This post is also available in the following languages. Japanese

Behavior Tree를 알아봅시다

overlast 2018-10-18
LINE Engineer

안녕하세요. 저는 Clova를 구성하는 시스템 가운데 NLU(Natural Language Understanding,자연어 이해)파트의 서버 쪽 개발을 담당하고 있는 @overlast입니다.

얼마전에 Youichiro Miyake(三宅陽一郎)씨와 대담(일본어)을 나눌 기회가 있었는데요. 이야기하다보니 게임 업계에서 캐릭터 AI 등을 만들 때 자주 사용되는 Behavior Tree(행동 트리) 모델을 저는 아직 한번도 다뤄본 적이 없다는 사실을 깨달았습니다. 이에 '좋은 기회다' 싶어 이참에 Behavior Tree와 기존 구현체를 사용해 보고 이 내용을 정리하게 되었습니다.

Behavior Tree 소개

우선 Wikipedia에서 Behavior Tree 페이지를 한번 참고해주시기 바랍니다.

Behavior Tree 개요

Behavior Tree 모델을 작심하고 설명하면 글이 너무 길어지기 때문에, 대략적인 이해에 도움이 되는 중요 포인트만 추려서 다음과 같은 목록으로 정리해보았습니다.

AI 분야의 Reactive Planning에서 발생함
- 2004년에 출시된 게임 Halo 2는 당시 Reactive Planning을 도입한 게임 중 하나임

게임 제작 시 AI를 효율적으로 구현할 수 있도록 실제 사례가 풍부함

행동(Behavior)을 트리(tree) 구조로 기술함
- 구축이 완료된 Behavior Tree의 데이터 구조는 DAG 또는 트리임

한 덩어리의 태스크가 서브트리(sub tree)를 이룸
- 서브 태스크, 서브 트리, 블록이라고도 함

트리의 구성 요소는 다음과 같이 크게 셋으로 나뉨
- root 노드 : 루트 노드
- control flow 노드 : 루트도 리프(leaf)도 아님
- execution 노드 : 리프 노드. 태스크라고도 함

평가 시 각 노드는 깊이 우선 탐색됨

탐색 결과, 자식 노드에서 부모 노드로 상태가 반환됨
- Success : 실행 성공
- Failure : 실행 실패
- Running(Continue) : 실행 중. 다음 번에 running을 반환한 노드가 다시 호출됨

모든 노드에는 평가 가능 여부를 나타내는 active/inactive 상태를 설정할 수 있음

Search for

Tags

Technical Field
#Client-side
#Web Develo
#Server-side
#QA #Syster
#VoIP #Data
#Security En
#Tech Mana
#Technical W

Programming
#JavaScript
#Go #Kotlin

Technical Tern
#Automatior
#CodeReado
#Agile #Opti
#Testing #lo
#Site Reliabil

Developer Too
#Kubernetes
#Vue.js #Spo
#Elasticsear
#Node.js

Tech Share
#Open Sourc
#LINE Events
#LINE DEVEL
#Sponsorshi
#Event Repo

Behavior Tree를 사용해 AI와 같은 동작을 실현하고자 하는 경우, 아래 이어지는 control flow 노드와 execution 노드 등 각 노드의 역할과 사용 방법을 알아두면 좋을 듯합니다.

Control flow 노드

Control flow 노드의 역할은 그 노드 아래에 있는 서브 트리(브랜치라 불림)를 통해 표현되는 행동(서브 태스크)을 제어하는 것입니다.

기존 구현체를 살펴 보면 다음 두 종류의 노드는, 실현 방식이나 네이밍 방식은 조금 다르더라도, 모든 구현체에 공통적으로 존재합니다.

Selector

자식 노드 가운데 하나를 실행하기 위한 노드입니다. 평가를 시작하면 selector의 자식 노드를 왼쪽에서 오른쪽(우선도가 높은 쪽에서 낮은 쪽)의 순서로 깊이 우선 탐색 방식으로 평가합니다.

Selector의 자식 노드 중 하나가 success나 running을 반환하면, selector는 즉시 success나 running을 부모 노드에 반환합니다. Selector의 모든 자식 노드가 failure를 반환했을 때는 selector도 부모 노드에 failure를 반환합니다.

