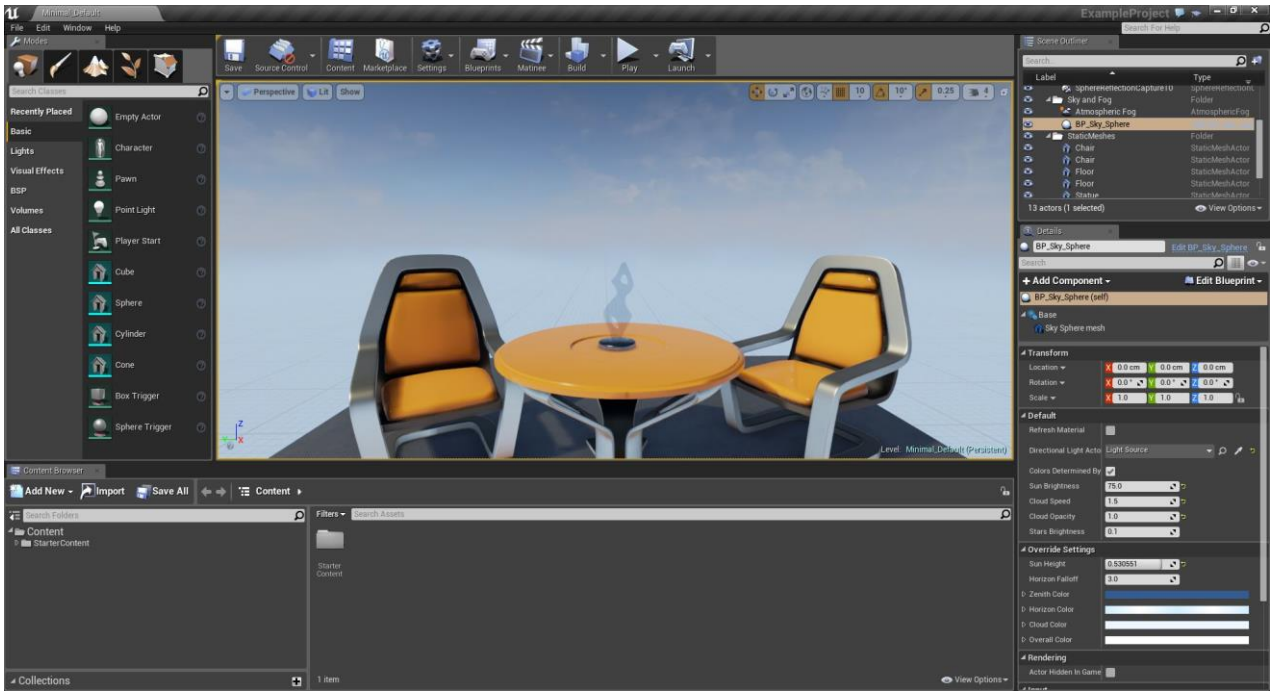
Туториал по Unreal Engine. Часть 5: Как создать простую игру



Если вы новичок в разработке игр, то логичнее всего начинать с создания простой игры. Она научит вас реализации простых механик и тому, как объекты взаимодействуют друг с другом.

В этой части туториала мы создадим игру от первого лица, которая длится бесконечно. Вы научитесь следующему:

- Бесконечно перемещать игрока вперёд
- Генерировать препятствия, которые игрок должен избегать
- Рандомизировать препятствия для создания вариаций
- Создавать кнопку перезапуска, которая отображается, когда игрок сталкивается с препятствием

Приступаем к работе

Скачайте https://koenig-media.raywenderlich.com/uploads/2017/08/InfiniteMatrixStarterProject.zip и распакуйте её. Перейдите в папку проекта и откройте *InfiniteMatrix.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 проверить управление движением. *Двигая мышью*, можно перемещаться вертикально и горизонтально.

Заголовок спойлера

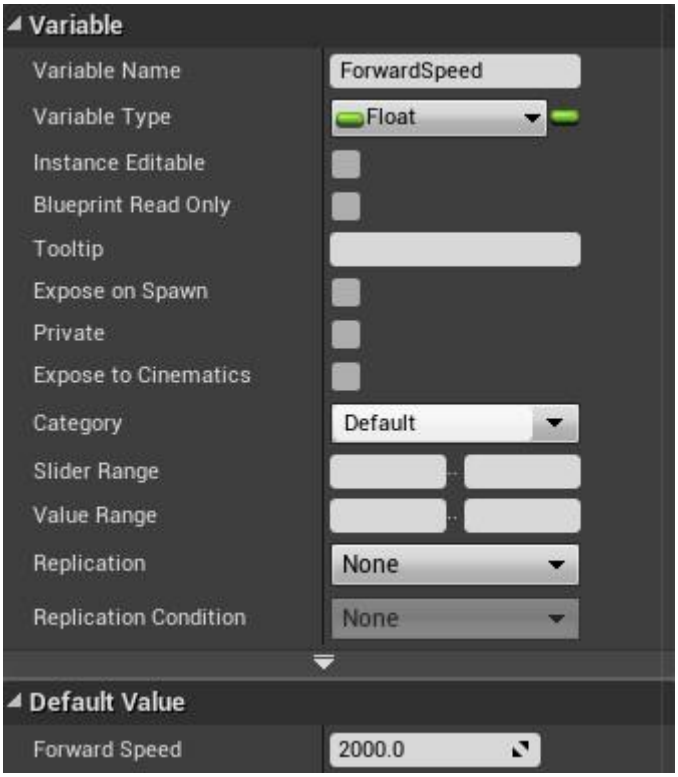
Первое, что нам нужно сделать — заставить игрока постоянно двигаться.

Движение игрока вперёд

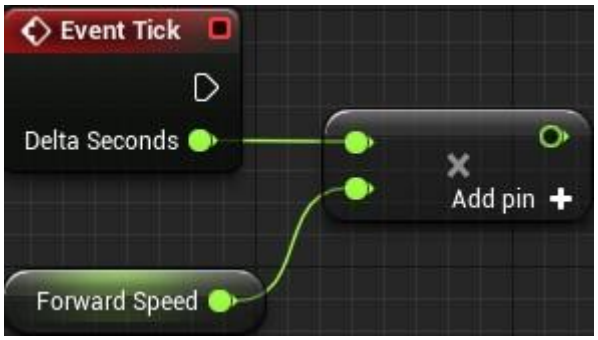
Перейдите в папку *Blueprints* и откройте *BP_Player*.

Чтобы двигать игрока вперёд, нужно добавлять в каждом кадре к местоположению игрока смещение.

Во-первых, нам нужно создать переменную, задающую скорость движения игрока вперёд. Создайте переменную *Float* по названием *ForwardSpeed* и задайте значение по умолчанию *2000*.



Теперь перейдите в Event Graph и найдите нод *Event Tick*. Создайте следующую схему:



Умножая *ForwardSpeed* на *Delta Seconds*, мы получим *независимый от частоты кадров* результат.

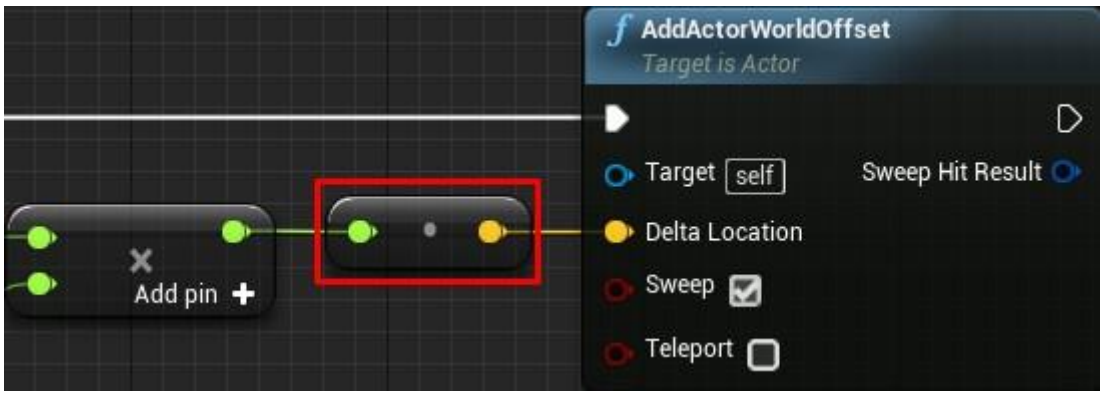
Примечание: если вам непонятно, что такое независимость от частоты кадров, то прочитайте часть туториала про Blueprints. Мы разбираем её в разделе *Независимость от частоты кадров*.

Теперь нам нужно использовать этот результат, чтобы перемещать игрока по одной оси.

Движение вдоль одной оси

Чтобы перемещать игрока, создайте нод *AddActorWorldOffset*. Измените значение *Sweep* на *true*, нажав на *флажок*.

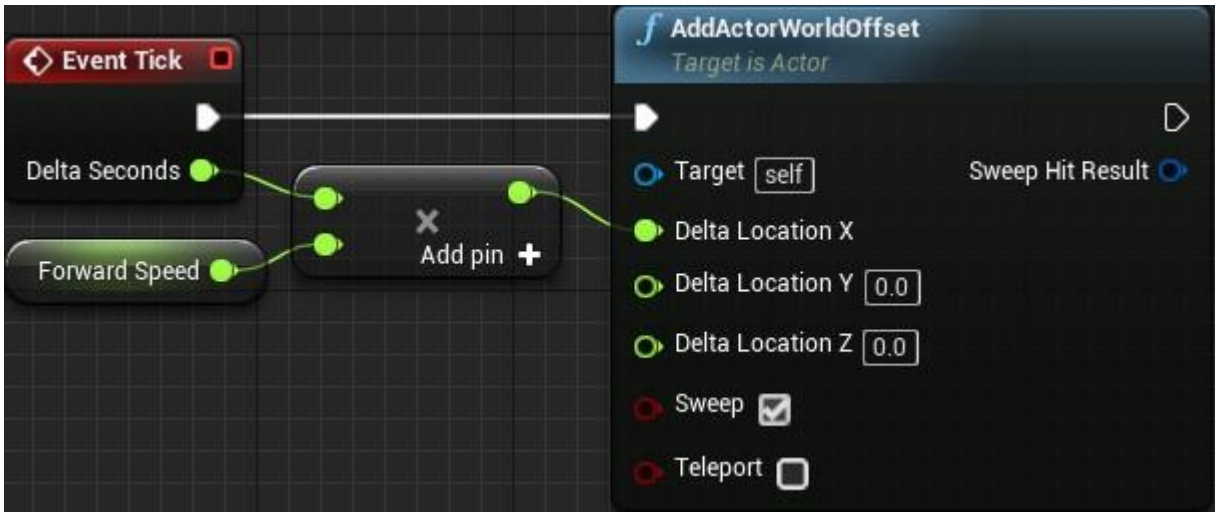
Если попробовать соединить результат *Float* с входом *Delta Location*, то Unreal автоматически преобразует его в *Vector*.



