**7. AI SERVICE / AI DECORATOR**

1. Новый концепт, Service – что это и для чего?

2. Перепишем логику нахождения противника через сервис – как (словесное описание того, что надо сделать)? Что надо добавить в Blackboard? Как создать новый сервис в дереве поведения в блюпринтах? Как это сделали в C++, от кого наследовались, как назвали?

3. Добавляем нашему сервису проперти и две функции – какие? Подключаем 4 ЗФ и определяем обе функции.

4. Теперь надо подредактировать контроллер. Добавляем проперти и функцию – какие и для чего (пока только ЗФ)?

5. Подключаем ЗФ в cpp-контроллера и определяем новую функцию, а так же модифицируем после этого функцию Tick – как?

6. Теперь сделаем два варианта поведения – какие? Что нужно для выбора (нода) в дереве поведения? Как работает? Как добавить сервис нодам? Что нужно указать у сервиса? Надо изменить теперь движение к позиции, но не получается – почему и как пофиксили? Какой еще надо было добавить флаг?

7. Однако сейчас персонаж реагирует на увиденного персонажа не сразу – почему? Как это исправили (через добавление специального модификатора)? Как он работает и кому добавили и как определили?

8. Теперь будем бежать не в точку, где находится враг, а так же как и с патрулированием будем находить некоторую точку в радиусе вокруг врага. Как это можно реализовать двумя способами, но как сделали мы (добавили два проперти и изменили функцию, которая выполняется при вызове таска)? Как после этого изменили дерево поведения?

1. Рассмотрим новый концепт, относящийся к BehaviorTree, называемым Service – это специальный класс, который может быть добавлен к узлам дерева поведения, у него имеется своя Tick-функция, в которую мы можем поместить нашу игровую логику и, например, при определенных условиях в игре изменять ключи в нашей доске Blackboard.

2. Перепишем логику нахождения противника через сервис. Когда NPC видит противника, рандомным образом будет вычисляться точка вокруг вражеского персонажа, в которую NPC будет бежать.

Для начала добавим ключ EnemyActor типа Object в наш Blackboard – в данном ключе будет храниться указатель на текущего врага.

Для того чтобы создать новый сервис через блюпринт в дереве поведения есть кнопка New Service. Мы же создадим его через C++, поэтому создадим класс STUFindEnemyService, наследующийся от BTSerice, в папке AI/Services (нужно будет указать в Build.cs).

3. В ЗФ добавляем конструктор (он будет такой же как у Task – будем просто определять имя ноды), проперти, в котором будет храниться ключ для вражеского персонажа.

Изображение выглядит как текст

Автоматически созданное описание

Сервис имеет свою собственную функцию Tick, ее будем рассматривать в блюпринтах, а сейчас в базовом классе найдем функцию TickNode и переопределим ее – логика поиска противника будет находится в данной функции.

Подключаем 4 ЗФ:

Изображение выглядит как текст

Автоматически созданное описание

В функции TickNode можем получить указатель на Blackboard и Controller через Behavior Tree.

Далее получаем PerceptionComponent, который находится в контроллере. Если он нашелся, то можем записать значение найденного ближайшего актора в Blackboard в EnemyActorKey.

По сути мы переписали то, что делали в AIController’e через сервис. После такого преобразования мы будем иметь доступ к нашему контроллеру как в Behavior Tree, так и в контроллере.

4. Теперь немного подредактируем контроллер. Создадим проперти для имени ключа, в котором содержится указатель на вражеского персонажа. Здесь мы не можем указать FBlackboardKeySelector, потому что контроллер ничего не знает о ключах:

Изображение выглядит как текст

Автоматически созданное описание

Данное имя в Blackboard имеет наш ключ, в котором хранится указатель на врага.

И добавим еще функцию, которая будет возвращать указатель на актора, на котором нам надо будет сфокусироваться.

5. В cpp-файле подключим ЗФ для Behavior Tree – через него мы будем получать объект, на котором надо сфокусироваться.

Изображение выглядит как текст

Автоматически созданное описание

Функция возвращает указатель на UObject, поэтому надо кастить к Actor.