Sequence

자식 노드를 순서대로 실행하기 위한 노드입니다. 평가를 시작하면 sequence의 자식 노드를 왼쪽에서 오른쪽(우선도가 높은 쪽에서 낮은 쪽)의 순서로 깊이 우선 탐색 방식으로 평가합니다.

Sequence의 자식 노드 중 하나가 failure나 running을 반환하면 selector는 즉시 failure나 running을 부모 노드에 반환합니다. Selector의 모든 자식 노드가 success를 반환했을 때는 selector도 부모 노드에 success를 반환합니다.

Composite

위 두 노드는 공통적으로 브랜치의 root가 됩니다. 또한 디자인 패턴의 composite 패턴과 같은 동작을 실현하므로 composite 유형으로 분류될 때가 있습니다.

실제 사용할 때는 selector와 sequence 노드 각각에 무작위나 확률을 도입해야 할 것으로 보입니다. 또 selector와 sequence 노드 외에도 서브 트리를 Behavior Tree로 병렬 평가(부모의 Behavior Tree가 종료되면 강제 종료)하기 위한 병렬 처리 계열 노드, 기존에 실행한 적 없는 자식 노드만 평가하기 위한 노드가 필요하겠지요.

하지만 이것만으로는 if문이나 while문 등의 프로그래밍 언어에서 자주 사용하는 제어문을 control flow 노드로 표현할 수가 없습니다.

Decorator

제어문을 실현하기 위한 노드는 디자인 패턴의 decorator 패턴과 같은 동작을 실현하기 때문에 decorator 유형으로 분류될 때가 있습니다. 혹은 call이나 condition으로 분류되기도 합니다.

다음은 Behavior Tree의 기존 구현체에 실현되어 있는, 또는 실현 가능한 대표적인 조건식들입니다.

While(Conditional Loop)

조건 충족을 평가하는 함수가 true를 반환하는 한, 자식 노드의 평가를 반복합니다. 조건 충족을 나타내는 함수가 failure를 반환하면 while도 부모 노드에 failure를 반환합니다.

If

조건 충족을 평가하는 함수가 true를 반환한 경우, 현재 유효한 자식 노드를 호출해서 그 결과를 반환합니다. 그 밖의 경우에는 자식 노드 호출 없이 부모 노드에 failure를 반환합니다.

Repeat(Loop)

미리 설정해 둔 횟수만큼 자식 노드 평가가 완료되었다면 부모 노드에 success를 반환합니다. 그 밖의 경우에는 부모 노드에 running을 반환합니다.

기타 노드

위 조건식으로 if문, while문, for문까지 구현할 수 있는데요. 그 밖에도 처리 중단 및 인터럽트용 노드도 있으면 좋겠습니다.

또 Behavior Tree 내 노드 간에 각 노드의 실행 결과를 필요에 따라 저장하고 공유할 수 있는 메모리 영역과, 메모리 영역에 저장된 데이터를 이용하여 제어를 구현하는 노드도 액션의 결과를 활용하는 수단으로 필요합니다.

Execution 노드

Execution 노드는 액션(action)이라고도 합니다. 액션은 리프(leaf) 노드로 작성되며 Behavior Tree 외부에 구현된 구체적인 함수를 호출합니다.

이 함수의 반환값에 따라 실행 결과의 상태(success, failure, running 등)를 판단하여 그 반환값을 부모 노드에 반환합니다.

Behavior Tree 예제

여기까지 설명한 내용을 바탕으로 '수중에 지닌 돈과 기분에 따라 주변 자판기에서 음료 구입하기' 행동을 아래 그림 1처럼 Behavior Tree로 작성하였습니다.