Однако таким образом значение Float будет записано в компоненты X, Y и Z вектора. Для нашей игры движение вперёд должно выполняться только вдоль оси *X*. К счастью, можно разделить Vector на три компонента *Float*.

Убедитесь, что контакт *Delta Location* нода *AddActorWorldOffset* ни с чем не соединён. *Нажмите правой клавишей мыши* на контакт *Delta Location* и выберите *Split Struct Pin*.

Наконец, соедините всё следующим образом:



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

Нажмите на *Compile* и вернитесь в основной редактор. Если нажать *Play*, то вы начнёте двигаться сквозь туннель.

Заголовок спойлера

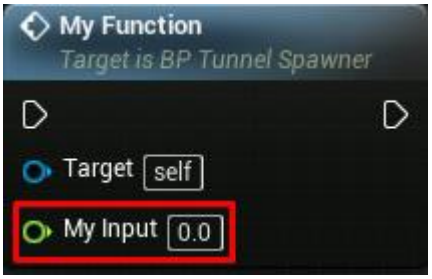
Вместо расположения туннелей вручную мы можем создать Blueprint, автоматически спаунящий туннели.

Создание системы спауна туннелей

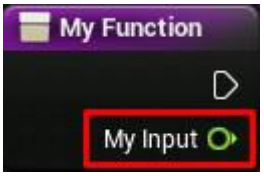
Перейдите в Content Browser и зайдите в папку *Blueprints*. Создайте новый *Blueprint Class* с родительским классом *Actor*. Назовите его *BP_TunnelSpawner* и откройте его.

Игра будет создавать туннели постоянно, поэтому неплохо было бы создать *функцию* для спауна. Перейдите в панель My Blueprint и создайте новую функцию *SpawnTunnel*. Задачей функции будет спаун туннеля в указанном местоположении.

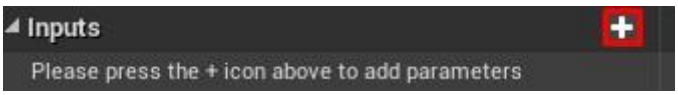
Чтобы передавать функции местоположение, функции нужен *входной параметр*. Они будут отображаться при вызове функции как входные контакты.



Также они будут отображаться как выходные контакты нода *Entry* функции.

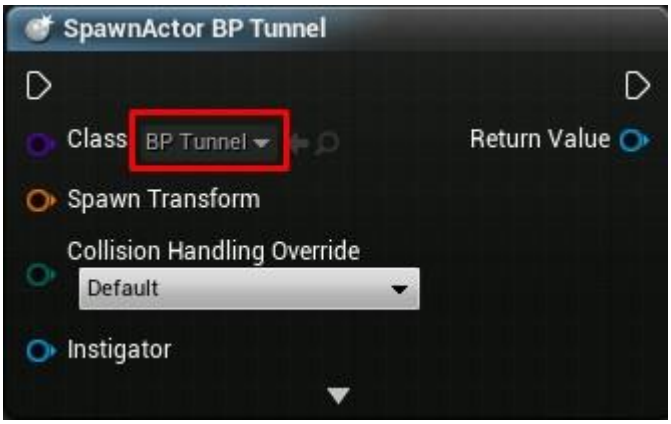


Давайте создадим входной параметр. Перейдите в граф функции *SpawnTunnel*. Выберите нод *Entry* и перейдите в панель Details. Нажмите на значок + рядом с разделом *Inputs*.

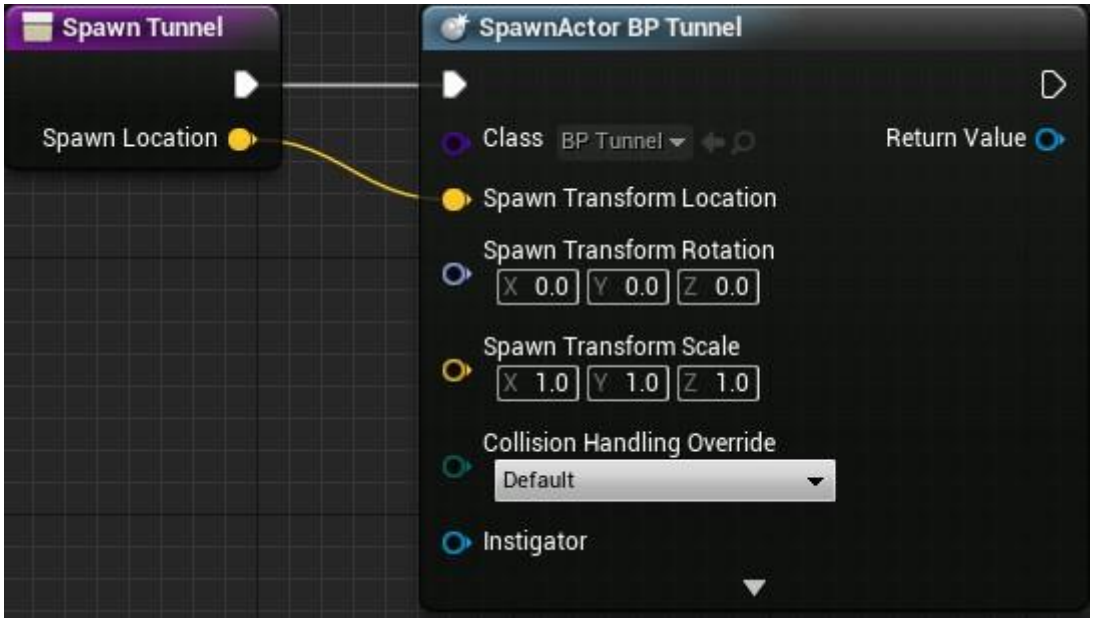


Переименуйте входной параметр в *SpawnLocation* и смените его тип на *Vector*.

Чтобы создать туннель, добавьте нод *Spawn Actor From Class*. Нажмите на *раскрывающийся список*, расположенный справа от контакта *Class* и выберите *BP_Tunnel*.



Чтобы задать местоположение спауна, *нажмите правой клавишей* на контакт *Spawn Transform* и выберите *Split Struct Pin*. Затем соедините нод *Spawn Actor From Class* с нодом *Entry*:



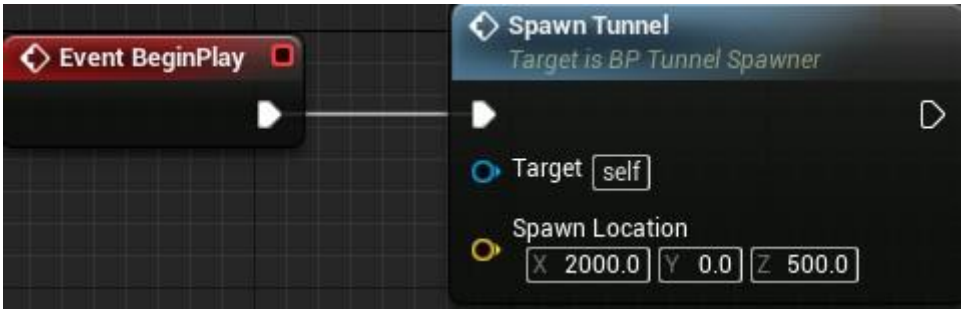
Теперь при вызове функции *SpawnTunnel* она будет спаунить экземпляр *BP_Tunnel* в переданном ей местоположении.

Давайте проверим, как это работает!

Проверка системы спауна туннелей

Переключитесь на Event Graph и найдите нод *Event BeginPlay*. Добавьте нод *SpawnTunnel* и соедините его с нодом *Event BeginPlay*.

В ноде *SpawnTunnel* задайте *Spawn Location* значение *(2000, 0, 500)*.



Теперь при запуске игры он будет спаунить туннель сверху и перед игроком. Нажмите на *Compile* и вернитесь в основной редактор.

Сначала удалите *BP_Tunnel* из уровня. Для этого *щёлкните* на *BP_Tunnel* в World Outliner. Затем нажмите на клавишу *Delete*, чтобы удалить его с уровня.

Затем перейдите в Content Browser. *Перетащите левой клавишей мыши BP_TunnelSpawner* во Viewport. Благодаря этому его экземпляр появится на уровне.

Если нажать на *Play*, то игра будет спаунить туннель сверху и перед игроком.

Заголовок спойлера

Закончив проверку, вернитесь к *BP_TunnelSpawner*. Сбросьте значения *Spawn Location* нода *SpawnTunnel* на *(0, 0, 0)*.

После этого нажмите на *Compile* и вернитесь в основной редактор.

В следующем разделе мы настроим работу *BP_Tunnel*.

Настройка Blueprint туннеля

BP_Tunnel будет выполнять две задачи. Во-первых, он будет определять, когда игра должна создать новый туннель. Для этого мы создадим зону триггера. После срабатывания триггера *BP_Tunnel* будет сообщать *BP_TunnelSpawner*, что нужно создать новый туннель. Благодаря этому можно создать иллюзию бесконечного туннеля.

Заголовок спойлера

Во-вторых, он будет задавать точку спауна. После этого *BP_TunnelSpawner* будет использовать эту точку как следующее местоположение спауна.

