**16. EQS CUSTOM TEST**

1. Какая у нас сейчас наблюдается проблема после подбора пикапа? Какие два варианта есть исправления?

2. Какой мы выберем? Как можно сделать тест? Как лучше? От кого наследуемся, как называем в соответствии с шаблоном?

3. Какие две функции определяем в нашем тесте? Когда вызывается вторая? Какую логику мы хотим в ней реализовать? Как реализован конструктор в классе теста Trace? Как мы реализуем аналогичное поведение у нас (какие там 3 переменные определяем)? Берем код из функции RunTest для итерации по акторам, что оставляем из него? Какую логику реализуем уже нашу?

4. Создаем функцию в базовом классе пикапа. Как реализуем? Что для этого приходится изменить в нашем классе?

5. Добавляем наш тест – кому? Какое поведение еще неправильно у нас настроено? Как установить данное значение (код из класса теста Trace)? Как мы переносим его к себе и изменяем логику функции теперь?

1. У нас наблюдается проблема – когда мы подбираем пикап, мы устанавливаем ему Visibility в false для root-компонента и всех дочерних компонент. Однако EQS до сих пор его видит, он невидимый только для нас. Значит, NPC может начать движение к невидимому пикапу. Исправим это. Сделать это можно разными способами – можно создать **кастомный** **генератор** в EQS, который будет выдавать только видимых акторов, или же создать свой **собственный** **тест**, который будет фильтровать таких акторов.

2. Выберем второй вариант. Тест можно сделать по-разному: можно полагаться на Visibility компонентов, но это не очень надежный способ, потому что возможны различные дизайнерские решения, когда Root-компонент невидимый, а пикап взять можно, либо наоборот. Более надежное решение – создать специальную функцию пикапов, которая будет проверять, доступен он для взятия или нет. Наш тест будет сделан конкретно под пикап.

Создадим его, наследуясь от EnvQueryTest. При этом название класса должно быть в соответствии с паттерном и заканчиваться на \_ и название теста – движок по этому паттерну определяет имя теста в BP-редакторе:

Изображение выглядит как текст

Автоматически созданное описание

Называем EnvQueryTest\_PickupCouldBeTaken.

3. В нем мы сразу определили конструктор с FObjectInitializer почему-то и подключили ЗФ для наших пикапов. В базовом классе находим функцию RunTest – она вызывается каждый раз, когда запускается EQS, переопределим ее поведение в нашем:

Изображение выглядит как текст

Автоматически созданное описание

В данной функции нам надо пройти по всем пикап-акторам и отсеять тех, кого взять нельзя. Посмотрим, как это сделано в тестах в библиотеке движка и реализуем аналогично наш тест. Для начала посмотрим на конструктор:

Изображение выглядит как текст

Автоматически созданное описание

В нем инициализируются некоторые переменные: Cost – отвечает за сложность теста, ValidItemType – может быть вектором, актором, зависит от того, с каким типом будет работать наш тест, SetWorkOnFloatValues – в зависимости от параметра устанавливается FilterType, наш тест логический, поэтому переменная должна иметь значение Match, оно установится, если передать в данную функцию false.

Перенесем данные строчки в наш конструктор с некоторыми изменениями:

Изображение выглядит как текст

Автоматически созданное описание

Возвращаемся в файл базового класса и смотрим на функцию RunTest и посмотрим, как происходит итерация по акторам:

Изображение выглядит как текст

Автоматически созданное описание

EQS имеет специальный тип итератора ItemIterator, который проходит по всем сгенерированным сущностям. Копируем данный цикл и вставляем в нашу функцию RunTest. Чистим код, нам не нужен внутренний цикл for, переменная ItemLocation, нужен только код получения актора:

Изображение выглядит как текст

Автоматически созданное описание

Преобразуем данный указатель к классу пикапа и в случае провала переходим к следующему элементу, а в случае успеха создаем локальную переменную, которой присвоим возвращаемое значение будущей функции CouldBeTaken нашего пикапа – она будет возвращать true, если пикап можно подобрать.

Изображение выглядит как текст

Автоматически созданное описание

Далее в зависимости от данного флага нам надо пометить текущий Item – проходит он наш тест или нет – с помощью функции SetScore. Она принимает:

* TestPurpose – переменная класса, просто передадим ее, этот enum также выставляется через Editor, имеет значения Filter, Score, FilterAndScore;
* FilterType – его установили мы ранее, это также переменная класса;
* bScore – передаем true, потому что можем поднять пикап;
* bExpected – ожидаемое значение, передаем true.

Если же условие не выполняется, вызываем функцию ForceItemState, которая принимает статус прохождения теста – Failed или Passed.

4. Осталось создать функцию CouldBeTaken в базовом классе пикапа.

Изображение выглядит как текст, оранжевый

Автоматически созданное описание

В нашем случае проверять, можно ли поднимать пикап, мы можем с помощью проверки того, установлен наш таймер или нет с помощью функции IsTimerActive, принимающая таймер в качестве аргумента.

Изображение выглядит как текст

Автоматически созданное описание

TimerHandle из функции PickupWasTaken придется сделать членом класса.

Было:

Изображение выглядит как текст

Автоматически созданное описание

Стало:



Изображение выглядит как текст

Автоматически созданное описание

5. Остается добавить наш тест не только на EQS\_FindAmmoPickup, но и на EQS\_FindHealthPickup, так как логика подходит для них обоих. При этом мы заметим, что проперти Bool Match никак не влияет при переключении на наш тест, хотя должен инвертировать условие. Исправим это.

Данное булевое значение в начале RunTest есть две строчки получения этого значения:

Изображение выглядит как текст

Автоматически созданное описание

Скопируем их в нашу функцию. Переменная BoolValue имеет специальный тип – это не чистый bool, данная переменная подписывается на изменение проперти, и, чтобы получить ее сырое значение, необходимо вызвать метод GetValue (на картинке выше переменная bWantsHit). Назовем переменную WantsBeTakable, она будет принимать значение true/false в зависимости от того, что мы выставим в эдиторе:

Изображение выглядит как текст

Автоматически созданное описание

Теперь мы можем упростить весь наш код фильтрации использовать только SetScore, изменив последние два параметра:

Изображение выглядит как текст

Автоматически созданное описание

В первый из них передадим результат нашего теста, во второй, который определяет желаемый результат теста, переменную bWantsBeTakable.