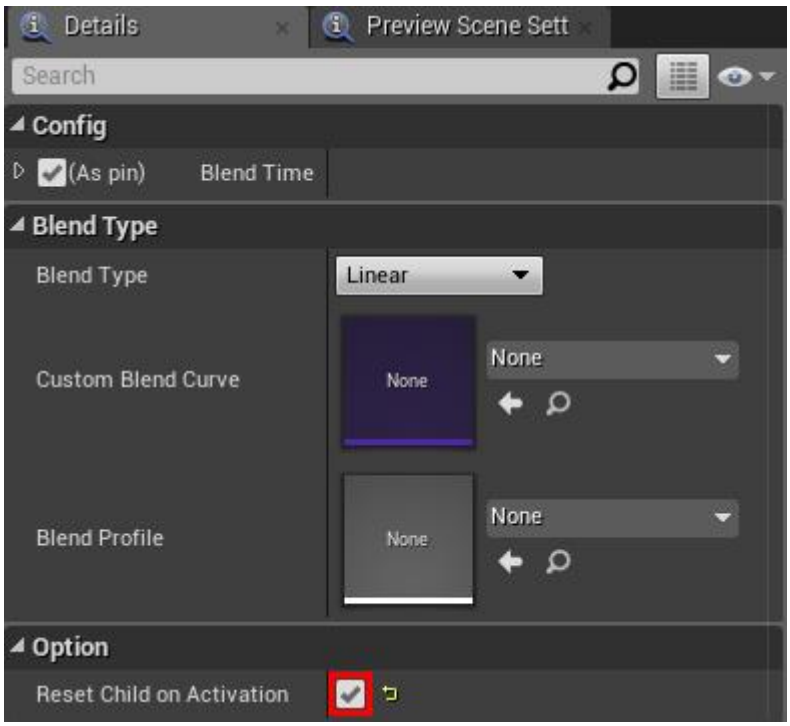
Переключитесь на Anim Graph и добавьте анимацию *SK_Muffin_Death*. Выбрав её, перейдите в панель Details и *снимите флажок* со свойства *Loop Animation*.



Благодаря этому анимация смерти будет проигрываться только один раз.

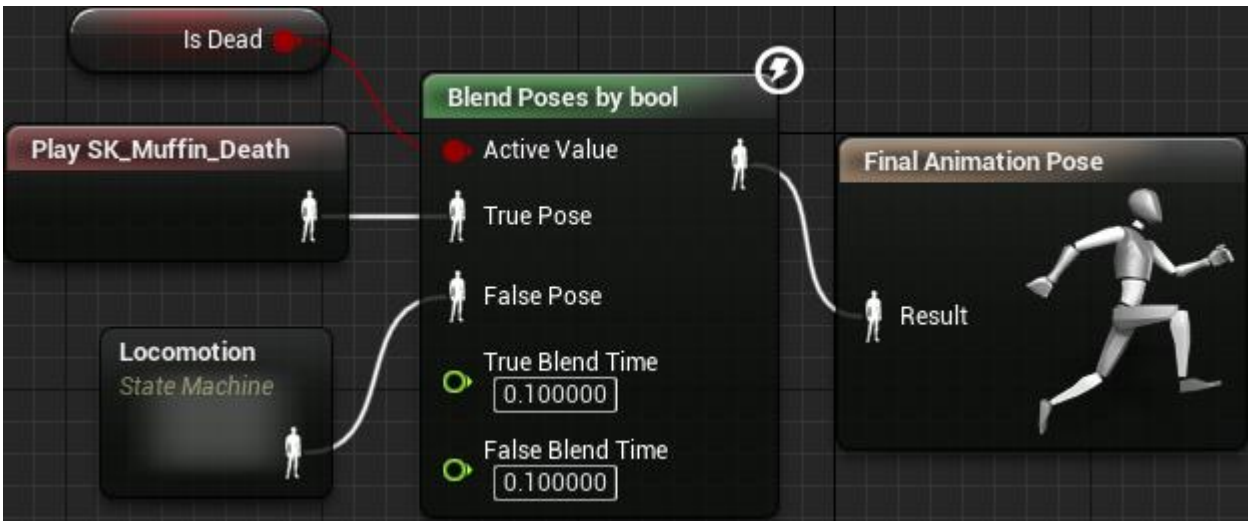
Затем создайте нод *Blend Poses by bool*.

Выберите нод *Blend Poses by bool* и перейдите в панель Details. В разделе *Option поставьте флажок* на свойстве *Reset Child on Activation*.



Поскольку анимация смерти проигрывается всего один раз, эта опция гарантирует, что анимация сбрасывается перед воспроизведением.

Наконец, добавьте переменную *IsDead* и соедините всё следующим образом:



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

Нажмите на *Compile* и закройте *ABP_Muffin*. Нажмите на *Play* и протестируйте новую анимацию смерти!