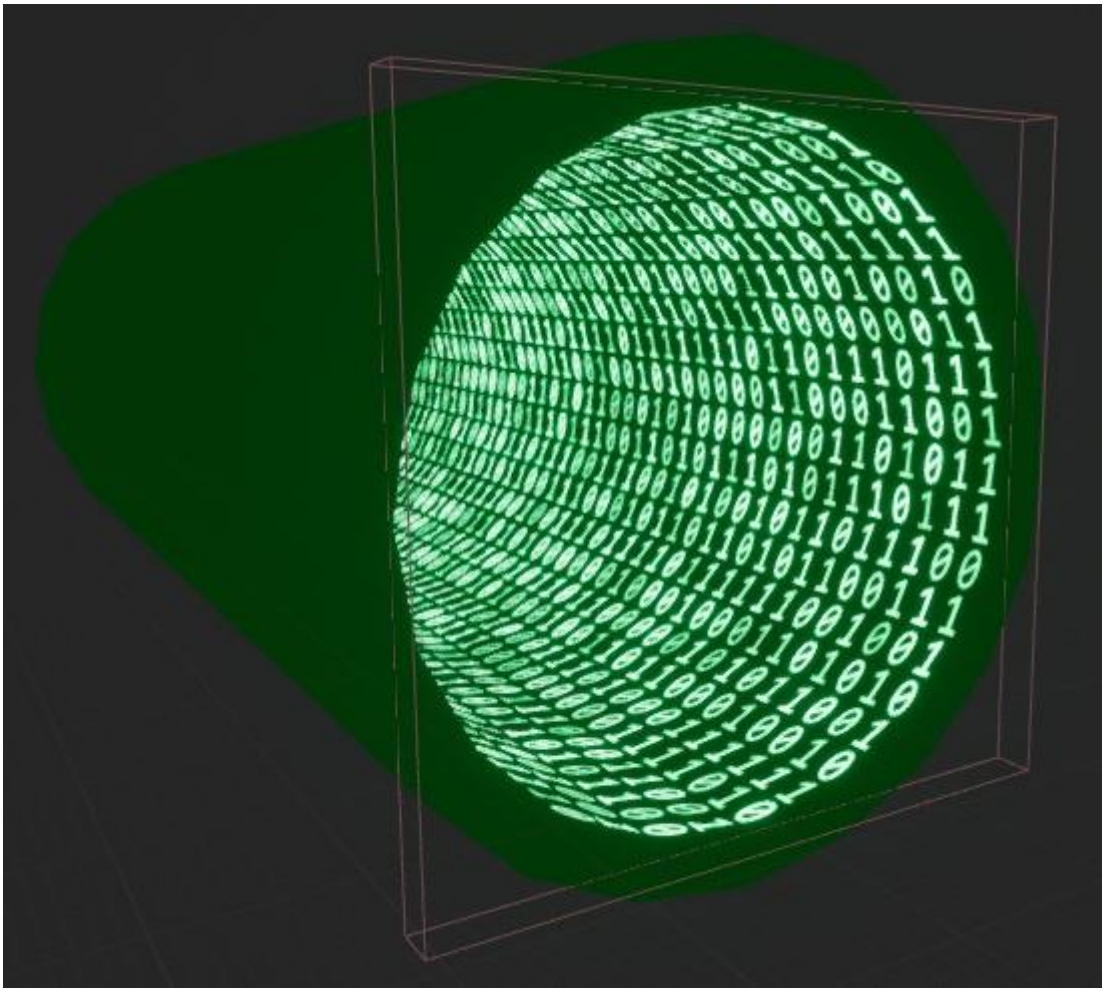
Давайте начнём с создания зоны триггера.

Создание зоны триггера

Откройте *BP_Tunnel* и перейдите в панель Components. Добавьте компонент *Box Collision* и назовите его *TriggerZone*.

Пока область коллизии довольно мала. Перейдите в панель Details и найдите раздел *Shape*. Задайте свойству *Box Extent* значения *(32, 500, 500)*.

Теперь задайте свойству *Location* значения *(2532, 0, 0)*. Это разместит *TriggerZone* прямо в конец меша туннеля. Это значит, что новый туннель должен будет создаваться только когда игрок достигнет *конца* туннеля.

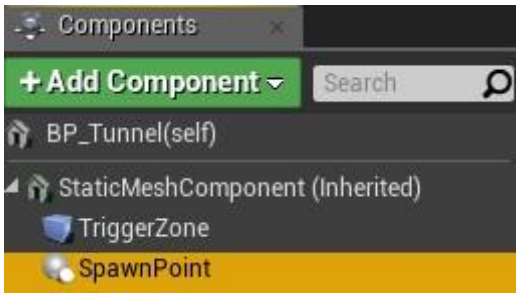


Теперь настало время создать точку спауна.

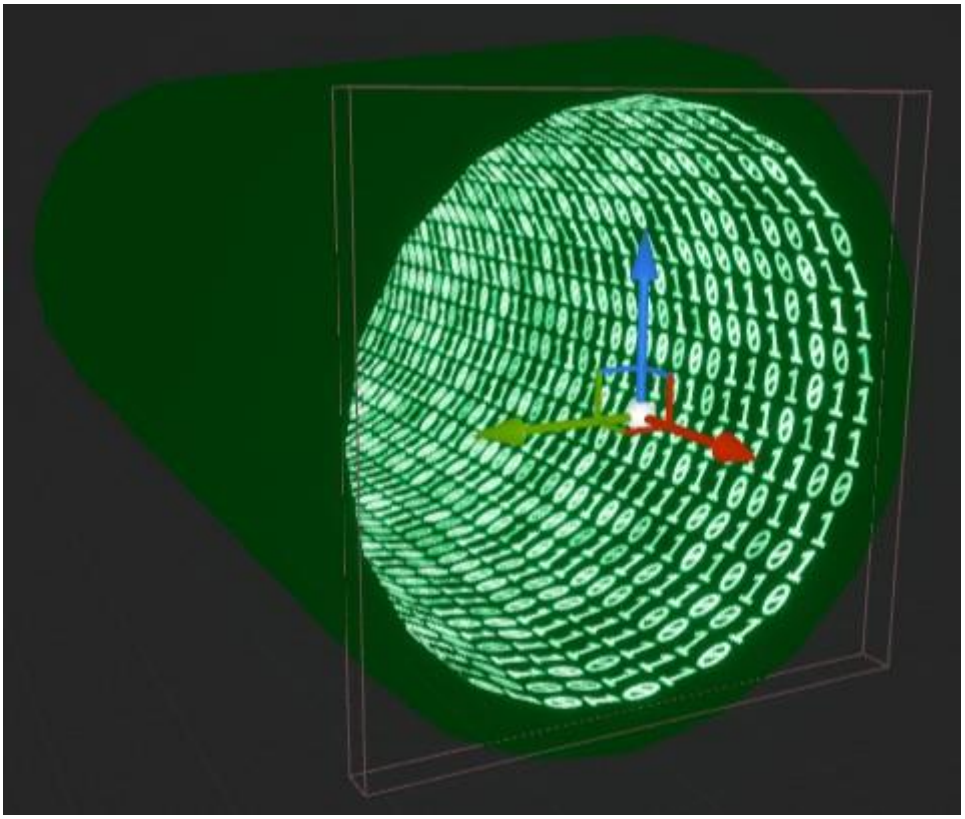
Создание точки спауна

Для задания местоположения точки спауна можно использовать компонент *Scene*. Эти компоненты идеально подходят для задания местоположений, потому что у них есть только Transform. Кроме того, они видимы во Viewport, поэтому вы можете видеть, где находится точка спауна.

Перейдите в панель Components и убедитесь, что ничего не выбрано. Добавьте компонент *Scene* и переименуйте его в *SpawnPoint*.



Меш туннеля имеет длину *2500* единиц по оси *X*, поэтому именно здесь должна быть точка присоединения. Перейдите в панель Details и задайте свойству *Location* значения *(2500, 0, 0)*.



Следующее, что нужно сделать — создать функцию, которая спаунит туннель в *SpawnPoint*.

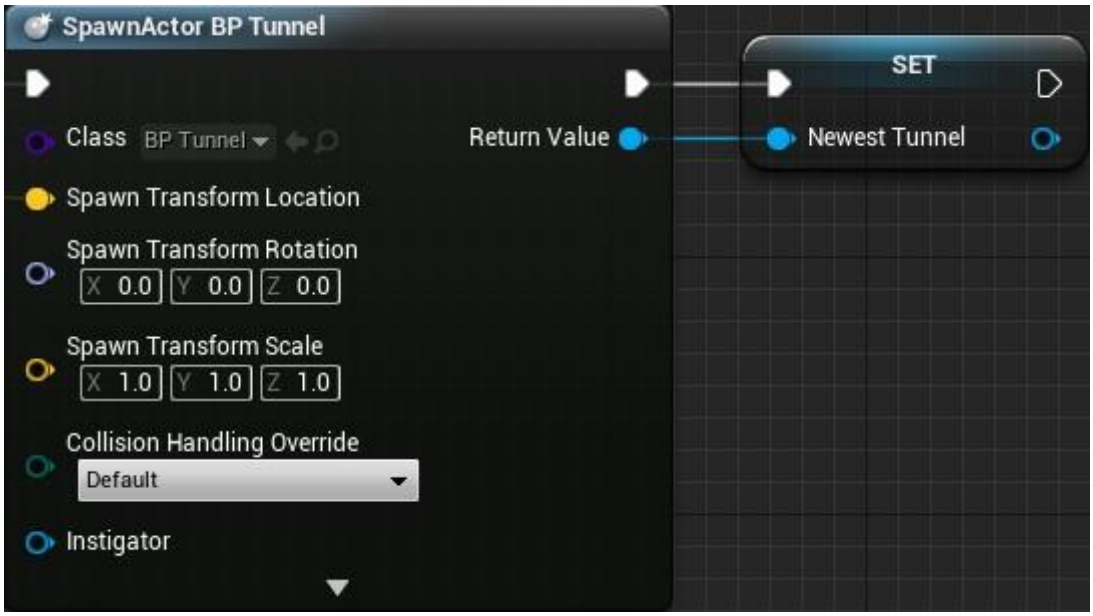
Создание туннелей в точке спауна

Нажмите на *Compile* и переключитесь на *BP_TunnelSpawner*.

Следующий *BP_Tunnel* должен спауниться в *SpawnPoint самого дальнего* туннеля. Благодаря этому туннель всегда будет продолжаться.

Поскольку самый дальний туннель всегда будет последним созданным, то нам легко получить ссылку на него.

Откройте граф *SpawnTunnel*. *Нажмите правой клавишей мыши* на контакте *Return Value* нода *Spawn Actor From Class*. Выберите *Promote to Variable* и переименуйте переменную в *NewestTunnel*.



Теперь у нас всегда будет ссылка на самый дальний туннель.

Далее создайте новую функцию и назовите её *SpawnTunnelAtSpawnPoint*. Создайте следующий граф:

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

Чтобы *BP_Tunnel* мог обмениваться данными с *BP_TunnelSpawner*, ему нужна ссылка. Без передачи данных *BP_TunnelSpawner* не будет знать, где спаунить следующий туннель.

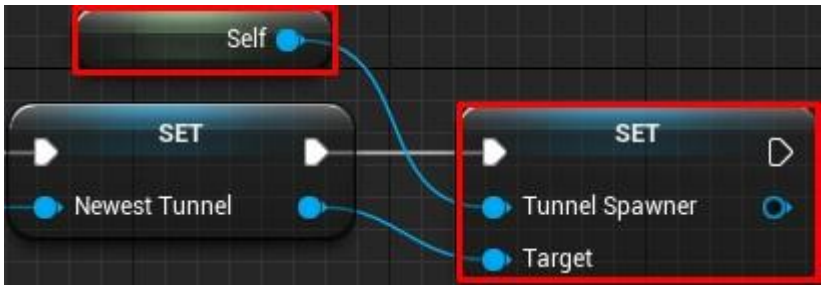
Создание ссылки на спаунер туннелей

Нажмите на *Compile* и закройте граф *SpawnTunnelAtSpawnPoint*. Переключитесь на *BP_Tunnel*.

Добавьте новую переменную и назовите её *TunnelSpawner*. В качестве *Variable Type* выберите *BP_TunnelSpawner\Object Reference*.

Нажмите на *Compile* и переключитесь на *BP_TunnelSpawner*.

