2단계_12. 행동트리 모델의 구현

수강일	@2023/07/12
이름	전영재
멘토	Min-Kang Song,현웅 최
작성 상태	In progress
강의 시청 여부	☑

Contents

행동트리 모델의 구현

> 💡 **강의 목표**

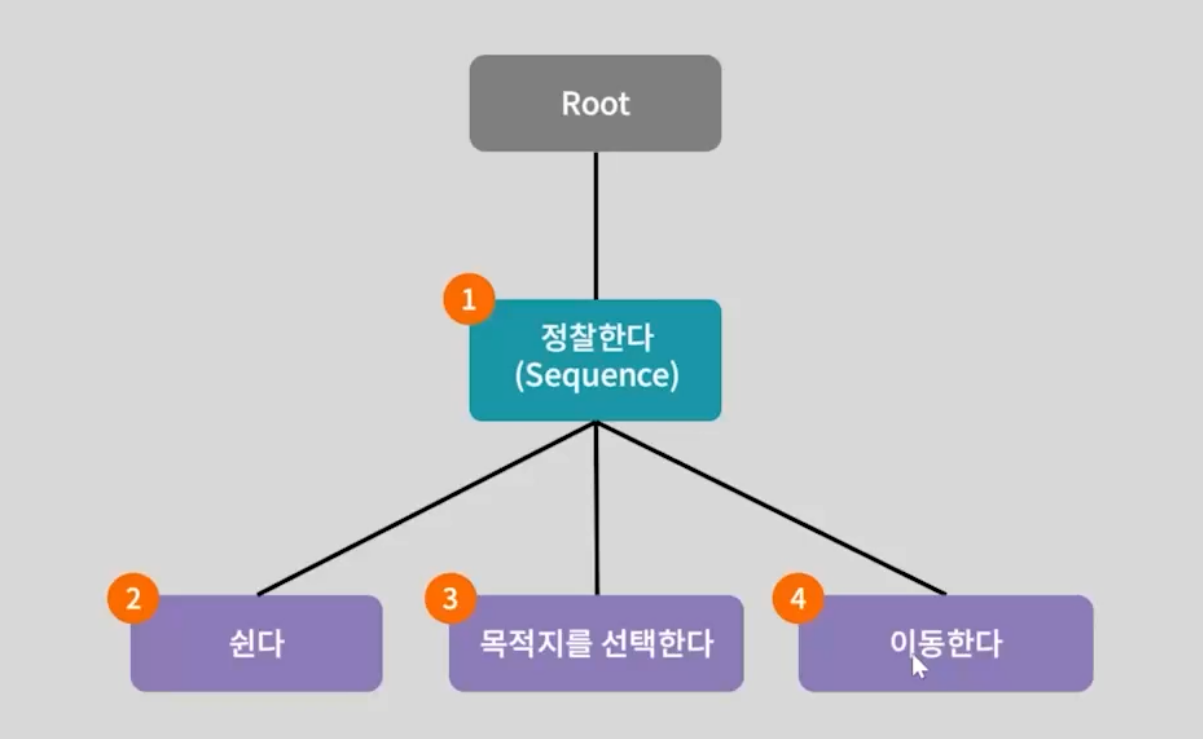
- NPC의 행동트리 모델을 기획하고 언리얼 엔진에서 구현

---

[1] NPC 행동트리 모델

이번 시간에는 전형적인 RPG게임 NPC의 행동트리 모델을 만들어 볼 것이다.

대기-탐색-이동을 반복하는 간단한 구조를 가질 예정이다.