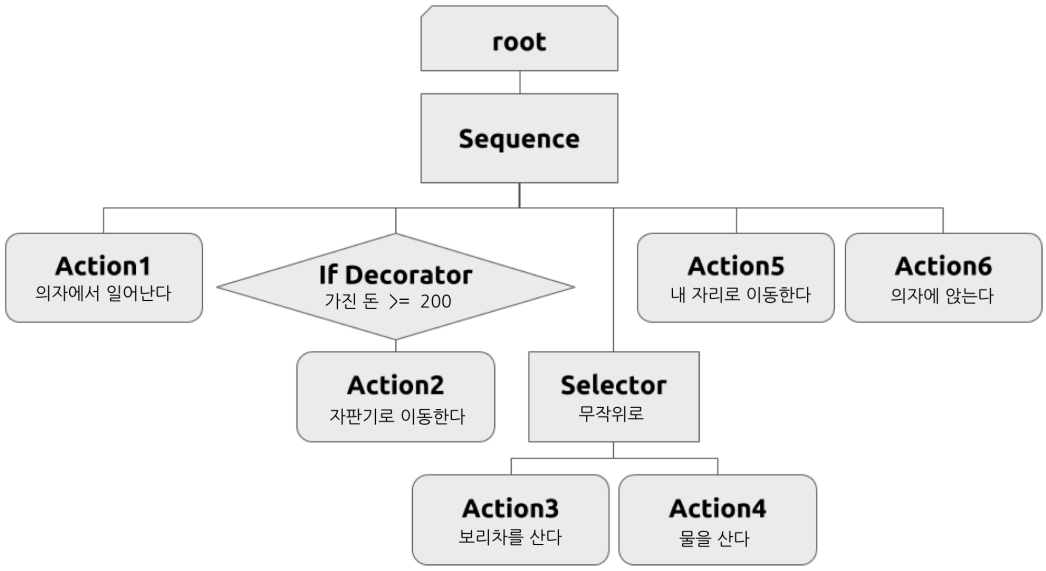

그림 1: '수중에 지닌 돈과 기분에 따라 주변 자판기에서 음료 구입하기'의 Behavior Tree

이 Behavior Tree를 평가하는 함수는 입력되는 인자에 '가지고 있는 돈의 액수'에 관한 정보가 있어야 합니다.

액수가 0원일 때와 300원일 때 행동이 어떻게 달라지는지 간단히 살펴보겠습니다.

Case 1 : 수중에 돈이 한푼도 없을 때 ⇒ 음료 구입 실패

먼저 돈이 한 푼도 없을 때 각 step별 행동은 다음과 같습니다. 그림 2는 모든 step을 수행하고 난 뒤의 결과를 보여줍니다.