Откройте граф *SpawnTunnel* и добавьте обозначенные ноды:



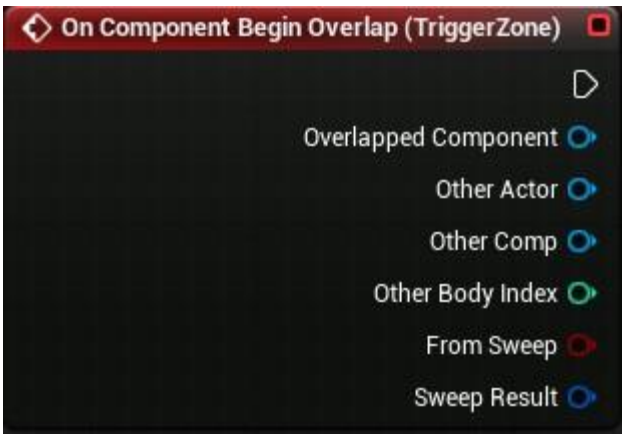
Теперь у каждого туннеля будет ссылка на *BP_TunnelSpawner*.

Далее нужно сообщить *BP_TunnelSpawner* о необходимости спауна следующего туннеля, когда игрок входит в *TriggerZone*.

Скриптинг зоны триггера

Нажмите на *Compile* и переключитесь на *BP_Tunnel*.

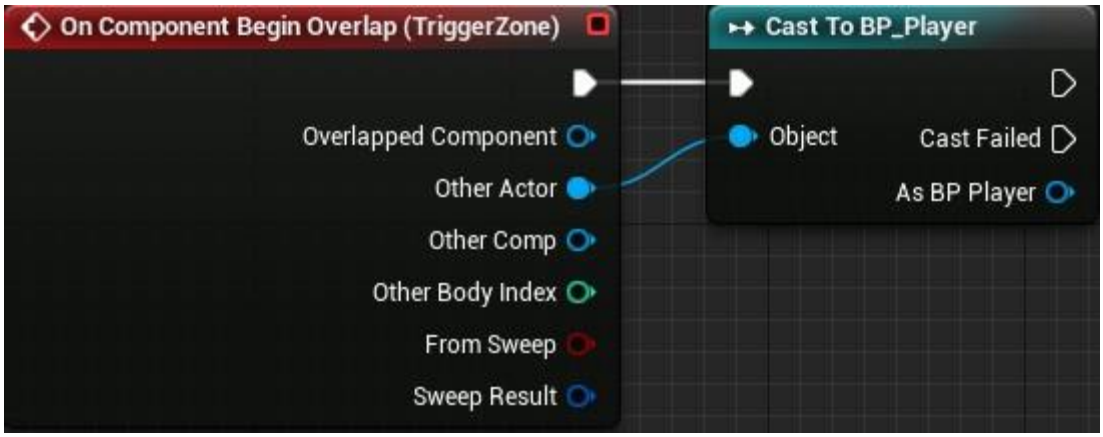
Перейдите в панель Components и *нажмите правой клавишей* на *TriggerZone*. Выберите *Add Event\Add OnComponentBeginOverlap*. Это добавит в Event Graph следующий нод:



Этот нод будет выполняться когда другой *Actor* касается *TriggerZone*.

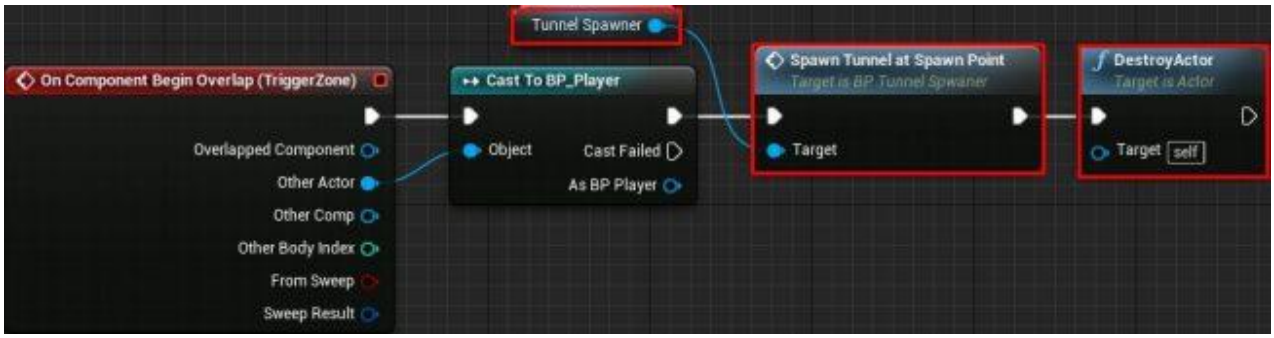
Сначала нам нужно проверить, является ли *Actor*, касающийся *TriggerZone*, игроком.

Перетащите контакт *Other Actor*. Отпустите *левую клавишу мыши* в пустом пространстве и выберите в меню *Cast to BP_Player*.



Примечание: туннель спаунится в *конце* другого туннеля, поэтому он приведёт к срабатыванию триггера *TriggerZone* этого туннеля. *Cast to BP_Player* предотвращает выполнение всех дальнейших нодов, если *Other Actor* является туннелем.

Теперь добавим обозначенные ноды после нода *Cast to BP_Player*:



Давайте посмотрим, что здесь происходит пошагово:

1. Когда *Actor* касается *TriggerZone*, выполняется нод *On Component Begin Overlap (TriggerZone)*

2. Нод *Cast to BP_Player* проверяет, является ли касающийся актор игроком
3. Если это игрок, то *BP_TunnelSpawner* создаст новый туннель. Его местоположение будет находится в компоненте *SpawnPoint* последнего созданного туннеля.
4. Поскольку старый туннель уже не нужен, игра удаляет его с помощью нода *DestroyActor*

Нажмите на *Compile*, вернитесь в основной редактор и нажмите *Play*. Когда вы достигнете конца туннеля, игра создаст новый.

Заголовок спойлера

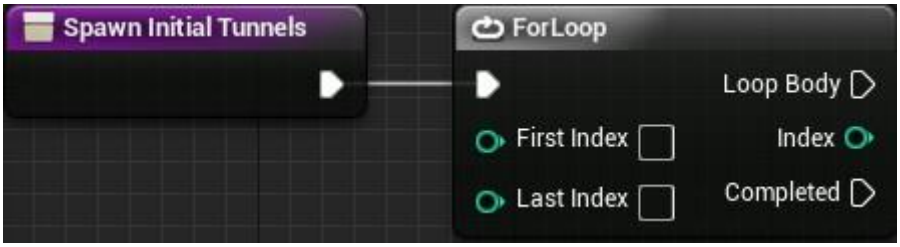
Хотя игра бесконечно создаёт туннели, они не *выглядят* бесконечными. Исправить это можно, сделав видимыми несколько туннелей. Позже, когда мы соединим их с препятствиями, игрок не сможет увидеть, как создаются туннели.

Создание нескольких туннелей

Первое, что нужно сделать — создать функцию, создающую определённое количество туннелей.

Откройте *BP_TunnelSpawner* и создайте новую функцию с названием *SpawnInitialTunnels*.

Чтобы спаунить определённое количество туннелей, можно использовать нод *ForLoop*. Этот нод будет выполнять соединённые с ним ноды указанное количество раз. Добавьте нод *ForLoop* и соедините его с нодом *Entry*.



Чтобы нод *ForLoop* выполнялся *n* раз, необходимо задать для *Last Index* значение *n – 1*.

В этом туториале мы будем спаунить *три* туннеля. Для выполнения трёх циклов задайте *Last Index* значение *2*.

Примечание: если не задать поля *First Index* или *Last Index*, то по умолчанию они будут равны *0*.
Когда игра начинается, игрок всегда должен находиться в туннеле. Для этого можно создать первый туннель в местоположении игрока.

Создание первого туннеля

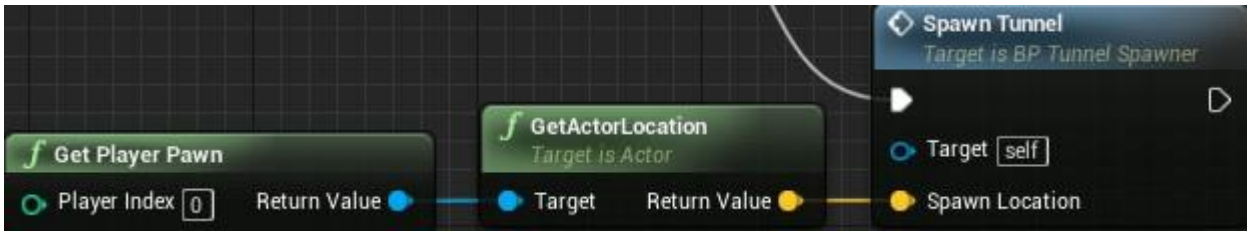
Чтобы определить, был ли создан первый туннель, мы можем проверить, задано ли *NewestTunnel*. Если нет, то это значит, что первый туннель ещё не создан, потому что *NewestTunnel* задаётся только *после* того, как игра создаст туннель.

Чтобы выполнить эту проверку, добавим нод *IsValid* (на котором есть знак вопроса) после нода *ForLoop*.

Затем получим ссылку на *NewestTunnel* и соединим её с контактом *Input Object* нода *IsValid*.

Если *NewestTunnel* не задан, то будет выполнен нод *Is Not Valid*, и наоборот.

Добавьте следующее и соедините ноды с контактом *Is Not Valid* нода *IsValid*:

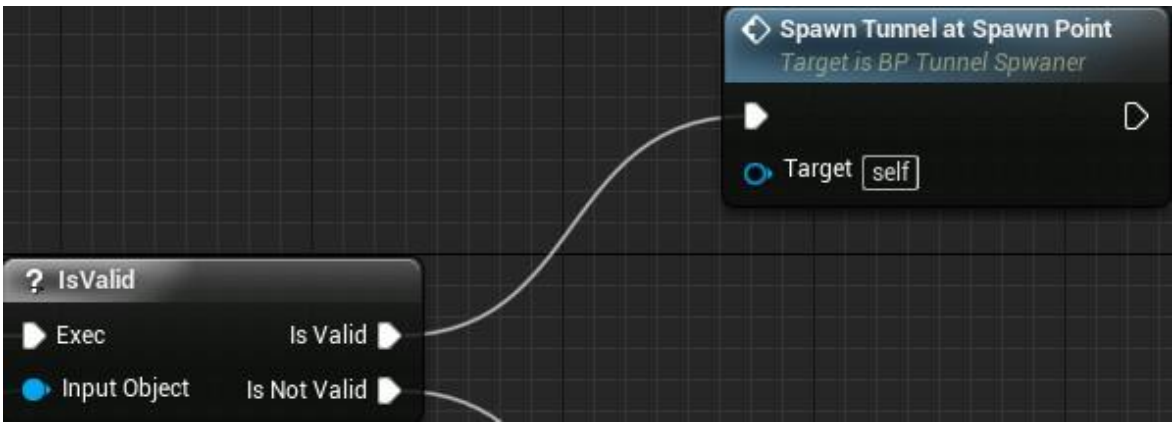


Эта схема будет создавать туннель в местоположении Pawn игрока.