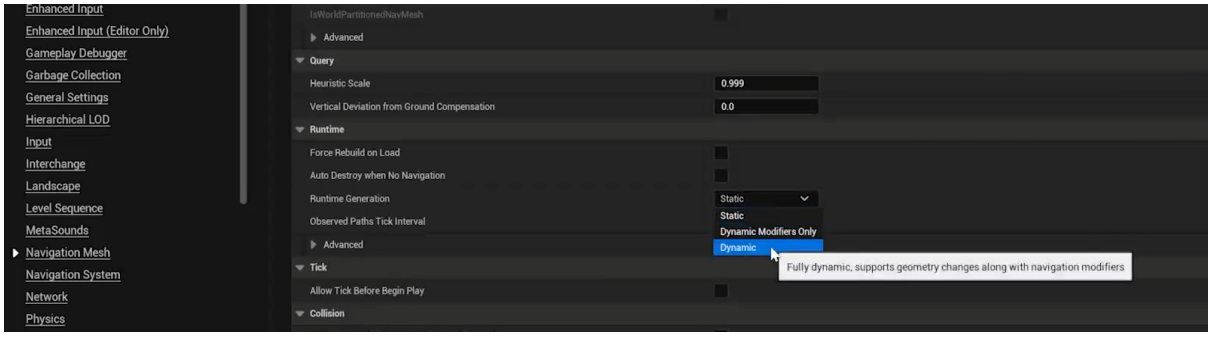


[2] 정찰 Task 구현

Dynamic Navigation Mesh Bounds Volume

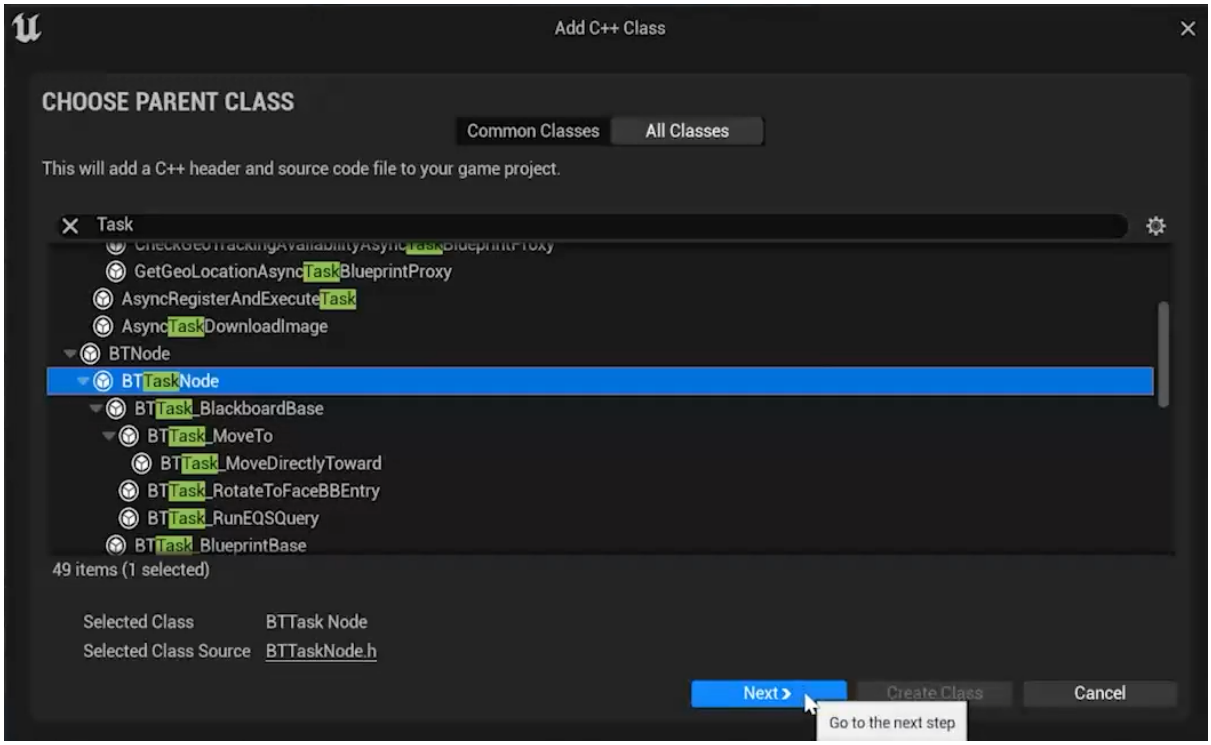
정찰 Task를 구현하기 위해서는 당연하게도 **정찰할 위치에 대한 데이터**가 필요하다. 언리얼엔진에서는 **NavigationMesh**를 제공하며, 이를 통해 AI가 도달할 수 있는 영역에 대한 정보를 쉽게 얻을 수 있다.

