**6. Character’s camera control**

1. Создаем новые биндинги от инпута мышки, какой за что отвечает

2. Объявляем две функции, биндим их в коде, сами функции реализуем через специальные встроенные, какие, что принимают, почему идет разделение и привязка именно к контроллеру?

3. Как внутри устроена функция для поворота (которая по вертикали)

4. По вертикали почему-то не крутится, как поправили?

5. Теперь все работает, однако с инверсией, как поправили?

6. Из-за чего наблюдалось такое поведение?

7. Вращение происходит вокруг камеры, а не по орбите вокруг персонажа, как это исправили? Что добавили, какой ЗФ подключили, как устроили привязку друг другу компонент, а также установили флаг из кода (который для вращения по вертикали устанавливали)

8. Что из себя представляет новый компонент?

9. Где добавить персонажу анимацию?

10. Какое сделали наблюдение и как исправили код (про биндинги и функции, которые мы ввели дополнительно)

1. Мы хотим научить камеру вращаться вокруг персонажа в зависимости от инпута мышки. Для начала надо создать два новых Axis Mapping’a:



Первый mapping – LookUp – будет отвечать за наклон камеры вверх и вниз. Данный угол будет меняться в зависимости от положения мыши по вертикали, поэтому был выбран MouseY.

Второй mapping – TurnAround – будет отвечать за повороты камеры вокруг персонажа. Угол данного поворота будет меняться в зависимости от положения мыши по горизонтали (MouseX).

2. У **персонажа** объявляем две функции, которые понадобятся нам для поворота камеры:

Изображение выглядит как текст

Автоматически созданное описание

Связываем данные функции с соответствующими мэппингами:

Изображение выглядит как текст

Автоматически созданное описание

Определяя поведение в функции LookUp, для поворота камеры вверх-вниз используем функцию Pawn’a, которая называется **AddControllerPitchInput**() (соответственно, есть функции для Roll и Yaw).

Функция принимает один аргумент – значение угла Pitch, который будет добавлен к вращению.

Аналогичным образом добавляем функцию **AddControllerYawInput**() для функции TurnAround, отвечающей за вращение камеры вокруг персонажа. Таким образом мы изменяем вращение нашего пауна (персонажа) через вращение контроллера. То есть паун не владеет вращением управления. Данное разделение не случайно и может быть использовано во многих ситуациях, например, если нам мышкой нужно целиться в одну точку, а визуально тело повернуто в другую.

Изображение выглядит как текст

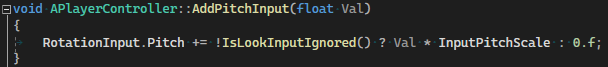
Автоматически созданное описание

3. Функция AddControllerPitchInput() на самом деле является оберткой над функцией AddPitchInput() PlayerController’a:

Изображение выглядит как текст

Автоматически созданное описание

Внутри AddPitchInput() же изменяется угол Pitch переменной RotationInput на значение переменной, которую мы передаем, умноженной на InputPitchScale:



RotationInput – это входное значение по оси, которое меняется каждый тик:



4. Перейдя в Editor заметим, что у нас вращение по горизонтали осуществляется, а по вертикали нет. Это можно исправить в BP-редакторе (чтобы не перекомпилировать проект). Для этого надо изменить у нашего персонажа **в компоненте камеры** флаг Use Pawn Control Rotation – теперь вращение нашей камеры будет соответствовать вращению контроллера, углы которого мы меняем при перемещении мыши.

5. Теперь у нас все работает, однако камера вращается с инверсией. Чтобы это исправить мы просто в нашем биндинге исправим значение Scale на отрицательное:



6. Если мы откроем наш BP класс для контроллера, то увидим причину изначальной инверсии – это параметр Input Pitch Scale. Данный параметр мы видели в коде функции AddPitchInput(). Изменяя его значение мы можем управлять скоростью поворота нашей камеры.

Изображение выглядит как текст

Автоматически созданное описание

7. Угол Pitch заработал, однако вращение происходит вокруг точки, в которой находится в данный момент камера. Исправим это и сделаем так, чтобы мы могли вращать камеру **по орбите вокруг персонажа**. Сделаем это с помощью дополнительного компонента, который называется SpringArmComponent:

Изображение выглядит как текст

Автоматически созданное описание

В cpp-файле подключаем ЗФ, который называется GameFramework/SpringArmComponent.h и создаем в конструкторе персонажа наш компонент и привязываем его к корневому компоненту.

Так же как до этого с камерой в BP, установим в коде нашему SpringArm-компоненту переменную UsePawnControlRotation в true, а также изменим привязку камеры с нашей компоненты капсулы на этот SpringArm-компонент:

Изображение выглядит как текст

Автоматически созданное описание

8. Теперь в BP нашего персонажа видим наш SpringArm-компонент. Сбрасываем относительную трансформацию у камеры (которую настроили ранее). Данный компонент (SpringArm) представляет собой некий «штатив», на котором находится камера. Мы можем менять у него различные параметры (длину плеча и другие). Так же данный компонент очень помогает при обработке коллизий, чтобы наша камера не уткнулась в стену.

Не забудем отключить флаг UsePawnControlRotation у камеры и настроим трансформацию у SpringArm-компонента.

9. Добавим анимацию нашему персонажу (захардкодим ее так, что он постоянно будет бежать) в меше в категории Animation.

10. Последнее: обратим внимание, что функции AddControllerPitch/YawInput() имеют такую же сигнатуру, что и наши LookUp и TurnAround, поэтому мы можем не создавать лишние функции, а забиндить их напрямую:

Изображение выглядит как текст

Автоматически созданное описание