**13. Класс APawn. Ввод с клавиатуры.**

1. На что обратим внимание в World Settings, хотя мы можем управлять полетом камеры, имеем какой-то мир и поверхности в нем?

2. Как убрать фокус из игры в UE (hotkey)?

3. Что происходит на самом деле при запуске нашего левела, в каком случае?

4. Какой GameMode был сгенерирован автоматически? Что происходит с остальными классами при его выборе?

5. Как выглядит GameModeSandbox (и от кого наследуется)

6. Создаем нового павна SandboxPawn, что сгенерировал UE, какая функция добавилась по сравнению с актором, для чего она

7. Что нужно создать у нашего game mode перед тем как указывать павна по умолчанию?

8. С помощью каких проперти и функции устанавливаем павна по умолчанию

9. Проваливаемся с помощью F12 и видим, как в конструкторе GameModeBase определены дефолтные классы

10. Можем ли в эдиторе сейчас перемещаться, как-то вращать камеру, почему? Где исправляем это поведение в меню UE? Что такое биндинг? Какие два вида биндингов есть? Чем отличаются? Какой вид мы определили? Как выбрать клавишу и какой параметр есть дополнительный? Как создать противоположные по логике мэпинги в одном?

11. Какой проперти создали для корневого элемента? Что он из себя представляет? Где это все дело вообще определяем?

12. Еще одно проперти для скорости движения пауна

13. Еще одно поле у класса для вектора скорости пауна, значение по умолчанию специальное (нулевой вектор)

14. Создаем две функции (в заголовочном), которые будут вызываться при вызове экшенов, какая сигнатура должна быть для axis мэппинга

15. Переходим в cpp и там в конструкторе создаем SceneComponent и делаем его корневым элементом

16. Создаем свою категорию логирования для SandBox

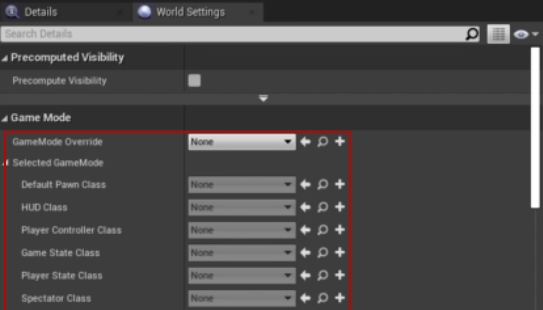
17. Добавляем логи в наши функции движения

18. Делаем так, чтобы наши функции вызывались при срабатывании соответствующих мэппингов, в какой функции это делаем, какой подключаем ЗФ, какие две функции существуют для биндинга функций на мэппинги, какую мы берем, что принимает, на что похожа внутри из того, что мы уже делали?

19. Как программируем движение пауна? Какие возможные значения она будет принимать? Куда мы заносим это значение и как потом это используется?

20. В какой функции меняем значения нашего павна и по какой формуле? Метод проверки вектора на нулевой вектор, синтаксис изменения положения павна

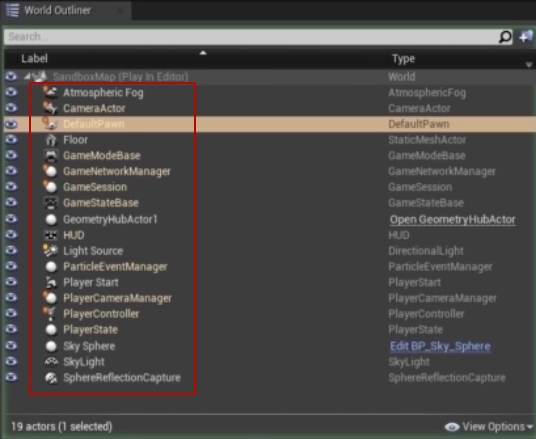
1. Обратим внимание на world settings, где мы будем настраивать все классы текущего уровня:



По умолчанию и все это время у нас game mode и все базовые классы стояли в None. Но тем не менее, когда мы запускали нашу группу нам было доступно управление, то есть мы могли передвигаться стрелочками и управлять камерой мышкой. Что же происходит на самом деле?

2. Давайте нажмем Shift+F1 Это позволяет убрать фокус из игры, но игра не останавливается, то есть все тики продолжаются. Если мы кликнем опять, то фокус вернется.

3. Нажимаем Shift+F1 и смотрим в world outliner. На самом деле все классы заспаунились unreal. Классы эти дефолтные: default pawn, player controller, player state, game mode base, game state base. **То есть если ничего не указано по умолчанию то у левела сетапятся все классы дефолтные**.



И у дефолтного пауна есть минимальный input, непосредственно которым мы и пользуемся.

4. Сейчас мы создадим собственного пауна и посмотрим как все это настраивается вручную. Нажимаем стоп. Устанавливаем Game Mode Override в GeometrySandboxGameModeBase.

Данный game mode был создан автоматически unreal, когда мы создавали проект.