이런 NavigationMesh는 게임 제작과정에서 구축된 배경에 따라 **정적인(Static) 영역에 맞춰 길찾기 영역이 생성**되는데, 게임의 진행에 맞춰 새로운 영역이 생성되는 우리의 프로젝트 같은 경우에는 위와 같은 방법이 적절하지 못하다. 그러므로 **동적으로(Dynamic) 길찾기 영역을 생성**하는 설정을 해줘야 한다.



Project Settings - Navigation Mesh - Runtime - Runtime Generation 의 설정을 Dynamic으로 바꿔줘야 한다.

BTTask 노드의 생성



우리의 의도에 맞는 Task의 생성을 하려면 위와 같이 `BTTaskNode`를 상속받는 C++클래스를 생성해야 한다.

> 💡 **BTTaskNode**와 달리 **BTTask_BlackboardBase**는 에디터 내의 BT에서 **Blackboard의 데이터를 프로퍼티로 사용**하여 이를 수정하면서 사용하는 Task노드를 만들 수 있다.
> 지금 같은 경우는 그러한 프로퍼티로써 PatrolPos만을 사용할 것이 분명하므로 굳이 추가하지 않은 것이다.

> 💡 Task 클래스를 작성하여 사용하고 싶다면 다음과 같은 모듈들을 추가해야 한다.
> **'NavigationSystem', 'AIModule', 'GameplayTasks'**

Task에서 주로 사용되는 함수와 타입들이 독특하므로 해당 코드의 전문을 가져왔다.

후에 Task를 작성할 일이 있다면 참고할 양식이 될 것이다.

(그러나 World체크, 소유자 폰 체크, 인터페이스 체크 등 확인해야 할 요소가 많기에 코드가 불가피하게 길어져서 다음 코드들은 토글 내에 첨부하였다.)

▼ 코드

```
#include "AI/BTTask_FindPatrolPos.h"
#include "ABAI.h"
#include "AIController.h"
#include "NavigationSystem.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "Interface/ABCharacterAIInterface.h"

UBTTask_FindPatrolPos::UBTTask_FindPatrolPos()
{
}

EBTNodeResult::Type UBTTask_FindPatrolPos::ExecuteTask(UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory)
{
  //실행될 내용은 ExecuteTask 함수내에 오버라이드하여 구현한다.
  EBTNodeResult::Type Result = Super::ExecuteTask(OwnerComp, NodeMemory);

  APawn* ControllingPawn = OwnerComp.GetAIOwner()->GetPawn(); //제대로 폰을 소지하고 있는지 확인
  if (nullptr == ControllingPawn)
  {
    return EBTNodeResult::Failed; //결과의 리턴이 굉장히 중요하다. 실패 혹은 중단 되었음을 꼭 반환해야 한다.
  }
```

```
  //Nav시스템이 업데이트 되어 다음과 같이 UNavigationSystemV1이라는 독특한 이름이 되었다.
  UNavigationSystemV1* NavSystem = UNavigationSystemV1::GetNavigationSystem(ControllingPawn->GetWorld());
  if (nullptr == NavSystem)
  {
    return EBTNodeResult::Failed;
  }

  IABCharacterAIInterface* AIPawn = Cast<IABCharacterAIInterface>(ControllingPawn);
  if (nullptr == AIPawn)
  {
    return EBTNodeResult::Failed;
  }

  FVector Origin = OwnerComp.GetBlackboardComponent()->GetValueAsVector(BBKEY_HOMEPOS);
  float PatrolRadius = AIPawn->GetAIPatrolRadius();
  FNavLocation NextPatrolPos;

  if (NavSystem->GetRandomPointInNavigableRadius(Origin, PatrolRadius, NextPatrolPos))
  {
    OwnerComp.GetBlackboardComponent()->SetValueAsVector(BBKEY_PATROLPOS, NextPatrolPos.Location);
    return EBTNodeResult::Succeeded; //성공 반환
  }

  return EBTNodeResult::Failed;
}
```

공격 태스크

공격 태스크는 한번의 흐름으로 끝나는 다른 명령들과는 달리 **공격 몽타주가 끝날 때까지 기다려야** 한다는 차이점이 있다.

