14_ 순수 함수와 형변환

<제목 차례>

인천대학교 컴퓨터공학부 박종승
무단전재배포금지

1. 개요

이 장에서는 순수 함수와 형변환에 대해서 학습한다.

먼저, **순수 함수**에 대해서 알아보자.
클래스의 멤버 변수를 읽기만 하고 수정하지 않는 함수를 **순수**(pure) 함수라고 하고 그렇지 않은 함수를 비순수(impure) 함수라고 한다.
순수 함수는 사실상 실행핀이 필요없다. 순수 함수의 결과값이 연결된 노드가 실행될 때에 자동으로 순수 함수를 실행해서 결과값을 연산하면 되기 때문이다.
에디터에서 함수를 순수 함수로 명시적으로 지정해두면 함수 노드의 실행핀이 사라진다. 따라서 순수 함수 노드는 실행핀이 없는 **Get** 노드와 비슷하게 생각하면 된다.
순수 함수로 지정해두면 실행핀이 없어지므로 스크립트가 더 단순해지고 또한 멤버 변수를 수정하지 않는다는 것이 명료해진다. 따라서 구현한 함수가 순수 함수인 경우에는 순수 함수로 지정되도록 해주자.

2. 순수 함수와 함수로 접기

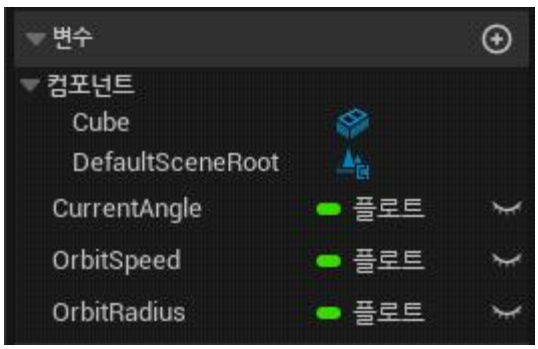
이 절에서는 순수 함수에 대해서 학습하고, 함수로 접기 기능에 대해서 학습한다.

---

1. 새 프로젝트 Pfuncapp를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pfuncapp으로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다. 창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 MyMap으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,112)로 지정하자.

2. 이제부터, 블루프린트 클래스를 만들자.
이전 예제에서 만들었던 스핀되는 큐브인 **BP_MyCube**를 동일한 방법으로 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyCube**으로 수정하자.
그다음, **BP_MyCube**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 디폴트로 **Cube**라는 이름으로 그대로 두자.

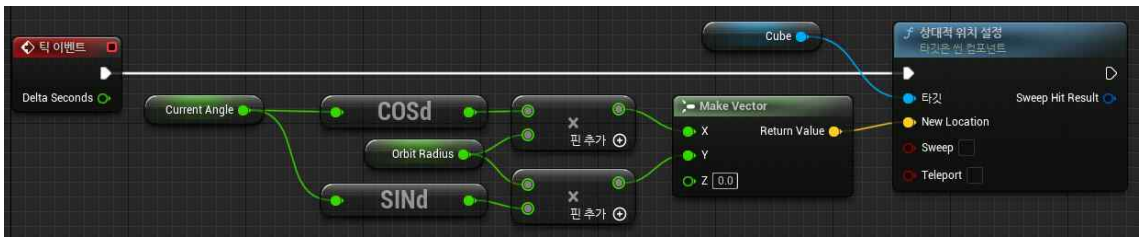
3. 큐브가 배치된 지점을 중심으로 고정된 반경의 원을 따라서 회전되도록 구현해보자. 배치된 지점과 동일한 높이의 XY평면과 평행한 평면상에서 회전되도록 하자. 회전 중심은 배치된 지점이다.
이를 위해서 변수를 추가해보자. **BP_MyCube** 블루프린트 에디터의 **내 블루프린트** 탭에서 **변수** 영역의 오른쪽의 + 아이콘을 클릭하여 추가하자. **CurrentAngle**, **OrbitSpeed**, **OrbitRadius**의 세 변수를 추가하자. 유형은 모두 **플로트**로 하자.



각 변수의 의미를 살펴보자. 먼저, **CurrentAngle**은 현재 회전 각도이다. 우리는 각도의 범위를 [0,360)로 하자. 0에서 시작하여 359까지 진행되면 한 바퀴를 회전하게 된다. 이후에 다시 0부터 반복한다.
그다음, **OrbitSpeed**는 초당 몇 도를 회전하는지를 정의한다. 180이라면 1초에 반 바퀴를 회전한다.
그다음, **OrbitRadius**는 회전 중심으로부터 회전 중인 큐브까지의 거리이다.
컴파일하자.
그다음, **OrbitSpeed**의 디테일 탭에서 **기본값**은 180으로 하자. 또한 **인스턴스 편집가능**에 체크하여 에디

터에서 기본값을 입력할 수 있도록 하자.
그다음, **OrbitRadius**의 디테일 탭에서 **기본값**은 200으로 하자.

