14. Class APlayerController

Всем привет! В данном уроке создадим класс PlayerController. Добавим ему свой собственный input и научимся переключаться между несколькими паунами. Для начала давайте немного доработаем класс SandboxPawn. Добавим ему визуальной части, для того чтобы мы могли отличать паунов на сцене. Добавим два дополнительных компонента. Первый из них нам знаком - это статик меш компонент. Второй компонент - новый, называется он UCameraComponent, отвечает он за настройки камеры viewport. Нам необходимо подключить соответствующий заголовочный файл для данного компонента. Давайте подключим его в C++, а в .h файле добавим forward declaration данного класса. Те кто не знает, что такое forward declaration рекомендую почитать, потому что данным концептом мы будем пользоваться постоянно. Переходим в .cpp файл и подключаем заголовочные файлы статик меш компонента и камера компонента. Создаем два компонента по аналогии со SceneComponent. Для этого используем функцию CreateDefaultSubobject. Далее нам необходимо добавить данные компоненты в иерархию компонентов нашего пауна. Для этого воспользуемся функцией SceneComponent, которая называется SetupAttachment. Да именно функцией SceneComponent, потому что как статик меш, так и как камера компонент наследуются от SceneComponent. Это базовый компонент, у которого есть трансформация в сцене. В качестве параметра функция также принимает указатель на класс SceneComponent. Мы будем приаттачиваться к нашему корневому компоненту, который является сцен компонентом. Поэтому мы можем в функцию передать явно указатель на наш компонент, либо сделать это немного по-другому: воспользоваться функцией актора, которая называется GetRootComponent. Данная функция возвращает указатель на корневой компонент актора. Мы в предыдущем разделе указали SceneComponent корневым, поэтому оба данных варианта нам подходят. Аналогичным образом делаем аттачмент для компонента камеры. Иерархию сцен компонентов в акторе можно настроить произвольным образом, то есть можно приаттачить камеру к статик мешу, либо наоборот. Все зависит от дизайна вашего актора. По сути attachment влияет на финальную матрицу преобразования вашего компонента, которая будет вычисляться как произведение матрицы преобразования родительского компонента на матрицу локального преобразования. Изменять трансформацию относительно родительского компонента можно с использованием функций, которые начинаются на SetRelative.. Можно изменить все элементы трансформации: scale, location и rotation. Более наглядно давайте посмотрим как это все выглядит в блюпринтах. Поэтому компилируем проект. Открываем директорию с C++ классами. Создаем на основе нашего C++ класса пауна блюпринт класс. Назовем его BP\_SandboxPawn. Нажимаем кнопку create blueprint class. Смотрим слева в окно Components. Здесь представлены все компоненты нашего актора. Сцен компонент является корневым и к нему приаттачены статик меш компонент и камера компонент. Давайте установим какой-нибудь меш в статик меш компоненте. На пример куб. Куб - гигантский. Давайте изменим ему скейл. Пусть будет 0.3 Отодвинем камеру немного подальше. Скейл куба и положение камеры в пространстве мы меняем относительно родительской трансформации. Давайте для примера создадим дополнительный компонент в блюпринтах. Выберем куб и приаттачим его к камере. И если мы с вами меняем положение камеры в пространстве, то куб также изменяет свое положение, потому что он является дочерним компонентом компонента камеры. Сохраняем блюпринт. В WorldSettings видим, что мы не можем поменять никакие другие классы, кроме GameMode. Это происходит, потому что наш GameMode имеет только С++ версию. Давайте создадим блюпринт наследника нашего C++ GameMode. Добавляем к имени класса BP\_ Мы можем внутри блюпринта GameMode выбрать интересующий нас DefaultPawnClass. Выбираем наш BP\_SandboxPawn. Сохраняемся. В world settings указываем блюпринтовый гейм мод и теперь мы можем настроить любой интересующий нас класс. Запускаем игру. Наш паун теперь имеет внешний вид. Это геометрия куба. Добавим дополнительный мэппинг. Переходим в project settings, в категорию input. В этот раз нам потребуется action mapping. Мы будем переключаться между различными паунами по нажатию клавиши. Назовем наш action mapping - ChangePawn. И добавим любую клавишу. Пусть это будет пробел. Находим spacebar. Переключение между паунами будет происходить в классе playercontroller. Поэтому давайте создадим данный класс. Находим в списке класс PlayerController. Нажимаем Next. Назовем наш класс SandboxPlayerController. Нажимаем на Create Class. Создался наш собственный класс playercontroller. В отличие от других акторов он абсолютно пустой, поэтому все функции придется добавлять вручную. Начнём мы с переопределения в гейм моде класса playercontroller. Открываем .cpp файл гейм мода. Подключаем заголовочный файл playercontroller. Проперти гейм мода для класса playercontroller называется PlayerControllerClass. Переопределяем его по аналогии, как мы это делали для класса пауна. Возвращаемся в класс контроллера. Нажимаем на playercontroller F12. Начнем с функции, в которой настраивается input. Нажимаем Ctrl+F и в