그렇기에 태스크를 성공해도 바로 `EBTNodeResult::Succeeded`를 반환하는게 아니라 진행중이라는 의미로 `EBTNodeResult::InProgress`를 반환해야 한다. 이 후 공격의 끝을 추적하는 방식으로는 Tick을 사용하는 방법도 있겠지만 **델리게이트**를 사용하면 좀 더 스마트하게 구현할 수 있다.

(의외로 몽타주의 EndDelegate를 사용하면 될 것이라 생각했지만 강의에서는 몽타주의 사용을 껄끄럽다고 표현하였다.)

1. 태스크 내에 델리게이트와 `FinishLatentTask`를 바인드해준다.

   (정확히는 FinishLatentTask를 호출하는 람다식과 바인드했다. 하지만 람다식의 경우에는 개인적으로 디버깅 시 추적이 어렵다 생각하기에 별도의 함수를 만드는 것이 좋다고 생각한다.)

2. NPC는 해당 델리게이트의 주소값을 가져와 자신의 공격끝 함수에서 델리게이트 방송을 시도한다.

즉, **캐릭터에서** 태스크의 델리게이트를 발동시키는 것이다.

```
...
void AABCharacterNonPlayer::SetAIAttackDelegate(const FAICharacterAttackFinished& InOnAttackFinished)
{
  OnAttackFinished = InOnAttackFinished;
}
void AABCharacterNonPlayer::NotifyComboActionEnd()
{
  Super::NotifyComboActionEnd();
  OnAttackFinished.ExecuteIfBound(); //헤더에서 델리게이트 선언했다. 헤더코드까지 넣고싶진 않아 그부분은 생략했다.
}
```

```
#include "AI/BTTask_Attack.h"
#include "AIController.h"
#include "Interface/ABCharacterAIInterface.h"

UBTTask_Attack::UBTTask_Attack()
{
}

EBTNodeResult::Type UBTTask_Attack::ExecuteTask(UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory)
{
  EBTNodeResult::Type Result = Super::ExecuteTask(OwnerComp, NodeMemory);

  APawn* ControllingPawn = Cast<APawn>(OwnerComp.GetAIOwner()->GetPawn());
  if (nullptr == ControllingPawn)
  {
    return EBTNodeResult::Failed;
  }

  IABCharacterAIInterface* AIPawn = Cast<IABCharacterAIInterface>(ControllingPawn);
```

```
  if (nullptr == AIPawn)
  {
    return EBTNodeResult::Failed;
  }

  FAICharacterAttackFinished OnAttackFinished
  OnAttackFinished.BindLambda(
    [&]()
    {
      FinishLatentTask(OwnerComp, EBTNodeResult::Succeeded); //후에 OnAttackFinished델리게이트가 오면 늦은 Succeeded반환
    }
  );

  AIPawn->SetAIAttackDelegate(OnAttackFinished); //NPC에게 태스크의 델리게이트 주소를 전달. 아래있지만 위 코드는 늦게 발동할 것이기에 순서는 문제없다.
  AIPawn->AttackByAI();
  return EBTNodeResult::InProgress; //진행중이기에 일단 InProgress 반환
}
```

BTService 클래스의 생성

지정한 인터벌 주기로 추가적인 **부가 명령**을 수행하는 기능으로, 주기적으로 적을 색적하는 기능을 구현해 볼 것이다.

주기적으로 수행되기에 `TickNode` 함수에서 기능을 구현하게 된다.