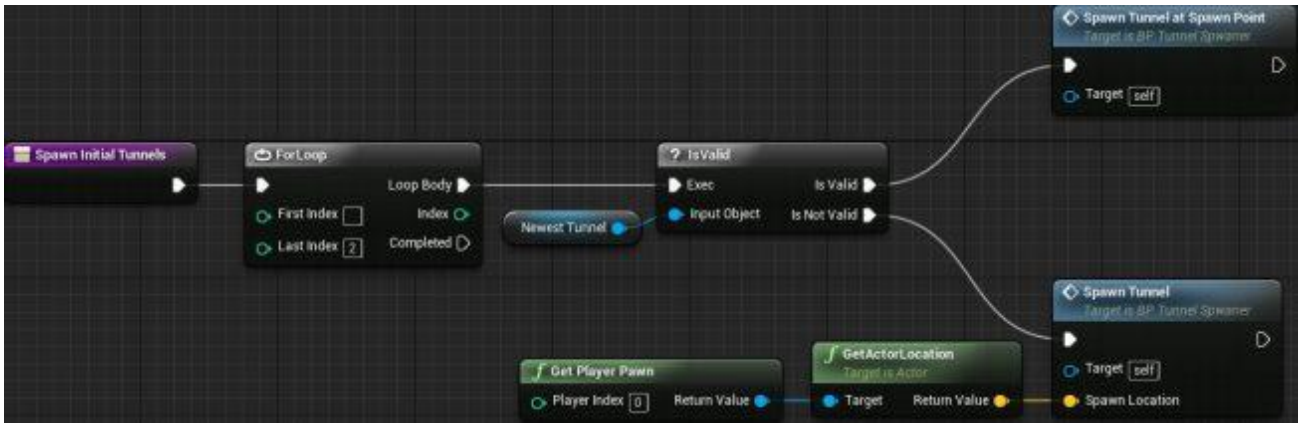
Затем нам нужно создать следующие туннели.

Создание следующих туннелей

Добавьте нод *SpawnTunnelAtAttachPoint* и соедините его с контактом *Is Valid* нода *IsValid*.



Вот как выглядит готовый граф:



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

Теперь перейдите в Event Graph и удалите нод *SpawnTunnel*. Добавьте нод *SpawnInitialTunnels* после *Event BeginPlay*.

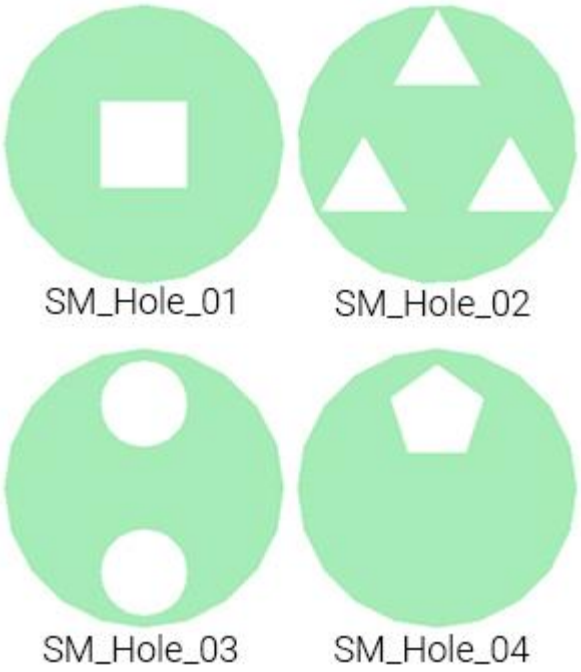
Нажмите на *Compile*, вернитесь в основной редактор и нажмите на *Play*. Теперь туннель стал гораздо длиннее!

Заголовок спойлера

Игра пока не очень сложна, так что давайте добавим препятствия.

Создание препятствий

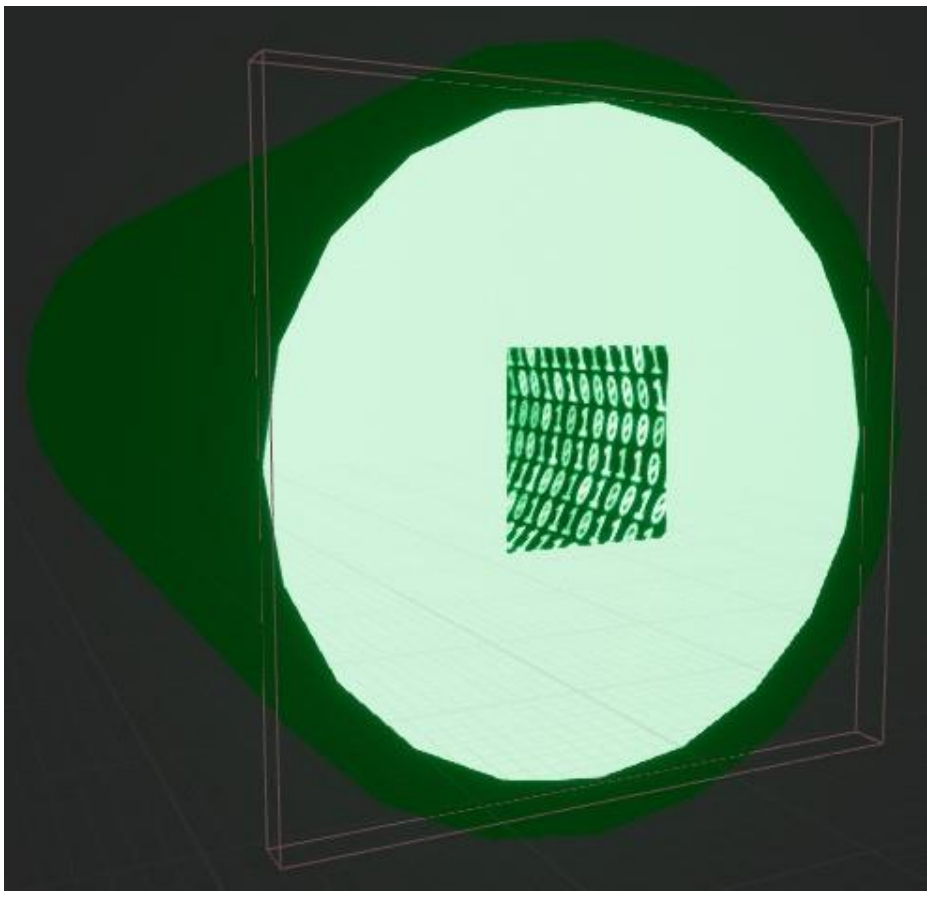
Вот меши, которые мы будем использовать как препятствия:



Откройте *BP_Tunnel* и перейдите в панель Components. Добавьте компонент *Static Mesh* и назовите его *WallMesh*.

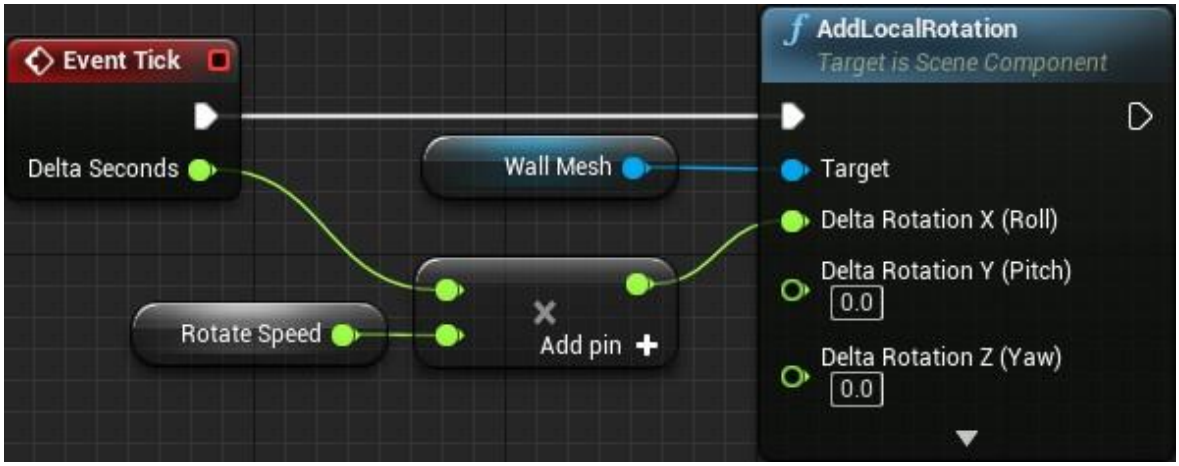
Перейдите в панель Details и измените его свойство *Static Mesh* на *SM_Hole_01*.

Затем задайте его свойству *Location* значение *(2470, 0, 0)*, расположив его таким образом в конце туннеля.



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

Переключитесь в Event Graph и найдите нод *Event Tick*. Создайте следующую схему:



Это заставит *WallMesh* поворачиваться каждый кадр на заданную величину.

Нажмите на *Compile* и вернитесь к основному редактору. Нажмите на *Play*, чтобы увидеть, как поворачиваются стены.

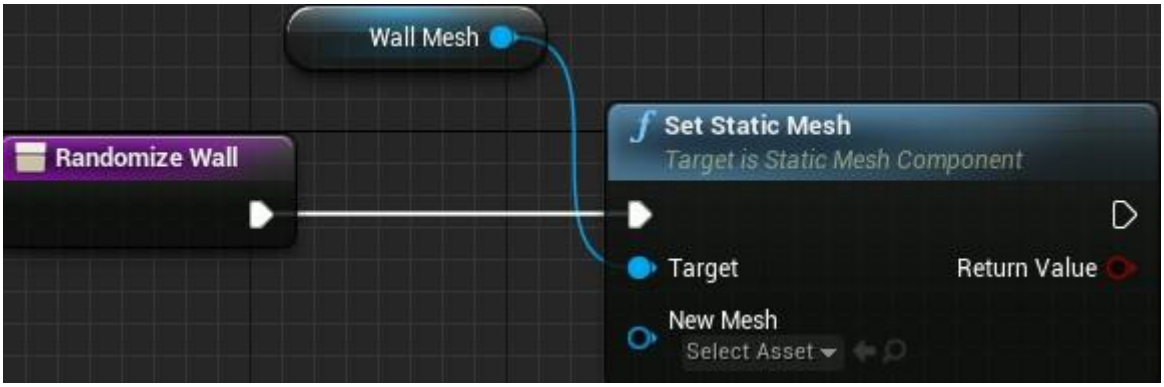
Заголовок спойлера

Давайте сделаем игру ещё интереснее, добавив стенам вариативности.

Создание вариаций стен

Вместо создания нового Blueprint для каждой вариации, мы можем просто рандомизировать *WallMesh*.