поиске будем SetupInput Копируем данную функцию и переопределим в нашем классе playercontroller. Создаем секцию protected. Создаем тело функции в .cpp файле. Итак, playercontroller может тоже принимать input. Функция SetupInputComponent контроллера немного отличается от соответствующей функции пауна. В функции пауна указатель на input component передается как аргумент функции. У playercontroller данный инпут компонент является членом класса. Подключим в контроллере также заголовочный файл инпут компонента. Член класса называется InputComponent. Мы можем напрямую к нему обратиться и забиндить необходимые нам мэппинги. В этот раз нам потребуется другая функция input компонента, которая называется BindAction. Она отвечает за биндинг дискретных мэппингов и немного отличается от функции BindAxis. В первом аргументе нам необходимо указать название нашего мэппинга ChangePawn. Второй параметр имеет тип EInputEvent. Это enum. Он отвечает за тип события, которое происходит с управляющим элементом. Мы с вами передадим туда значение IE\_Pressed. Если нажать F12, то можно посмотреть какие значения имеются у данного enum. Основные типы - это, когда клавиша зажата, либо, когда клавиша отпущена. Мы указали IE\_Pressed и наша функция callback будет вызываться, когда клавиша нажата. И последние два классических параметра для делегатов: это указатель на объект и ссылка на вызываемую функцию. Функцию мы сейчас создадим. Назовем ее ChangePawn. И в .h файле создаем секцию private. Декларируем функцию ChangePawn шпала для функций callback экшн мэппинга. Никакие параметры в сигнатуре не нужны. Создаем тело функции в .cpp файле. Создадим для контроллера собственную категорию логирования. Копируем известный нам макрос. И сразу залогируем, что мы меняем паун. Итак, в нашем случае функция ChangePawn вызывается каждый раз, когда мы нажимаем на клавишу пробел. Мы чуть попозже на сцену добавим несколько паунов и в функцию ChangePawn добавим логику, которая позволит изменять пауна, которого мы сейчас контролируем, на одного из тех, которые находятся на сцене. Для начала нам нужно каким-то образом получить всех паунов, которые находятся на сцене. Сделать это удобнее всего в функции BeginPlay, которая вызывается один раз при старте игры. Функция BeginPlay имеется у каждого актора. Player controller также является актором, поэтому мы можем переопределить его функцию BeginPlay. Создаем тело функции BeginPlay в .cpp файле. В unreal engine существует много различных вспомогательных классов, которые сильно ускоряют процесс разработки. Один из данных классов называется UGameplayStatics Подключим его заголовочный файл Kismet/GameplayStatics Данный класс состоит из различных статических функций. Все функции данного класса также доступны в блюпринтах, поэтому если раньше вы с ними работали, то многие из функций данного класса вам будут знакомы. Функция GameplayStatics, которая позволяет получить всех акторов определенного типа со сцены называется GetAllActorsOfClass Данная функция принимает три аргумента. Первый из них это указатель на WorldContextObject - по сути на любой объект игры. Мы можем передать как указатель на глобальный объект мира, вызвав функцию GetWorld, так и указатель на текущий объект player controller, передав туда ключевое слово this Следующий параметр это указатель на класс, объекты которого мы хотим получить. Мы хотим получить всех SandboxPawn, поэтому передаем указатель на данный класс. И так же необходимо добавить заголовочный файл SandboxPawn. Последний параметр - это массив указателей на акторы. Данный параметр передается по ссылке, т.е. передав в функцию массив мы получим его заполненным указателями на акторов, интересующего нас типа, которые находятся на сцене. Создадим для этой цели локальный массив указателей на акторы. Назовем его Pawns. Данная функция возвращает массив указателей на базовый тип актора. Нам придется делать Cast - то есть преобразовывать указатель от типа AActor к типу APawn. Мы переключение паунов будем делать в функции ChangePawn, поэтому нам необходимо данный массив сделать членом нашего класса. Переходим в заголовочный файл и добавляем данный массив в секцию private. Также пометим данный член класса макросом UPROPERTY, без каких-либо параметров. Зачем это нужно, мы рассмотрим в следующей лекции. Итак, переходим в функцию ChangePawn и в самом начале проверим, что количество элементов массива у нас больше единицы, т.