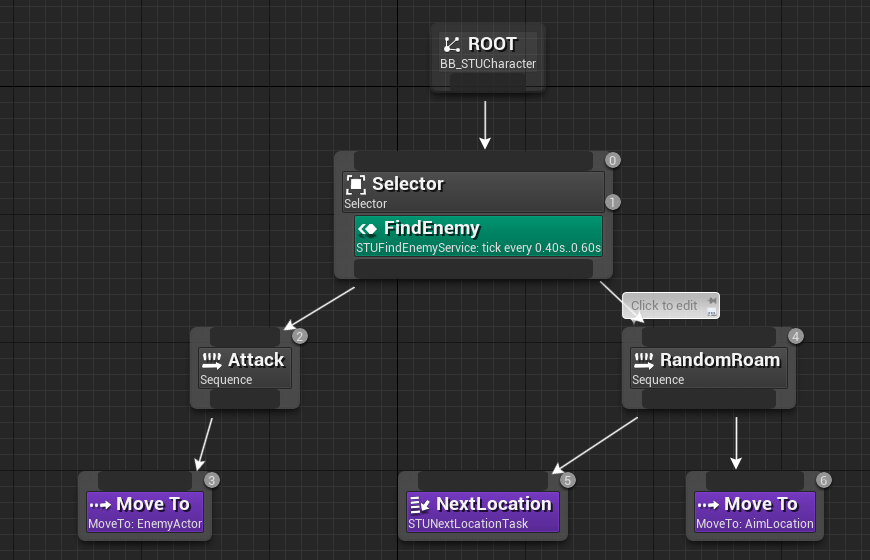
И теперь мы можем в нашей функции Tick присвоить переменной AimActor значение, которое возвращает функция GetFocusOnActor.

6. Теперь сделаем два варианта поведения:

* патрулирование территории (уже реализован);
* атака – добавим таск MoveTo к врагу.

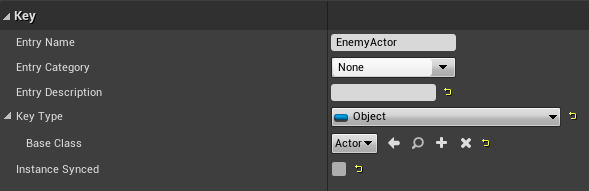
Чтобы сделать выбор между двумя вариантами, которые мы хотим запустить, есть нода Selector. Она запускает дочерние деревья слева направо. Если выполнение поддерева заканчивается Failed, то запускается следующее дерево и т.д.

Добавим созданный нами сервис. Сервис можно добавить к любому узлу дерева поведения через ПКМ->AddService. Сервис поиска врага должен работать всегда, поэтому добавляем его в самую верхнюю ноду.



В проперти EnemyActorKey надо указать ключ EnemyActor. Ниже находятся настройки обновления функции Tick.

Теперь нам надо изменить в таске MoveTo движение к позиции EnemyActor, однако мы не можем выбрать наш EnemyActor. Чтобы это исправить, надо в Blackboard указать в качестве BaseClass нашего EnemyActor из Object в Actor (не работало, потому что у UObject трансформации в мире нет).

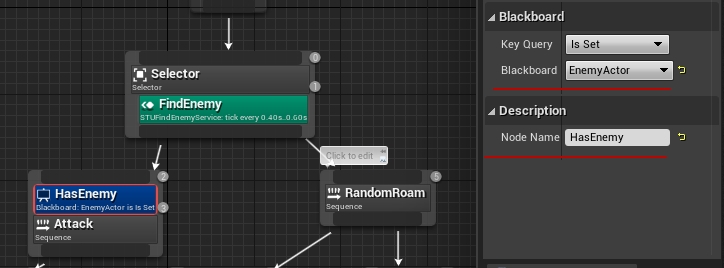


Так же надо поставить Observe Blackboard Value в true. Если ключ изменится, то целевая локация для MoveTo будет пересчитана. И если EnemyActor будет равен nullptr, то нода вернет Failed и начнется RandomRoam.

Изображение выглядит как текст

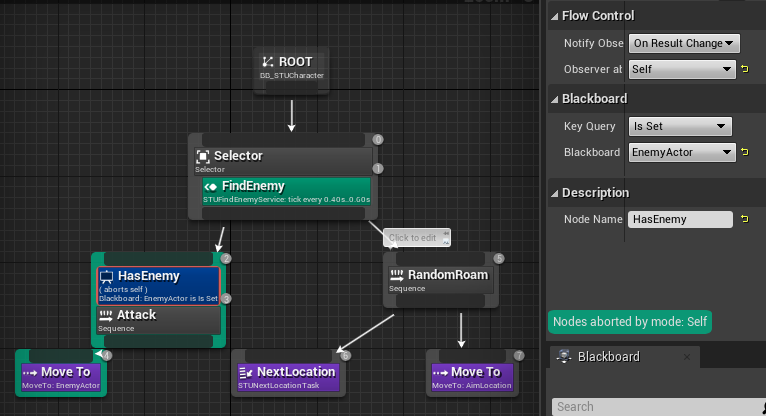
Автоматически созданное описание

7. Однако сейчас после того, как персонаж нас заметит, атака начнется лишь в том случае, когда по завершению последовательности RandomRoam, объект EnemyActor будет не нулевой. Однако хотелось бы, чтобы NPC **сразу** изменял поведение, завидев EnemyActor.

