**10. Back, right and left movement animation**

1. Для начала создали новый BS – чем он отличается? Как определили оси? Как расположили анимации?

2. Как изменили анимацию в нашем состоянии Walk? Какая доп. переменная появилась?

3. Для начала сделаем все через блюпринты – как нарисовали стрелочку направления нашего персонажа?

4. Как нарисовали стрелочку для скорости?

5. Как вычислить угол между нашими двумя векторами?

6. Если мы сейчас выведем значение угла – какая проблема возникнет? Как это исправляем (здесь именно получение координаты Z другого вектора)?

7. Имеем все для подсчета нашего угла – как это делаем, подставляем данное значение в переменную направления, ради которой все пляски

8. Переписываем получение направления в C++ - как?

9. Как выглядит наш ортогональный вектор?

10. В логах были Warning’и – почему и как это пофиксили?

11. Вызываем нашу переменную из C++ и устанавливаем в блюпринтах

12. Последнее – оптимизация кода нашей функции, в каком случае нет смысла ее выполнять и в каком случае будет наблюдаться баг и как это исправить?

1. Для начала создадим новый BlendSpace (уже не 1D) для обновления нашей машины состояний. Назовем BP\_LocomotionWalk2.

Единственное отличие данного ассета от 1D-ассета в том, что он двухмерный – две оси.

Горизонтальная ось у нас также будет отвечать за скорость (Velocity от 0 до 600), а вертикальная – Direction (от -180 до 180, то есть вектор направления в правом полукруге от 0 до 180, а в левом – от -180 до 0), она будет отвечать за направление, в котором бежит наш персонаж.

Выкидываем посередине в левую часть нашу анимацию Idle (в ноль по скорости и в ноль по направлению), RunFwd – посередине в правую часть, RunBwd – в правой части, но сверху и снизу (то есть направление 180 и -180 градусов). Движение влево и вправо – RunLt и RunRt - -90 и 90 соответственно:



2. Сохраняем BS и отправляемся в анимационный блюпринт, где в нашей машине состояний в состоянии Walk заменяем предыдущий BS новым, однако теперь нам нужно два параметра. Поэтому создаем переменную Direction для направления. Нам надо будет ее вычислить:

Изображение выглядит как текст, внутренний, электроника

Автоматически созданное описание

3. Для начала вычислим данную переменную в BP в ивент графе. Начнем с визуализации векторов, между которыми нам надо будет вычислять углы.

Для начала копируем ноду TryGetPawnOwner.

Далее используем ноду DrawDebugArrow – она рисует вектор в пространстве.

Начнем с вывода вектора направления нашего персонажа – forward-вектора. В LineStart идет позиция персонажа (получаем через GetActorLocation).

Далее получаем сам вектор – GetActorForwardVector, данный вектор единичный (нормализованный), поэтому домножим его на какой-нибудь множитель.

И для того, чтобы получить LineEnd, нам надо сложить GetActorLocation с GetActorForwardVector, умноженным на множитель.

Изображение выглядит как текст, внутренний

Автоматически созданное описание

Изображение выглядит как с плиткой, плитка, автомат

Автоматически созданное описание

4. Копируем нашу логику и теперь выведем вектор скорости. Для этого просто заменяем GetActorForwardVector на GetVelocity. Однако вектор скорости не единичный, поэтому нормализуем его через ноду Normalize:

Изображение выглядит как текст, внутренний

Автоматически созданное описание

Изображение выглядит как внешний

Автоматически созданное описание

5. Осталось вычислить угол между forward-вектором и вектором скорости и подставить данное значение угла в переменную Direction.

Копируем наши forward-вектор и вектор скорости. Первое, что надо посчитать, это угол между данными векторами.

В векторной математике угол между векторами высчитывается с помощью скалярного произведения.

Функция в классе Vector называется Dot Product. Данная функция возвращает косинус угла между векторами (если они единичные), поэтому нам надо применить функцию арккосинуса Acos, чтобы получить значение угла. Находится она в классе FMath.

