Итерирование

Чтобы реализовать вашу реальную логику Механики, вам потребуется обработать все Сущности, удовлетворяющие Фильтру. В целях эффективности и последовательности это делается вручную через Цепи. Итерируясь по цепям, вы итерируетесь по всем Сущностям и Сущностным объектам внутри конкретной цепи.

В итерации по цепям помогают Курсоры. Их семантика напоминает итераторы в стандартных контейнерах.

Работа в C++

Итерирование по цепи сделано через специальный тип объекта *Курсор* (Cursor). Можете использовать столько, сколько захотите, но обычно, достаточно одного:

```
FChain::FCursor Cursor = Chain->Iterate();
```

Если цепь твердотельная, то код будет выглядеть так:

```
FSolidChain::FCursor SolidCursor = SolidChain->Iterate();
```

Когда вы получили желанный курсор, вы можете построить простой while-цикл:

```
while (Cursor.Provide())
{
  auto Trait = Cursor.GetTrait<FMyTrait>();
  ...
}
```

`Provide()` метод подготавливает нужное состояние и возвращает `false`, когда закончились слоты в цепи (`true` иначе).

Имея твердотельный Курсор вы можете получить прямую ссылку (без копирования) на трейт (используя метод `GetTraitRef()`):

```
while (SolidCursor.Provide())
{
  auto& Trait = SolidCursor.GetTraitRef<FMyTrait>();
  ...
```

```
}
```

Пожалуйста, заметьте, что цепи утилизируются автоматически, когда все итерируемые курсоры закончили итерироваться по слотам. Чтобы предотвратить такое поведение особо, можете использовать вызовы Retain()/Release(), чтобы самостоятельно контролировать время жизни объектов:

```
Chain->Retain(); // Забрать цепь.
FChain::FCursor Cursor = Chain->Iterate();
while (Cursor.Provide())
{
  ...
}
// Здесь выполняем операции над цепью.
// Гарантируется, что она не будет удалена.
...
Chain->Release(); // Очищаем данные цепи.
```

Встроенные курсоры

Аппарат предоставляет способ итерироваться по цепям встроенными Курсорами. В основном, эта технология используется внутри плагина для корректной работы Blueprint-ов, и вам её стоит избегать в своём C++ коде.

Код будет довольно прост. Он состоит из while-цикла с одним условием:

```
while (Chain.BeginOrAdvance())
{
  ...
}
```

Внутри этого цикла вы можете реализовать нужную логику, используя Сущности напрямую или служебные методы Цепей:

```
while (Chain.BeginOrAdvance())
{
  FSubjectHandle Subject = Chain.GetSubject();
  UMyDetail* MyPosition = Chain.GetDetail<UMyDetail>();
  FMyTrait   MyVelocity;
  Chain.GetTrait(MyVelocity);
  MyPosition->X += MyVelocity.VelocityX * DeltaTime;
  MyPosition->Y += MyVelocity.VelocityY * DeltaTime;
  ...
  MyVelocity.VelocityX = 0;
  MyVelocity.VelocityY = 0;
```

```
  Subject.SetTrait(MyVelocity);
}
```

Когда указатель-Курсор Цепи пройдёт последнюю доступную Сущность (или Сущностный объект), Цепь будет уничтожена и ранее заблокированные чанки и ремни вновь разблокируются, все ожидаемые структурные изменения будут незамедлительно выполнены (если они вообще были).

Прямое итерирование

Если вам необходимо итерироваться по чанкам напрямик, вы можете инициализировать Чанк-Прокси. Обычно, вам потребуется проитерироваться по всем собранным Прокси через соответсвующий Enchain метод и в каждом чанке итерироваться по его Сущностям-слотам.

Пример будет таким:

```
TArray<TChunkProxy<FSolidSubjectHandle, FJumpingTrait>> ChunkProxies;
Mechanism->Enchain(TFilter<FJumpingTrait>(), ChunkProxies);
for (int32 i = 0; i < ChunkProxies.Num(); ++i) // Итерируемся по всем
подходящим чанкам...
{
    auto& ChunkProxy = ChunkProxies[i];
    for (int32 j = 0; j < ChunkProxy.Num(); ++j) // Итерируемся по слотам...
    {
        // Perform the necessary logic...
        ChunkProxy.TraitRefAt<FJumpingTrait>(j).Position +=
FVector::UpVector * DeltaTime;
    }
}
```

Заметьте, однако, что этот подход абсолютно ручной, и не выполняет какие-либо логические проверки во время итерирования, например, совпадение флагов.

Если вы также выполняете изменение топологии внутри ваших циклов, то ваши сущности могут произвольно изменить свои чанки или слоты, поэтому вам скорее всего потребуется проверять слоты на (не-)свежесть.

Сделать это можно так:

```
for (int32 j = 0; j < ChunkProxy.Num(); ++j) // Итерируемся по всем слотам
чанка...
{
    if (ChunkProxy.IsStaleAt(j)) continue; // Пропустить сущность, если она была
перемещена или удалена...

    ChunkProxy.SubjectAt(j).SetTrait(FSpeedBoostTrait{10.0f});
}
```

Итерация чанков напрямую (через прокси) может дать определенный прирост производительности по сравнению с обычной итерацией и оперированием. В основном это связано с возможностью контролировать каждый аспект итерации вручную и исключать все лишние проверки. Вам просто нужно точно знать, что вы делаете и чего пытаетесь достичь, так как этот способ менее безопасен.