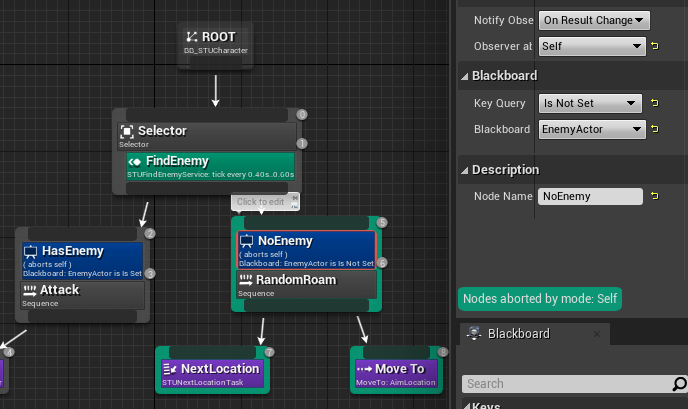


Для подобных целей имеется специальная нода Decorator. Она содержит логическое условие, в случае невыполнения которого можно остановить все поддерево. Через ПКМ->Add Decorator на нужной ноде выбираем Blackboard – данный декоратор проверяет, установлен у нас какой-то ключ в Blackboard или нет. В настройках зададим ему имя HasEnemy и проверим, не является ли наш EnemyActor нулевым.

Если у нас враг не находится, то нам необходимо остановить данную Sequence с помощью проперти Observer Aborts, выбираем Self (зеленым подсветились останавливаемые ноды):



Такой же декоратор, но с противоположной логикой, добавим RandomRoam, чтобы останавливать его в случае, когда мы находим актора.



8. Теперь будем бежать не в точку, где находится враг, а так же как и с патрулированием будем находить некоторую точку в радиусе вокруг врага. Можно добавить отдельный таск, мы же улучшим существующий NextLocation.

Изображение выглядит как текст

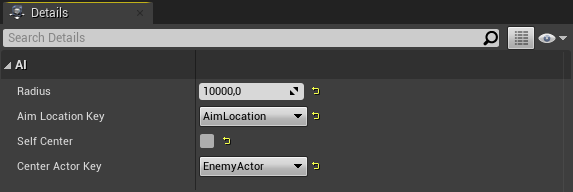
Автоматически созданное описание

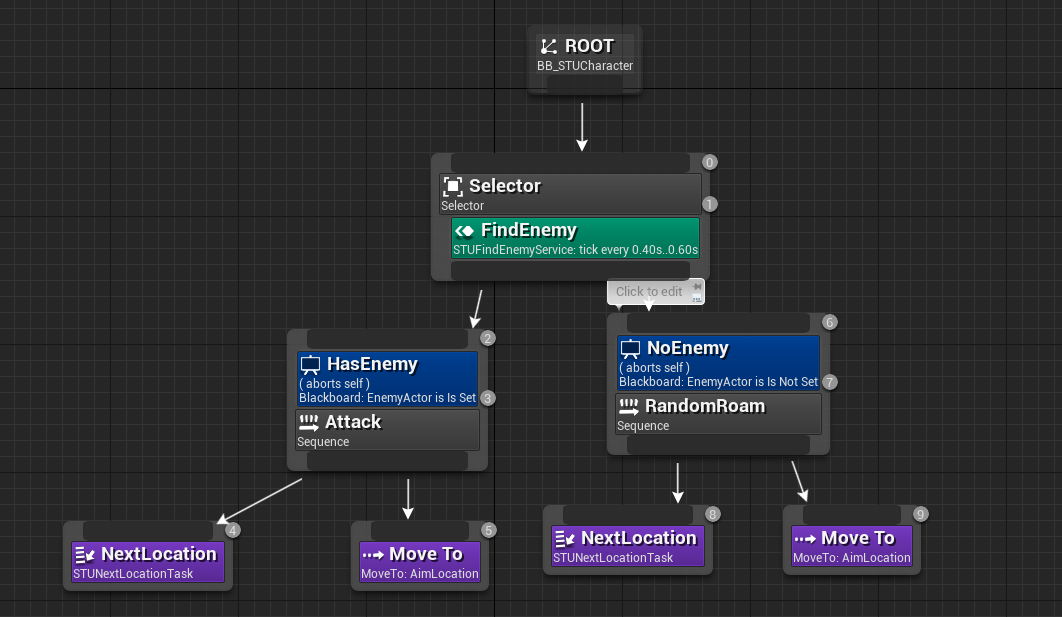
В ЗФ данного таска добавим проперти – она будет показывать, хотим мы найти новую точку на навигационном меше относительно себя или другого актора. И еще одно проперти, хранящее ключ для объекта, относительно которого мы будем находить рандомную точку.

Изображение выглядит как текст

Автоматически созданное описание

В cpp-файле создаем дополнительную переменную, которую мы будем передавать в функцию GetRandomReachablePointInRadius. Будем ее менять в зависимости от нашей булевой переменной.





В дереве поведения удаляем ноду MoveTo, добавляем такие же две ноды, как и для RandomRoam, только у NextLocation выключаем SelfCenter и выбираем EnemyActor.