Изображение выглядит как текст, внутренний, черный, зеленый

Автоматически созданное описание

Тут не в то место присоединен вывод ACOSd – надо в In String. А еще нет соединения через белую связь..

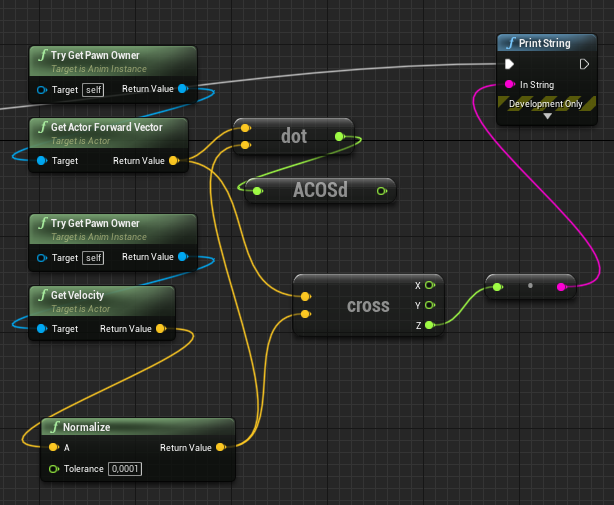
6. Если мы сейчас будем выводить значение полученного угла, в обеих частях окружности оно будет положительным.

Так же при нулевой скорости значение равно 90, потому что arccos(0) = 90.

Для определения знака угла нам потребуется функция вычисления векторного произведения из класса Vector с названием Cross Product.

Мы будем пользоваться следующим свойством данной мат. операции: итоговый вектор (являющийся результатом векторного произведения) ортогонален первому и второму вектору, а так же тем, что UE имеет левую систему координат. То есть мы можем взять координату Z данного вектора и посмотреть на ее знак.

Через правую кнопку мыши разбиваем вывод (Split Struct Pin). При выводе данной переменной на экран у нас ее знак будет слева отрицательным, а справа – положительным.



7. Теперь мы имеем все данные для подсчета нашего угла уже со знаком. Для этого нам нужно от координаты Z взять функцию Sign, определяющую знак координаты, и перемножить со значением градусов.

Так же теперь когда вектор скорости равен 0, то и Cross Product выдает 0, поэтому данный случай мы тоже корректно обрабатываем.

Подставляем данное значение в Direction:

Изображение выглядит как текст, внутренний

Автоматически созданное описаниеПри значениях угла между 0-90 и 90-180 анимации просто интерполируются между двумя состояниями.

8. Перепишем все в C++. Просто будем вызывать функцию нашего персонажа и подставлять значение в Direction.

Создаем функцию, которая будет возвращать значение угла:

Изображение выглядит как текст

Автоматически созданное описание

Здесь мы будем программировать формулу, которую определили в блюпринтах.

Изображение выглядит как текст

Автоматически созданное описание

GetSafeNormal() – функция, которая возвращает нормаль, GetVelocity() – возвращает вектор скорости.

Арккосинус возвращает угол в радианах, поэтому использовали RadiansToDegrees.

9. Перед тем, как заменить все блюпринты на вызов функции из C++, посмотрим, как выглядит наш ортогональный вектор:

Изображение выглядит как текст, внутренний, черный

Автоматически созданное описание

Изображение выглядит как внешний, человек, автомат

Автоматически созданное описание Изображение выглядит как внешний

Автоматически созданное описание

Если вектор скорости справа, то векторное произведение направлено вверх – координата Z положительна, если вектор скорости слева – то вниз и Z координата отрицательна.

10. У нас в логах были Warning’и. Мы не учли один момент. Если мы посмотрим на функцию Update нашего анимационного блюпринта, то мы увидим, что она активна (красная полоска с точками). Она вызывается не только в игре, но и в едиторе.