Step1 : Root에서 출발

Step2 : Sequence로 이동. 이후, 깊이 우선 탐색을 계속함

Step3 : Action1으로 이동. 의자에서 일어나는 데 성공하고 success를 반환함

Step4 : If의 decorator로 이동. 갖고 있는 돈의 액수가 0원이므로 failure를 반환함

Step5 : Sequence로 다시 이동. Failure가 돌아왔으므로 탐색을 마치고 failure를 반환함

Step6 : Behavior Tree 갱신 결과 failure를 반환하고 종료

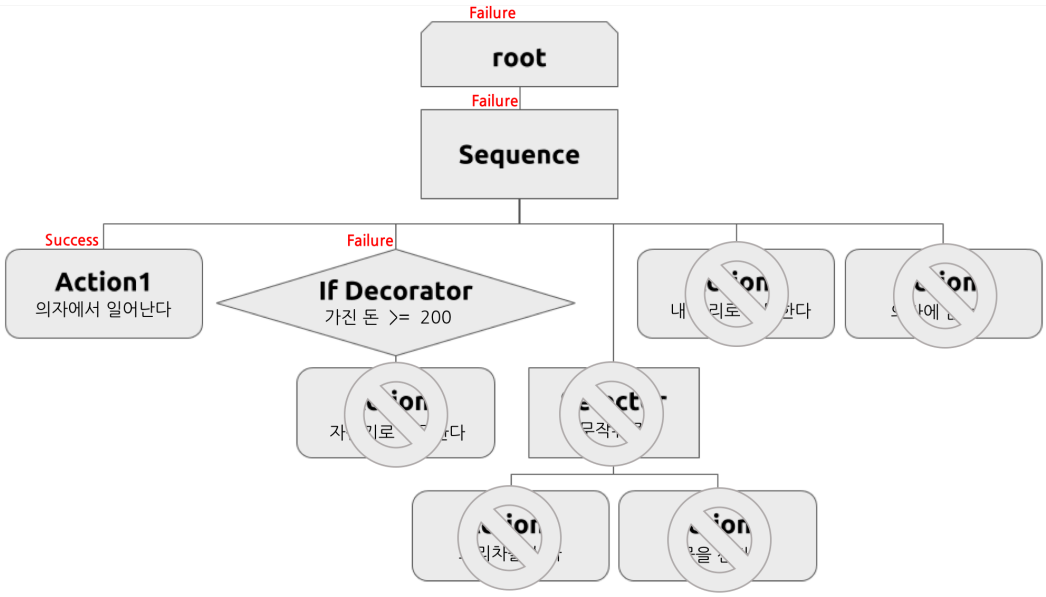

그림 2 : '돈은 없지만 자판기에서 음료를 사고 싶을 때'의 Behavior Tree

Step5에서 서브 태스크의 순차적인 실행에 실패한 sequence가 트리 탐색을 종료해 버렸습니다. 돈이 없으면 자판기 쪽으로 가는 의미가 없지요. 그렇다면 돈이 300원 있을 때는 어떨까요?

Case 2 : 수중에 돈이 300원 있을 때 ⇒ 음료 구입 성공

이어서 돈이 300원 있을 때 각 step별 행동입니다. 아래 그림 3은 역시 모든 step을 수행하고 난 뒤의 결과를 보여줍니다.