Обратим внимание, что все классы поменялись на дефолтные. Непосредственно их мы и наблюдаем, когда нажимаем play.

Изображение выглядит как текст

Автоматически созданное описание

5. Если мы перейдем в C++ и откроем наш геймод, то увидим, что он абсолютно пустой. Он наследуется от GameModeBase. И поэтому у нас ничего не меняется. Все настройки, которые сейчас в GameModeBase передаются нашему GameModeSandbox.

6. Cоздадим класс пауна: Create new C++ class.

Выбираем в списке пауна.

Назовем нашего пауна SandboxPawn.

Нажимаем кнопку Create Class.

Наш класс пауна создался. Функции, которые сгенерировались по умолчанию, такие же как у актора. Единственное отличие – дополнительно добавилась функция SetupPlayerInputComponent.

В данной функции настраивается пользовательский ввод. С ним мы поработаем немного позднее.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

7. Для начала давайте укажем нашему game mode, что данный класс пауна будет классом пауна по умолчанию. Переходим в заголовочный файл game mode. Добавим конструктор для класса game mode. Создаем секцию public. Декларируем конструктор.

Изображение выглядит как текст

Автоматически созданное описание

В .cpp файле создаем тело функции конструктора.

8. В теле конструктора запишем DefaultPawnClass равняется. Нажимаем F12 на DefaultPawnClass. Открылся заголовочный файл GameModeBase. DefaultPawnClass является проперти данного класса. В нем устанавливается класс, на основе которого game mode создаст объект пауна.

Изображение выглядит как текст

Автоматически созданное описание

Мы с вами уже использовали шаблон TSubclassOf. Данное проперти можно установить в blueprint версии game mode, но также его можно изменить и в С++.

Возвращаемся в .cpp файл нашего гейм мода. Подключим заголовочный файл sandbox пауна. Для того чтобы получить указатель на UClass, существует специальная функция, которая называется StaticClass. Функция является статической.

Изображение выглядит как текст

Автоматически созданное описание

Таким образом в нашем гейм моде мы переопределяем дефолтный класс пауна.

9. Нажимаем на базовом классе GameModeBase F12. Провалимся в заголовочный файл. Если еще раз нажать F12 на любой функции, то мы провалимся в .cpp файл. В самом верху файла видим тело конструктора базового класса. В конструкторе мы можем видеть как засетаплены все основные классы фреймворка. DefaultPawnClass, PlayerControllerClass и так далее.

Изображение выглядит как текст

Автоматически созданное описание

10. В нашем же гейм моде мы с вами пока что переопределили DefaultPawnClass. Возвращаемся в едитор, запускаем игру. И теперь мы не можем перемещаться по сцене. Вращение камеры не работает. Это происходит, потому что в нашем классе пауна пользовательский input никак не обрабатывается.

Давайте это исправим, идем в меню edit, project settings. Открывается окно настроек проекта. Нам нужна подкатегория input. Давайте её выберем в списке, нажимаем, попадаем в секцию биндинга.

Биндинг – это привязка пользовательского имени к какому-то действию: будь то нажатие клавиши на клавиатуре, нажатие кнопки мыши, движения мыши или нажатие кнопки на контроллере.

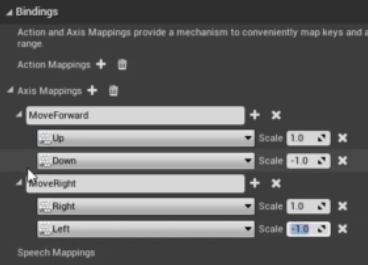
Биндинги делятся на 2 типа. Первый из них – это action mappings. И второй – axis mappings. Action mapping – это дискретное действие. Мы получаем callback event всего лишь один раз, например, когда пользователь нажал на какую-нибудь клавишу.

А axis мэппинг это непрерывное получение состояния элемента управления, то есть если мы зажали кнопку, то нам каждый тик будет приходить значение, что кнопка зажата или, например, мы можем следить за положением мыши.

В интерфейсе создался input, в который необходимо ввести название нашего действия. Давайте назовем его MoveForward. Данный биндинг будет отвечать за движение нашего пауна вперед. Ищем клавишу up. Выбираем ее. Также мы видим что имеется параметр scale. Данная величина домножается на текущее значение axis мэппинга. Как это работает мы увидим совсем скоро.

Мы с вами можем в тот же мэппинг добавить дополнительную кнопку. Кнопку down. И не создавать новый мэппинг для движения назад (move backward), а просто установить скейл у down в -1.

Создаем еще один мэппинг. Называем его move right. В списке находим клавишу right. И по аналогии с move forward добавляем клавишу left. И устанавливаем скейл в -1 Мы создали два мэппинга для движения нашего пауна. Присвоили им клавиши клавиатуры, которые будут вызывать данные экшены.



11. Переходим в код. Теперь мы готовы запрограммировать движение пауна. Давайте для начала немного поработаем в заголовочном файле и создадим все необходимые переменные и функции, а потом запрограммируем все в .cpp файле.