Изображение выглядит как текст, внутренний, зеленый, черный

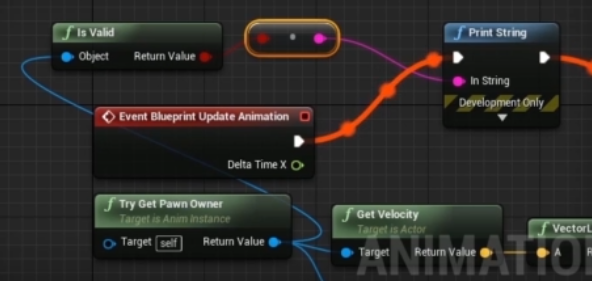
Автоматически созданное описание

Если мы выведем сейчас имя нашего персонажа (сверху показано как), то увидим, что в лог печатается пустая строчка – наш персонаж – Null, потому что анимационный блюпринт работает как standalone и, грубо говоря, если бы мы так обратились к указателю в C++, то получили бы неопределенное поведение.

Изображение выглядит как текст

Автоматически созданное описание

Если проверим его на null:

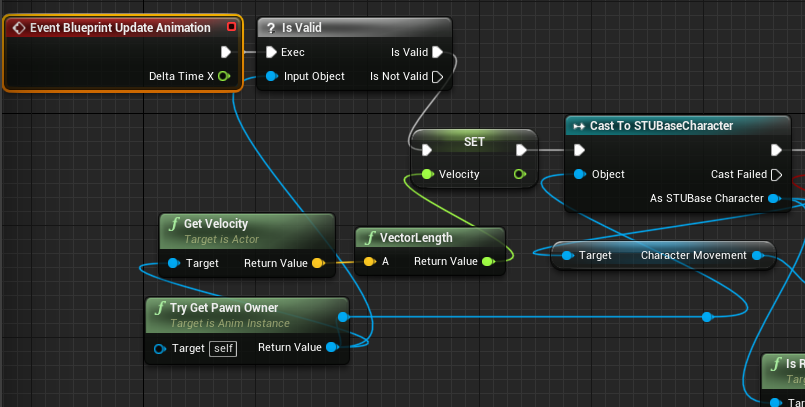


Получим:

Изображение выглядит как текст

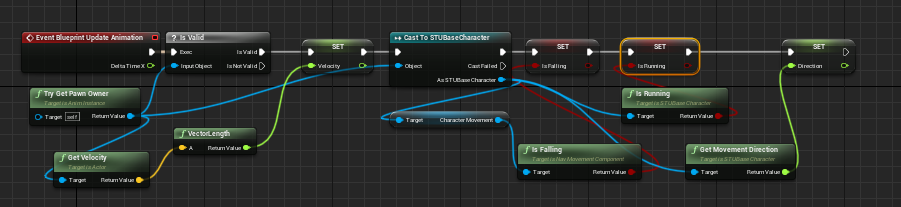
Автоматически созданное описание

Поэтому поставим проверку на валидность:



Теперь нет Warning’ов и дальше нашей функции IsValid поток не идет, так как она не позволяет этому случиться.

11. Теперь выведем нашу переменную Direction из C++, удалив все визуализации:



Изображение выглядит как текст, внутренний

Автоматически созданное описание

И удалим старый BS, так как теперь нам нужен только новый.

12. Последнее – оптимизируем наш код – в функции GetMovementDirection нет смысла выполнять операции, если скорость равна 0.

И еще один момент – векторное произведение возвращает 0, когда два вектора либо одинаково направлены, либо противоположно – поэтому в нашем случае, если вектор скорости будет противоположно направлен FWD-вектору, возвращаемое значение будет равняться 0 и анимация бега назад у нас включаться не будет. Исправим это тернарным оператором проверяя, является ли наше векторное произведение нулевым вектором:

Изображение выглядит как текст

Автоматически созданное описание