Step1 : Root에서 출발

Step2 : Sequence로 이동. 이후, 깊이 우선 탐색을 계속함

Step3 : Action1로 이동. 의자에서 일어나는 데 성공하고 success를 반환함

Step4 : If decorator로 이동. 갖고 있는 돈의 액수가 300원이므로 If decorator에 설정된 조건을 충족함

Step5 : Action2로 이동. 자판기로 이동하는 데 성공하고 success를 반환함

Step6 : If decorator로 다시 이동. 자식 노드의 state인 success를 반환함

Step7 : 무작위가 적용된 selector로 이동. 자식 노드 중 하나로 이동함

Step8 : 무작위로 Action4로 이동. 물을 사는 데 성공하고 success를 반환함

Step9 : Selector로 이동. Success가 반환됐으므로 탐색을 종료하고 success를 반환함

Step10 : Action5로 이동. 자기 자리로 이동하는 데 성공하고 success를 반환함

Step11 : Action6로 이동. 의자에 앉는 데 성공하고 success를 반환함

Step12 : Sequence로 이동. 모든 자식 노드가 success를 보내왔으므로 success를 반환함

Step13 : Root로 이동. Behavior Tree 갱신 결과 success를 반환하고 종료

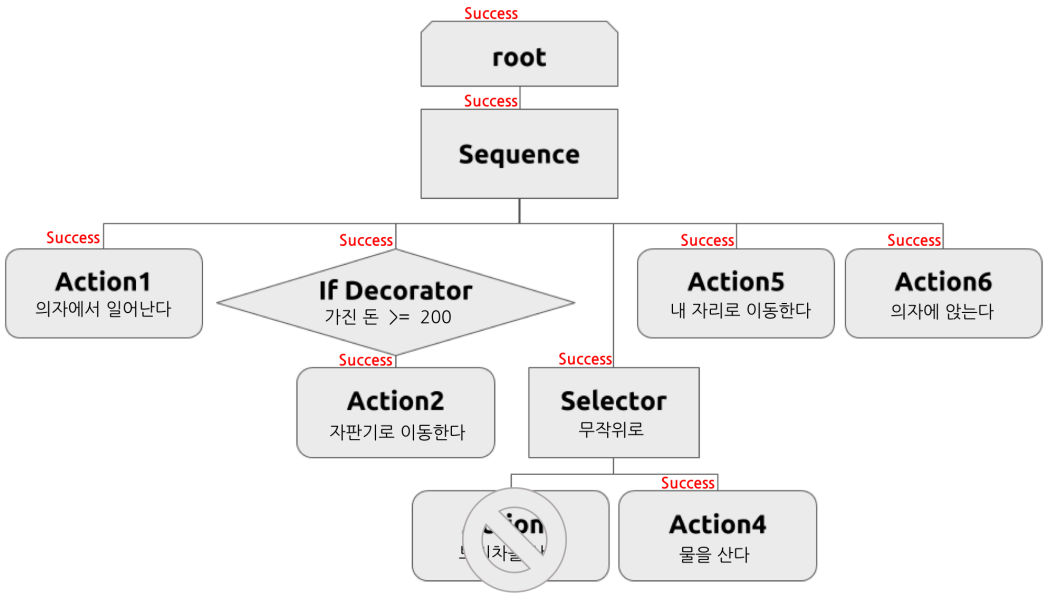

그림 3 : '돈이 300원 있어 자판기에서 음료를 살 수 있을 때'의 Behavior Tree

상당 부분 생략했는데도 0원일 때 step수의 두배가 넘습니다.

이 case에서 selector의 서브 태스크 선택 방식을 꼭 무작위로 할 필요는 없습니다. 하지만, 실제로 매번 실행할 때마다 AI의 행동을 변화시키기 위해서는 무작위같이 우선순위 외에 방법도 있어야겠지요.

Behavior Tree 적용 시 고려해볼 만한 점

게임 AI에 적용 시

게임에 구현되는 AI에서는 'tick'이라고 하는 업데이트 처리가 이루어질 때마다 Behavior Tree가 평가됩니다. 이 점을 감안해서 원활히 돌아갈 수 있도록, 또 선택된 행동이 불필요하게 수습되지 않도록 Behavior Tree 구성을 잘 생각해야 합니다.

예를 들어, 그림 1처럼 Behavior Tree가 단순한 경우, 몇 번의 업데이트 후 돈을 다 써버리고 나면 이후 업데이트 처리를 할 때마다 수중에 돈이 0원 있을 때와 같은 동작만 하게 됩니다. 자기 자리 앞에 그냥 서 있기만 하게 되는거죠. 또한 게임에서는 NPC나 적 캐릭터 등이 자연스럽고 지적으로 움직이는 것처럼 보여야 하므로 업데이트 처리는 매우 자주, 가령 1프레임에서 적어도 수 프레임마다는 이루어져야 합니다. 따라서 반드시 running 상태여야 합니다.

Behavior Tree의 라이브러리는 그래픽 기반의 개발 도구가 잘 갖추어져 있어 놀랍더군요. 이유는 여러 가지 있을 것 같은데, 게임 업계에서 쓰이는 라이브러리의 경우 개발 효율이 무척 중요하다 보니 시각적이고 다루기 쉬운 개발 도구가 많은 건지도 모르겠습니다.

대화 시스템에 적용 시

많은 대화 시스템에서는 대화가 한 마디 진행될 때마다 AI의 Behavior Tree가 한 번 업데이트되면 일단 충분한 것으로 봅니다. 서버쪽에서 암묵적으로 Behavior Tree가 자꾸 업데이트되면 인간 발화자는 AI의 말을 따라갈 수 없게 될 테니까요. 또 대화 중에 태스크가 줄곧 running 상태를 유지하면 대화가 멈춰버리므로, 어느 정도 시간이 경과한 running 상태의 태스크는 정지해야 할 것입니다.

맺으며

저도 아직 튜토리얼을 참고로 간단한 Behavior Tree를 구현한 정도라 공개하더라도 저 자신만 만족하는 게 아닐까 싶네요. 대신, 아래 참고 문헌에 나와 있는 튜토리얼을 따라해 보시기 바랍니다. 이번에 소개한 Behavior Tree는 업무에도 활용할 수 있는 내용이니 연관된 내용을 앞으로도 계속 소개할 수 있을 것입니다. 개발 성과 중 일부는 OSS(Open Source Software)로도 공개할 생각입니다.

참고 문헌

GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI
- 여러 명칭 중 하나로 "a behavior tree"라는 표현이 등장함

UNREAL ENGINE의 Behavior Tree 튜토리얼

Unity의 Behavior Tree 매뉴얼

The Behavior Tree Starter Kit, Alex J. Champandard, Philip Dunstan, Game AI Pro, 2013

aigamedev/btsk(Behavior Tree Starter Kit in C++)

miccol/Behavior Trees Library in C++

arvidsson/BrainTree - A C++ behavior tree header-only library

Behavior Tree