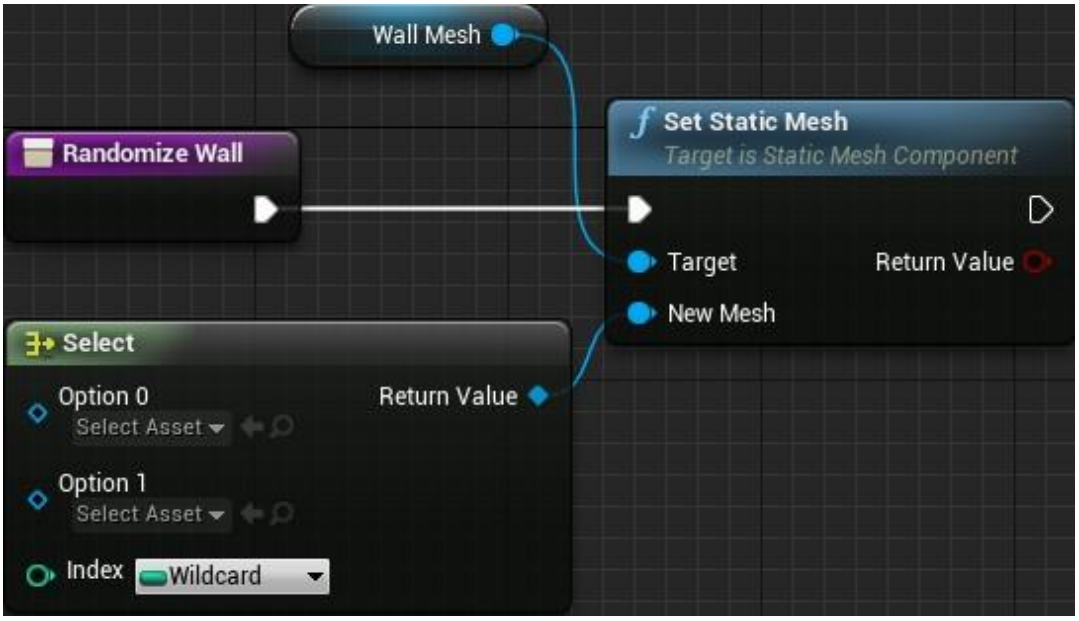
Откройте *BP_Tunnel* и создайте новую функцию под названием *RandomizeWall*. Затем создайте следующий граф:



Как следует из названия, нод *Set Static Mesh* будет задавать *WallMesh* переданный меш.

Чтобы создать список статичных мешей, можно использовать нод *Select*.

Перетащите контакт *New Mesh*. Отпустите *левую клавишу мыши* на пустом пространстве и добавьте нод *Select*.

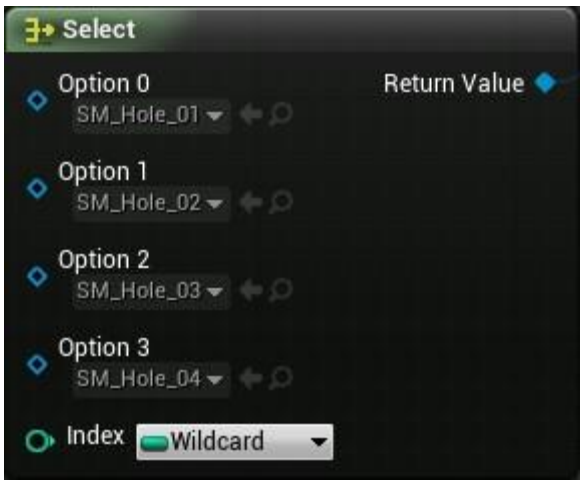


Нод *Select* позволяет задать список вариантов. Вход *Index* определяет, какой вариант будет выводить нод *Select*.

У нас есть четыре меша стен, поэтому нужно создать ещё два контакта *Option*. Это можно сделать *правым щелчком* на ноде *Select*, выбрав *Add Option Pin*. Повторяйте эту операцию, пока у вас не будет *четыре* контакта Option.

Теперь задайте каждому варианту следующие значения:

- *Option 0:* SM_Hole_01
- *Option 1:* SM_Hole_02
- *Option 2:* SM_Hole_03
- *Option 3:* SM_Hole_04

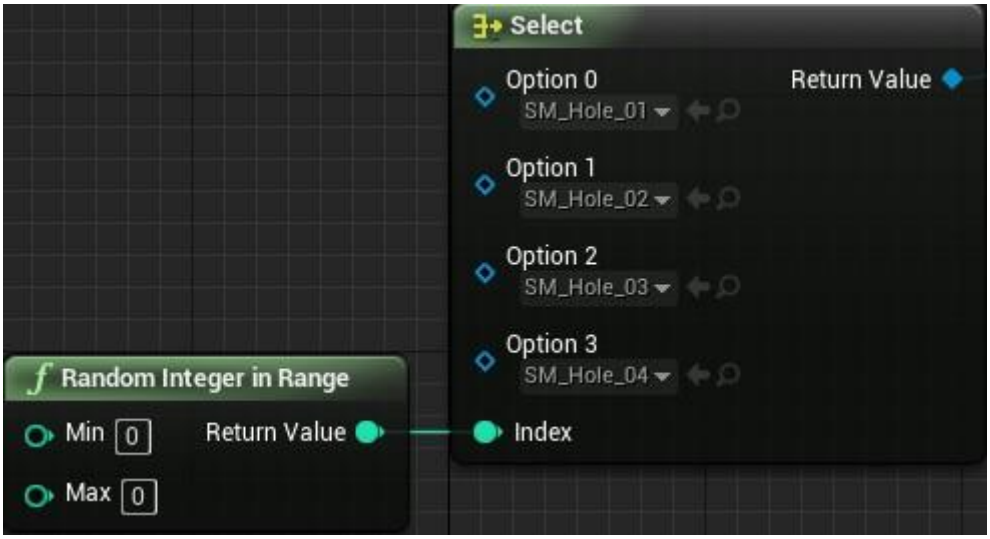


Теперь сделаем так, чтобы выбирался случайный вариант.

Рандомизация стен

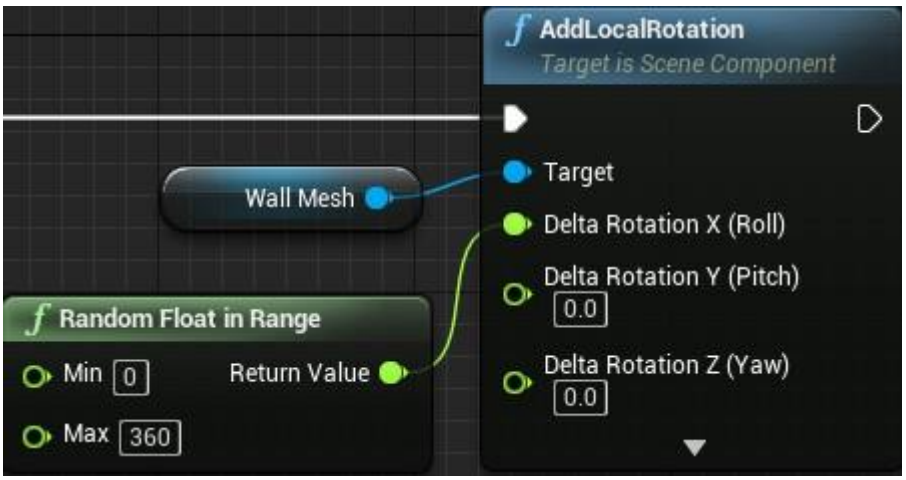
Для получения случайного числа можно использовать нод *Random Integer in Range*. Этот нод возвращает значение >= *Min* и <= *Max*.

Добавьте нод *Random Integer in Range* и соедините его с контактом *Index* нода *Select*.



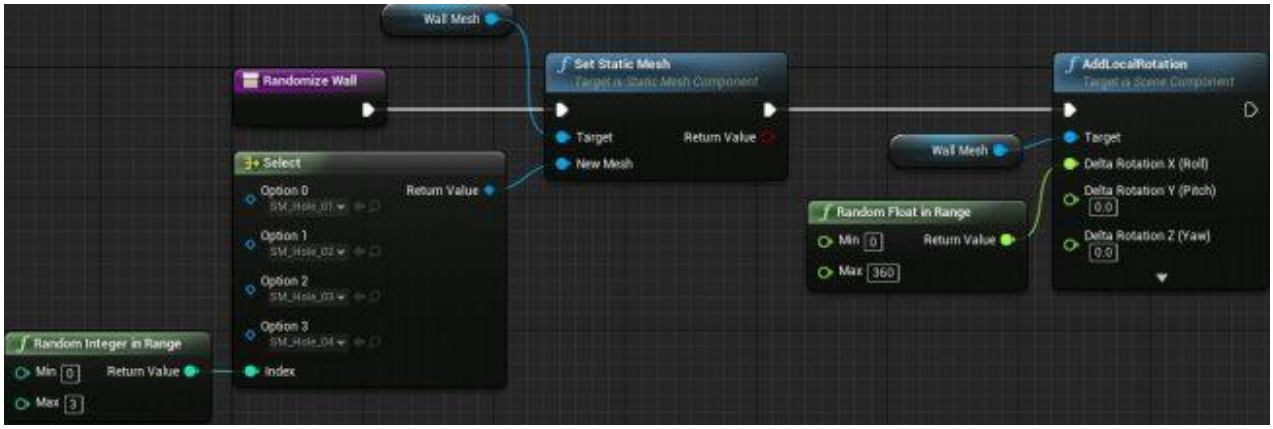
Задайте *Max* значение *3*. Это даст нам четыре возможных числа: 0, 1, 2 и 3.

Чтобы добавить случайности, давайте придадим *WallMesh* случайный поворот. Добавьте следующее после нода *Set Static Mesh*:



Это добавит *WallMesh* случайный поворот в интервале от *0* до *360* градусов.

Вот как выглядит готовый граф:

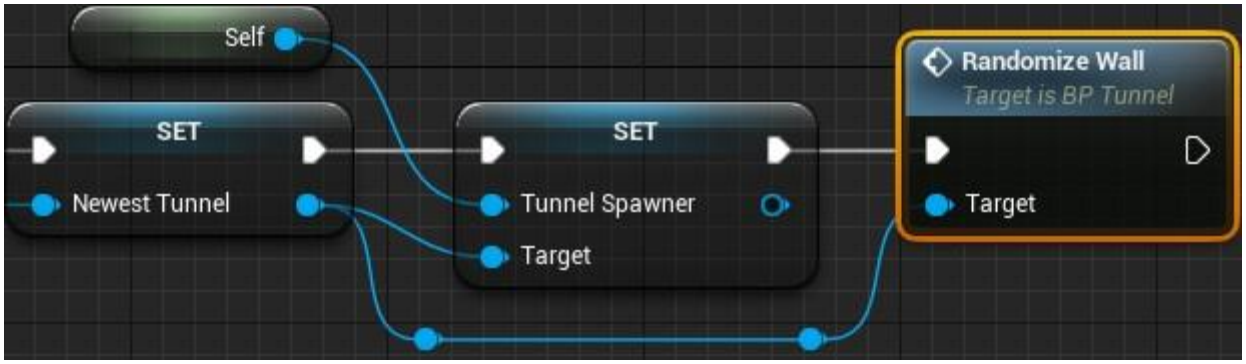


Подведём итог:

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*

Нажмите на *Compile* и закройте граф *RandomizeWall*.