Создадим property для корневого компонента. Корневым в нашем пауне сделаем специальный компонент, который называется SceneComponent. Данный компонент не имеет никакого визуального представления и содержит в себе только трансформацию в мире.



12. Далее нам потребуется property, в котором будет храниться скорость движения пауна. Тип переменной будет float. Назовем ее Velocity. И по умолчанию установим какое-нибудь значение. Пусть будет 300.

Изображение выглядит как текст

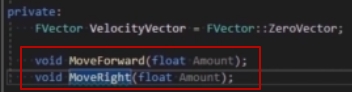
Автоматически созданное описание

13. Добавляем внизу заголовочного файла секцию private. Нам потребуется еще один член класса типа FVector, в котором будет храниться вектор скорости нашего пауна. Называем перемену VelocityVector. По умолчанию пусть данный вектор будет нулевым. Можно воспользоваться специальной константой типа FVector, которая называется ZeroVector.



14. Создадим две функции, которые будут вызываться при вызове экшенов. Первая функция будет отвечать за движение вперед/назад. Назовем ее MoveForward. Функция имеет один параметр float. Все функции, которые биндятся на axis мэппинг, должны иметь один параметр. Что приходит в данном параметре мы совсем скоро увидим.

Вторая функция будет иметь такую же сигнатуру. Назовем ее MoveRight. Создадим тело для каждой функции в .cpp файле. Закрываем заголовочный файл, он нам больше не понадобятся.



15. И переходим в .cpp файл. Переходим в конструктор пауна и создаем компонент SceneCompnent.

Данный компонент будет корневым, поэтому вызываем функцию SetRootComponent. И передаем в нее указатель на SceneComponent.

Изображение выглядит как текст

Автоматически созданное описание

16. Также создаем пауну свою собственную категорию логирования. Копируем макрос. Вставляем и называем категорию LogSandboxPawn.

Изображение выглядит как текст

Автоматически созданное описание

17. Переходим в функцию MoveForward. Давайте залогируем параметр amount. То же самое делаем в функции MoveRight.

Изображение выглядит как текст

Автоматически созданное описание

18. Давайте сделаем так, чтобы наши функции вызывались при срабатывании соответствующих мэппингов. Для этого переходим в SetupPlayerInputComponent.

Нам необходимо добавить заголовочный файл, где объявлен InputComponent. Давайте это сделаем. Находится он по адресу "Components/InputComponent.h"

Для того чтобы забиндить функции на соответствующие мэппинги, в InputComponent существуют две функции. Одна из них называется BindAction, а вторая BindAxis.

Сейчас нам нужна функция BindAxis.

В первом аргументе передаем имя нашего мэппинга MoveForward. Второй аргумент – это указатель на объект. А третий аргумент, это ссылка на функцию, которую мы хотим вызывать.

Если нажать F12, заглянуть в код данной функции, то можно увидеть, что внутри происходит знакомый нам уже бинд делегата, но немного в другой форме.

Дублируем строчку кода, заменяем MoveForward на MoveRight и начинаем программировать движение пауна.

19. Будем изменять значение вектора скорости в зависимости от переменной Amount. Сейчас мы не будем с вами заниматься векторном математикой, предположим, что актор двигается либо вдоль оси X, либо вдоль оси Y.

Поэтому в функции MoveForward у VeolocityVector будем изменять компоненту X, а в функции MoveRight будем изменять компоненту Y. Присваиваем компоненте X значение переменной Amount.

Забегая вперед, сразу хочу сказать, что значение Amount будет либо 1, либо -1, либо 0, то есть у нас когда клавиша не нажата будет приходить значение 0, когда мы нажали на up то Amount будет 1, а когда нажали на Down то Amount будет -1, потому что мы с вами указали параметр scale.

В функции MoveRight изменяем компоненту Y у VelocityVector.

Изображение выглядит как текст

Автоматически созданное описание

20. VelocityVector изменяет значения в зависимости от инпута, и мы с вами можем каждый фрейм, то есть в функции Tick изменять положение нашего пауна в зависимости от данного вектора.

Вначале давайте проверим, что данный вектор не нулевой, если он нулевой – то можно лишний раз вычисления не делать.

В классе FVector имеется специальная функция, которая называется IsZero. Данная функция возвращает true, когда все компоненты текущего вектор равны нулю.

Создаём локальную константу, типа FVector. В ней будет храниться новое положение пауна, которое нам необходимо установить. Называем переменную NewLocation и вычисляться она будет по формуле равномерного прямолинейного движения.

Мы берем текущее положение актора в пространстве, то есть значение, которое возвращает функция GetActorLocation. И присваиваем ей значение скорости помноженное на DeltaTime и помноженное на VelocityVector. После этого нам необходимо установить новое положение нашему пауну. Для этого вызываем функцию SetActorLocation и передаем в неё константу NewLocation.