▼ 코드

```
#include "AI/BTService_Detect.h"
#include "ABAI.h"
#include "AIController.h"
#include "Interface/ABCharacterAIInterface.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "Physics/ABCollision.h"
#include "DrawDebugHelpers.h"

UBTService_Detect::UBTService_Detect()
{
  //노드이름과 주기를 설정
  NodeName = TEXT("Detect");
  Interval = 1.0f;
}

void UBTService_Detect::TickNode(UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory, float DeltaSeconds)
{
  Super::TickNode(OwnerComp, NodeMemory, DeltaSeconds);

  APawn* ControllingPawn = OwnerComp.GetAIOwner()->GetPawn();
  if (nullptr == ControllingPawn)
  {
    return;
  }

  FVector Center = ControllingPawn->GetActorLocation();
  UWorld* World = ControllingPawn->GetWorld();
  if (nullptr == World)
  {
    return;
  }

  IABCharacterAIInterface* AIPawn = Cast<IABCharacterAIInterface>(ControllingPawn);
  if (nullptr == AIPawn)
  {
    return;
  }

  float DetectRadius = AIPawn->GetAIDetectRange();

  TArray<FOverlapResult> OverlapResults;
  FCollisionQueryParams CollisionQueryParam(SCENE_QUERY_STAT(Detect), false, ControllingPawn);
  bool bResult = World->OverlapMultiByChannel(
    OverlapResults,
    Center,
    FQuat::Identity,
    CCHANNEL_ABACTION,
    FCollisionShape::MakeSphere(DetectRadius),
    CollisionQueryParam
  );

  if (bResult)
  {
    for (auto const& OverlapResult : OverlapResults)
```

```
    {
      APawn* Pawn = Cast<APawn>(OverlapResult.GetActor());
      if (Pawn && Pawn->GetController()->IsPlayerController())
      {
        OwnerComp.GetBlackboardComponent()->SetValueAsObject(BBKEY_TARGET, Pawn);
        DrawDebugSphere(World, Center, DetectRadius, 16, FColor::Green, false, 0.2f);

        DrawDebugPoint(World, Pawn->GetActorLocation(), 10.0f, FColor::Green, false, 0.2f);
        DrawDebugLine(World, ControllingPawn->GetActorLocation(), Pawn->GetActorLocation(), FColor::Green, false, 0.27f);
        return;
      }
    }
  }

  OwnerComp.GetBlackboardComponent()->SetValueAsObject(BBKEY_TARGET, nullptr);
  DrawDebugSphere(World, Center, DetectRadius, 16, FColor::Red, false, 0.2f);
}
```

BTDecorator 클래스의 생성

주어진 **프로퍼티**를 검사하여 컴포짓 노드의 **실행 조건**을 지정하는데 사용된다. 이것을 통해 Target이 내 공격 범위 내에 있는지 검사할 것이다.

프로퍼티를 검사하기에 `CalculateRawConditionValue` 함수를 사용한다.

▼ 코드

```
#include "AI/BTDecorator_AttackInRange.h"
#include "ABAI.h"
#include "AIController.h"
#include "BehaviorTree/BlackboardComponent.h"
#include "Interface/ABCharacterAIInterface.h"

UBTDecorator_AttackInRange::UBTDecorator_AttackInRange()
{
  NodeName = TEXT("CanAttack");
}

bool UBTDecorator_AttackInRange::CalculateRawConditionValue(UBehaviorTreeComponent& OwnerComp, uint8* NodeMemory) const
{
  bool bResult = Super::CalculateRawConditionValue(OwnerComp, NodeMemory);

  APawn* ControllingPawn = OwnerComp.GetAIOwner()->GetPawn();
  if (nullptr == ControllingPawn)
  {
    return false;
  }

  IABCharacterAIInterface* AIPawn = Cast<IABCharacterAIInterface>(ControllingPawn);
  if (nullptr == AIPawn)
  {
    return false;
  }

  APawn* Target = Cast<APawn>(OwnerComp.GetBlackboardComponent()->GetValueAsObject(BBKEY_TARGET));
  if (nullptr == Target)
  {
    return false;
  }

  float DistanceToTarget = ControllingPawn->GetDistanceTo(Target);
  float AttackRangeWithRadius = AIPawn->GetAIAttackRange();
  bResult = (DistanceToTarget <= AttackRangeWithRadius);
  return bResult;
}
```

📌 **Summary**

- BT노드를 사용하기 위한 블랙보드 설정법 (모듈, NavMesh설정…)
- 종류별 BT노드의 생성 및 사용법

- 일반태스크와 지연태스크의 제작과 활용
- 인터페이스를 활용한 AI와 캐릭터의 설계 분리

0n. 행동트리 모델의 구현 강의 과제

❓ **Q1. 행동 트리 모델을 구현하고 이를 직접 디버깅하면서 동작과정을 분석하시오.**

❓ **Q2. 예시의 행동 트리를 보다 발전시킬 방법은 없는지 생각하시오.**