е. проверяем, что у нас есть хотя бы еще один паун, на которого мы сможем переключиться. Функция TArray, которая возвращает количество элемента массива называется Num. Поэтому, если это значение меньше либо равно единице, то мы сразу выходим из функции Далее получим указатель на pawn из массива. Для этого создаем локальную переменную типа указателя на ASandboxPawn, хотя в данном конкретном случае нам достаточно и просто указателя на базовый класс APawn. Называем переменную CurrentPawn и присваиваем данной переменной каст элемента из массива. Каждый раз при нажатии на пробел мы с вами будем получать нового пауна, изменяя индекс массива. У нас отсутствует переменная индекса элемента массива. Назовем её CurrentPawnIndex. Добавим ее в заголовочный файл SandboxPawn. Она будет иметь целочисленный тип int32 и по умолчанию присвоим ей 0. После этого нам необходимо увеличить CurrentPawnIndex на единицу, чтобы при очередном нажатии на пробел нам выдался следующий элемент массива - следующий паун. Возьмем все выражение по модулю размера массива, чтобы CurrentPawnIndex никогда не вышел за его границы. Под лекцией будет ссылка на то, как работает оператор деления по модулю (если кто не в курсе) Далее проверяем на nullptr указатель на паун, над которым мы хотим получить контроль. Если указатель нулевой, то сразу выходим из функции. Опционально так же можно проверить, что паун, который мы хотим установить не совпадает с текущим пауном. Функция плеер контроллера, которая меняет контроль над пауном, называется Possess. Она принимает в качестве аргумента указатель на паун, над которым мы хотим получить контроль. Давайте лог выводить перед вызовом данной функции. И дополнительно добавим две функции в класс SandboxPawn, которые вызываются непосредственно когда контроль над пауном был получен и когда он был утерян. Функции виртуальные, находятся они в классе базового пауна. И мы их можем переопределить в собственном классе. Первая называется PossessedBy У нее имеется один параметр - указатель на контроллер, который получил контроль над данным пауном. А вторая функция называется UnPossessed Создаем ее в .cpp файле. Мы добавили данные функции чисто в учебных целях, чтобы знать об их существовании и посмотреть в каком порядке они вызываются. Поэтому логики в них не будет никакой, просто выведем в консоль сообщения. В функции Possess выведем имена пауна и контроллера, который захватил нашего пауна. Функция UnPossess идет без параметров, поэтому просто напечатаем имя пауна. Посмотрим что у нас получилось. Запускаем editor. В WorldSettings видим, что плеер контроллер класс переопределился. Мы указали SandboxPlayerController. Добавим на сцену несколько паунов. Разместим их в разных частях сцены. Запускаем игру. В начале мы управляем пауном, который создался по умолчанию и так же на сцене имеются 3 дополнительных пауна. Нажимаем пробел и мы получили контроль над другим пауном. Еще раз пробел - над следующим, и так далее. Давайте очистим лог. Так же удалим со сцены GeometryHubActor, чтобы у нас не было в лог спама. Запускаем игру. В предпоследней строчке видим, что SandboxPawn\_C0 был захвачен контроллером - это наш паун по умолчанию. А перед этим видим другие 3 строчки, в которых сообщается, что другими паунами завладели AIController Также, если мы взглянем в WorldOutliner, то увидим, что на сцене существуют данные 3 AIController Unreal сделал это автоматически. Мы можем изменить настройки, поскольку нам AIController сейчас не нужны. Открываем класс пауна и в настройках находим проперти, которое называется Auto Possess AI По умолчанию данное проперти стоит в Place World Мы с вами можем выключить спаун AIController, выбрав опцию Disabled Компилируем блюпринт. Открываем лог, очищаем и запускаем игру еще раз. Все теперь никаких AIController не заспаунилось. Мы это можем видеть, как в WorldOutliner, так и в нашем логе. Нажимаем пробел, видим всю информацию в логах. Сменили пауна. Вызвалась функция UnPossess у текущего пауна и вызвалась функция Possess у пауна, над которым мы получили контроль. У нас имеется баг: если мы зажмем любую клавишу движения и в этот момент поменяем пауна то предыдущий паун будет продолжать двигаться. Давайте этот баг подфиксим. Итак, мы перемещаемся по сцене: то есть одна из клавиш, либо обе у нас зажаты и в данный момент мы нажимаем на пробел, чтобы изменить пауна. После чего вызывается функция UnPossess Если зайти в родительскую функцию UnPossess, то можно видеть, что вызывается функция DestroyPlayerInputComponent. В данной функции input компонент удаляется. После этого биндинг всех наших функций автоматически удаляется и, когда пользователь отпустит клавишу функции MoveForward и MoveRight вызваны не будут. Поэтому компоненты VelocityVector никогда не обнулятся. Данный баг можно подфиксить несколькими способами. Первый из них - это добавить дополнительное условие, когда мы меняем положение актора. В классе пауна существует вспомогательные функции, которые проверяют, кто владеет данным пауном. Функции начинаются на IsControlled.. Просто IsControlled, IsPlayerControlled, IsLocallyControlled и так далее. Давайте зайдем в функцию IsControlled. У пауна имеется член-указатель на текущий контроллер и по сути здесь проверяется является ли он playercontroller или нет. Ниже видим, что имеется функция IsPawnControlled которая просто проверяет существует ли какой-либо контроллер. Это может быть как AIController, так и PlayerController. Но существует более простой способ пофиксить данный баг: после того как мы устанавливаем пауна на новую позицию - просто обнулять VelocityVector. Присвоим ему значение ZeroVector Давайте также добавим проверки на нулевой указатель в функции PossessdBy. Мы с вами используем указатель на контроллера, но не проверяем его на nullptr. Также проверим PlayerInputComponent И в SandboxPlayerController мы так же использовали InputComponent. Добавляем его под if На этом мы данный урок заканчиваем.