Переключитесь на *BP_TunnelSpawner* и откройте граф *SpawnTunnel*. Добавьте выделенный нод:



Теперь при спауне туннеля у него будет случайный меш стены.

Закройте граф *SpawnTunnel* и нажмите на *Compile*. Вернитесь в основной редактор и нажмите на *Play*, чтобы увидеть вариации стен!

Заголовок спойлера

Если вы столкнётесь со стеной. то перестанете двигаться вперёд. Однако если продолжите двигаться и пройдёте сквозь отверстие, то снова начнёте двигаться вперёд.

Следующим шагом будет отключение движения вперёд после столкновения игрока со стеной.

Обработка коллизий со стенами

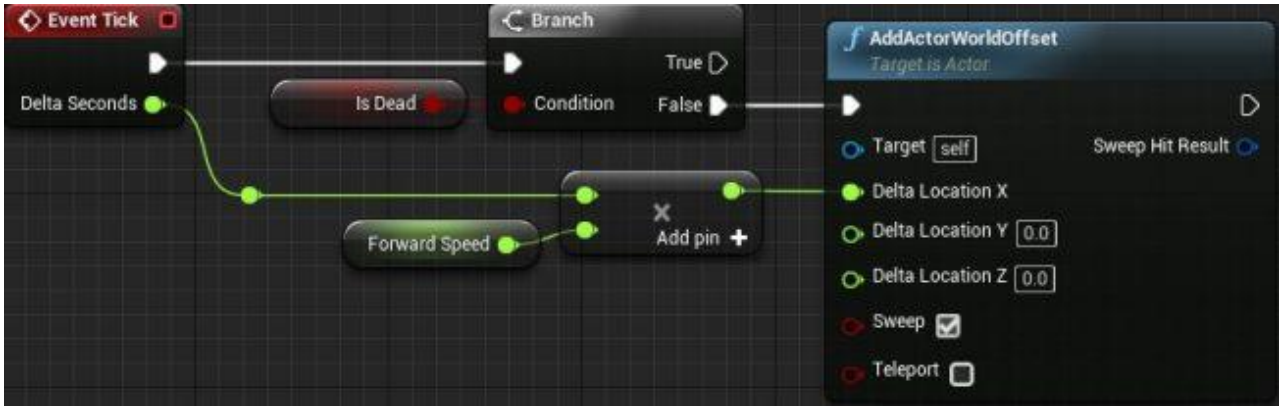
Для включения и отключения движения вперёд можно использовать переменную *Boolean*. Она имеет всего два состояния: *true* и *false*.

Откройте *BP_Player* и создайте новую переменную *Boolean* под названием *IsDead*.

Перейдите к ноду *Event Tick* и создайте нод *Branch*.

Теперь получите ссылку на *IsDead* и соедините его с контактом *Condition* нода *Branch*.

Соедините нод *Event Tick* с нодом *Branch*. Затем соедините контакт *False* нода *Branch* с нодом *AddActorWorldOffset*.



Теперь когда *IsDead* равно *true*, игрок будет останавливаться.

Теперь нужно задавать переменную *IsDead*, когда игрок сталкивается со стеной.

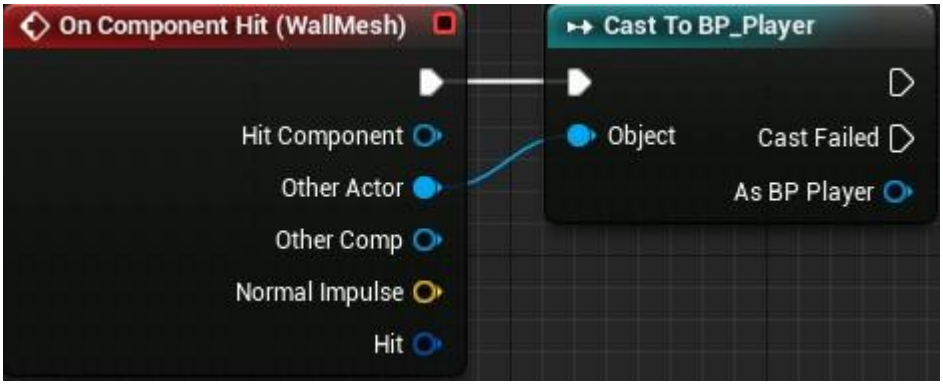
Задание переменной IsDead

Нажмите на *Compile* и переключитесь на *BP_Tunnel*. В панели Components *нажмите правой клавишей мыши* на *WallMesh* и выберите *Add Event\Add OnComponentHit*. Это добавить в Event Graph следующий нод:

Этот нод будет выполняться при столкновении другого *Actor* с *WallMesh*.

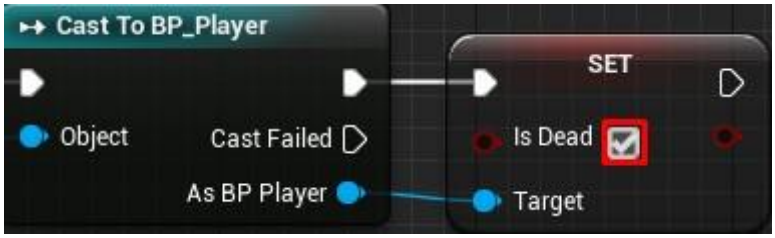
Во-первых, нам нужно проверить, является ли игроком *Actor*, столкнувшийся с *WallMesh*.

Перетащите контакт *Other Actor*. Отпустите *левую клавишу мыши* на пустом пространстве и выберите в меню *Cast to BP_Player*.



Перетащите контакт *BP_Player* нода *Cast to BP_Player*. Отпустите *левую клавишу* на пустом пространстве и добавьте нод *Set Is Dead*.

Задайте *IsDead* значение *true*, поставив *флажок*.



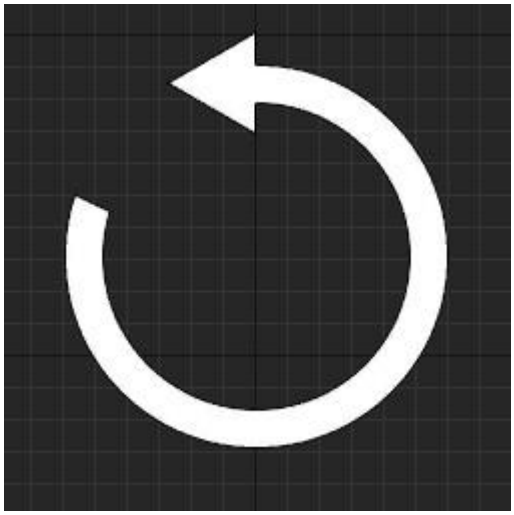
Нажмите на *Compile* и вернитесь к основному редактору. Нажмите на *Play* и попробуйте столкнуться со стеной. Если вы переместитесь к отверстию, то не будете двигаться сквозь него дальше.

Заголовок спойлера

В следующем разделе мы научимся отображать кнопку перезапуска, когда игрок ударяется о стену.

Отображение кнопки перезапуска

Виджет, который мы будем отображать, называется *WBP_Restart*. Его можно найти в папке *UI*. Вот как он выглядит:



Чтобы отобразить или скрыть виджет, нам нужна ссылка на него. Откройте *BP_Player* и создайте новую переменную *RestartWidget*. Измените *Variable Type* на *WBP_Restart\Object Reference*.

Затем перейдите в Event Graph и найдите нод *Event BeginPlay*.

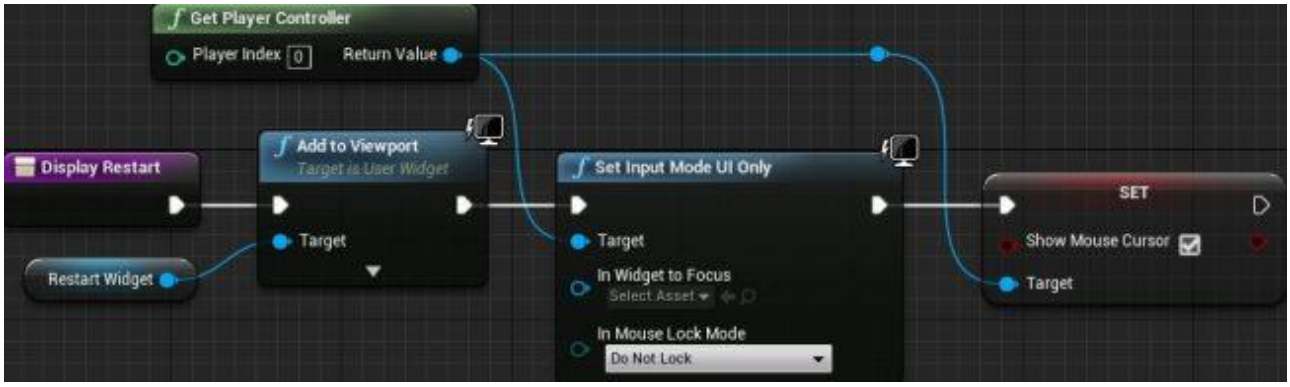
Добавьте нод *Create Widget* и задайте *Class* значение *WBP_Restart*.

Теперь добавьте нод *Set Restart Widget* и соедините всё следующим образом:

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

Создание функции отображения

Создайте новую функцию и назовите её *DisplayRestart*. Сделав это, создайте следующий граф:



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

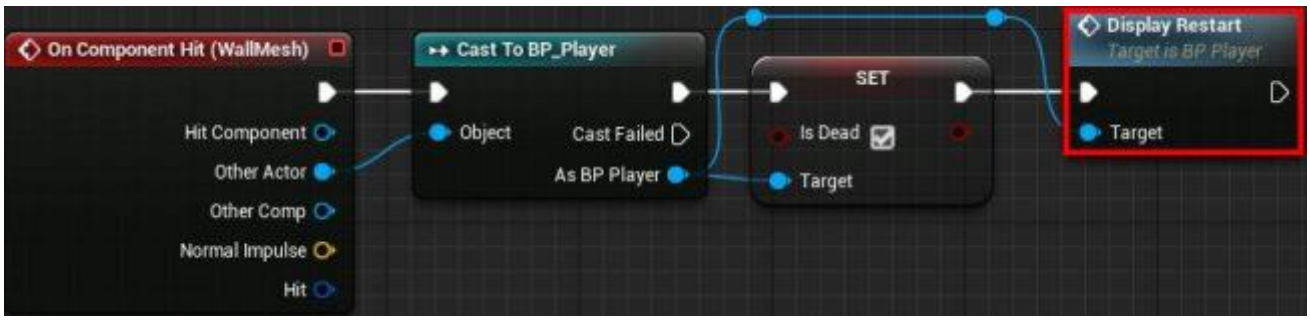
Для отображения кнопки перезапуска нам нужно просто вызывать *DisplayRestart* после столкновения игрока со стеной.

Вызов функции отображения

Закройте граф *DisplayRestart* и нажмите на *Compile*.

Переключитесь на *BP_Tunnel* и найдите нод *On Component Hit (WallMesh).*

Добавьте нод *DisplayRestart* в конец цепочки нодов.



Нажмите на *Compile* и закройте *BP_Tunnel*. Вернитесь к основному редактору и нажмите на *Play*. Когда игрок сталкивается со стеной, отображается кнопка перезапуска.

Заголовок спойлера

Последним шагом будет перезапуск игры при нажатии на кнопку.

Перезапуск игры

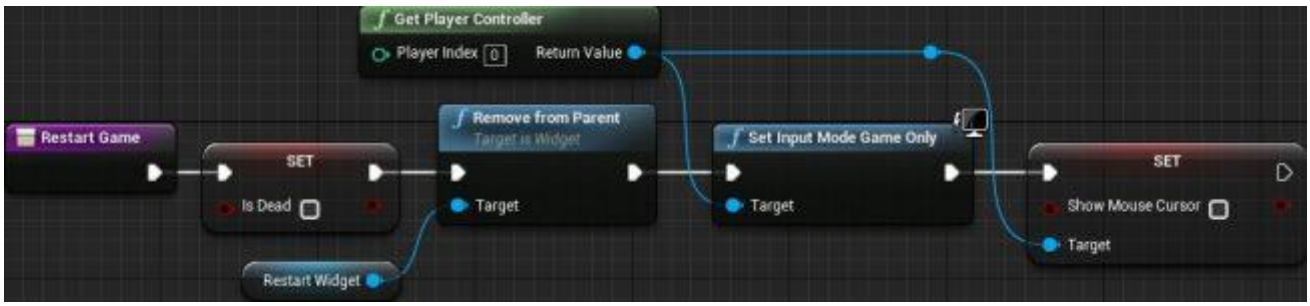
При перезапуске игра должна выполнять два действия:

1. Сбрасывать состояние игрока. Это включает в себя и скрытие кнопки перезапуска с экрана.
2. Повторно создавать туннели. Таким образом игрок стартует с начала туннеля.

Давайте начнём со сброса игрока.

Сброс игрока

Откройте *BP_Player* и создайте новую функцию *RestartGame*. Создайте следующий граф:



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

Теперь нам нужно снова создать туннели.

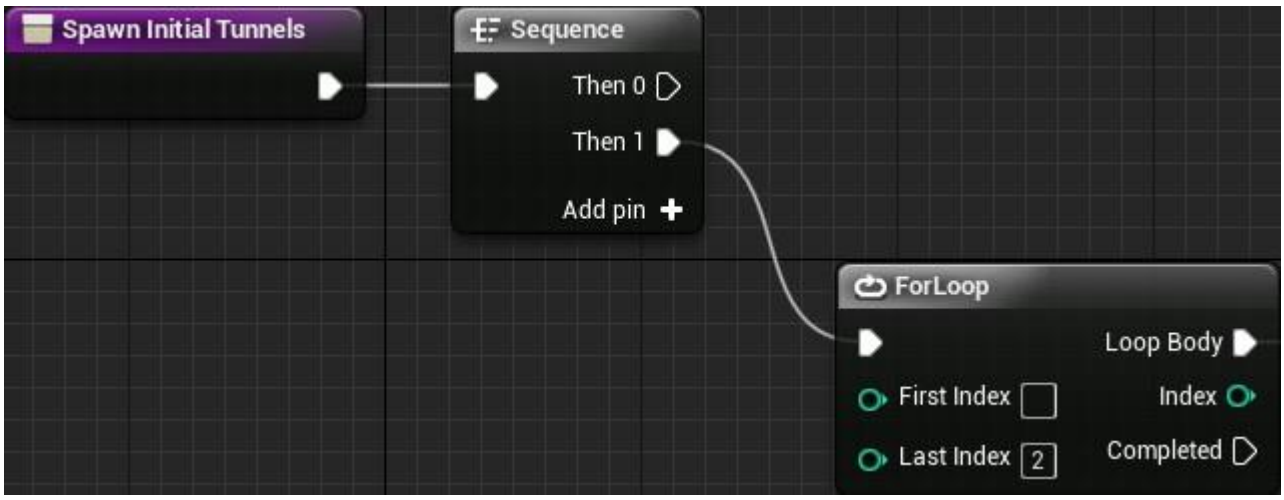
Повторное создание туннелей

Нажмите на *Compile* и закройте *BP_Player*.

Откройте *BP_TunnelSpawner* и перейдите в граф *SpawnInitialTunnels*.

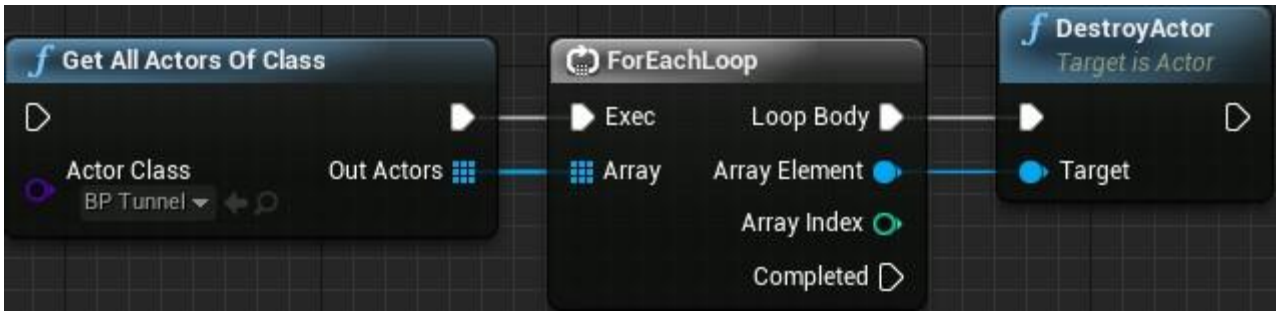
Во-первых, перед созданием новых туннелей нам нужно удалить уже имеющиеся.

Добавьте нод *Sequence* после нода *Entry*. Соедините контакт *Then 1* с нодом *ForLoop*.



Примечание: нод *Sequence* выполняет свои выходы в последовательном порядке. Это отличный способ упорядочить граф вертикально, особенно когда цепочки нодов очень длинные.

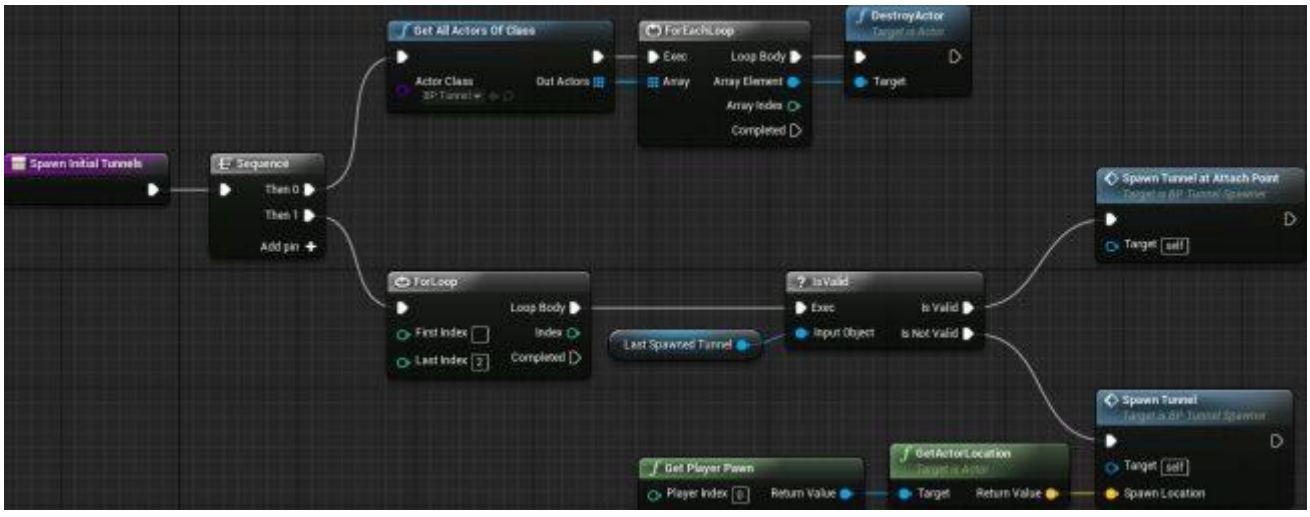
Затем создайте следующие ноды:



Эта схема получит все существующие туннели и удалит их из игры.

Наконец, соединим контакт *Then 0* нода *Sequence* с нодом *Get All Actors of Class*. Благодаря этому туннели удалятся до процесса создания.

Вот как выглядит конечный граф:



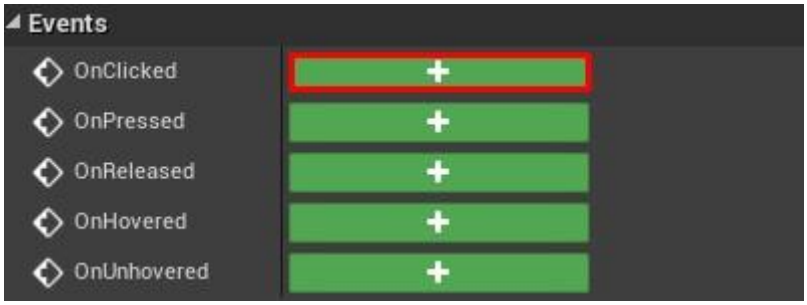
Последнее, что нужно сделать — обработать нажатие на кнопку.

Обработка нажатий на кнопку

Нажмите на *Compile* и закройте *BP_TunnelSpawner*.

Перейдите в Content Browser и найдите папку *UI*. *Дважды щёлкните* на *WBP_Restart*, чтобы открыть его.

Выберите *RestartButton* и перейдите к панели Details. Перейдите в раздел *Events* и нажмите на *кнопку* рядом с *OnClicked*.



Это создаст нод *On Clicked (RestartButton)*. Этот нод выполняется, когда игрок нажимает на *RestartButton*.

Воссоздайте следующую схему:

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* возвращает *BP_TunnelSpawner* и вызывает *SpawnInitialTunnels*. Эта функция удаляет все существующие туннели и спаунит новые.

Примечание: Вам может быть интересно, почему я не использовал переменную-ссылку для *BP_TunnelSpawner*. Главная причина заключается в том, что *BP_Tunnel* не связан с *WBP_Restart*. Для нашей простой игры проще реализовать такой способ, чем выяснять, где хранить переменную-ссылку.

Нажмите на *Compile* и закройте Blueprint editor. Нажмите на *Play*, чтобы проверить работу кнопки перезапуска!