4. 이제, 루트 컴포넌트에 상대적인 **Cube** 컴포넌트의 위치를 계산해보자. 기본적으로는 우리가 피봇을 수정하지 않았으므로 루트 컴포넌트에 상대적인 **Cube** 컴포넌트의 피봇의 위치가 (0,0,0)일 것이다. 현재 각도가 θ이고 반지름이 r이라면 **Cube** 컴포넌트의 회전 위치는 (cosθ*r, sinθ*r, 0)이다. 이 위치를 액터 내에서의 큐브의 상대 위치로 지정해주면 된다.
Tick 이벤트 노드에서 다음과 같이 그래프를 만들자.



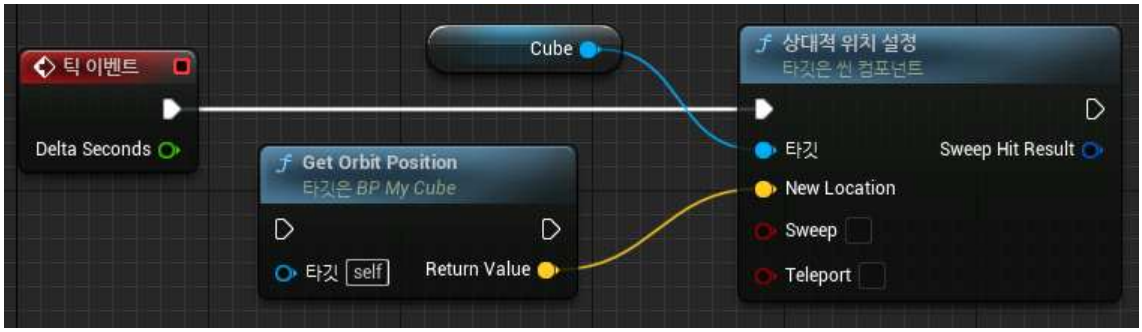
삼각함수는 라디안(radian) 단위값을 사용하는 버전과 도(degree) 단위값을 사용하는 버전이 모두 제공된다. 라디안 버전은 **SIN**과 같이 표시되며 도 버전은 **SINd**와 같이 **d**가 접미사로 붙어서 표시된다. **x**로 표시되는 **곱하기** 노드는 ***를 검색하여 배치하면 된다. **MakeVector**는 **Vector** 구조체를 만드는 구조체에 대해서 제공되는 **Make**류의 함수이다.
마지막으로, **SetRelativeLocation (Cube)** 노드를 배치하자. **Cube**의 **상대적 위치 설정**(**SetRelativeLocation**) 함수를 호출하면 계산된 위치 벡터를 액터의 루트에 상대적인 **Cube**의 피봇 위치로 지정해준다.

<참고> **Vector** 타입에 대해서도 **Break**류의 함수와 **Make**류의 함수가 있다. **BreakVector** 노드는 **Vector** 타입의 입력 인자를 받아서 세부 원소들인 X, Y, Z 값을 출력한다. 이를 반대로 수행하는 **MakeVector** 노드는 세부 원소들인 X, Y, Z 값을 입력 인자로 받아서 **Vector** 타입을 구성하여 출력한다.

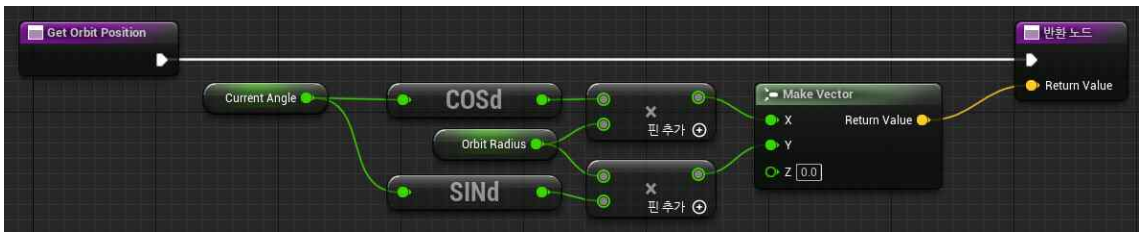
5. 스크립트를 작성하다 보면 노드 네트워크가 점점 커지게 된다. 이런 경우에는 노드 네트워크의 일부를 독립된 함수로 분리해주도록 하는 것이 좋다.
위의 노드 네트워크의 **Cube**의 피봇 위치를 계산하는 부분을 독립된 함수로 만들어보자. 먼저, **Current Angle**의 **Get** 노드부터 **MakeVector** 노드까지를 마우스로 모두 드래그하여 선택하자. 그다음, 우클릭하고 팝업메뉴에서 **함수로 접기**를 선택하자.

디폴트로 정해지는 함수 이름인 **NewFunction_0**을 **GetOrbitPosition**로 수정하자.

6. 노드 그래프는 아래와 같이 간단하게 바뀌었을 것이다.

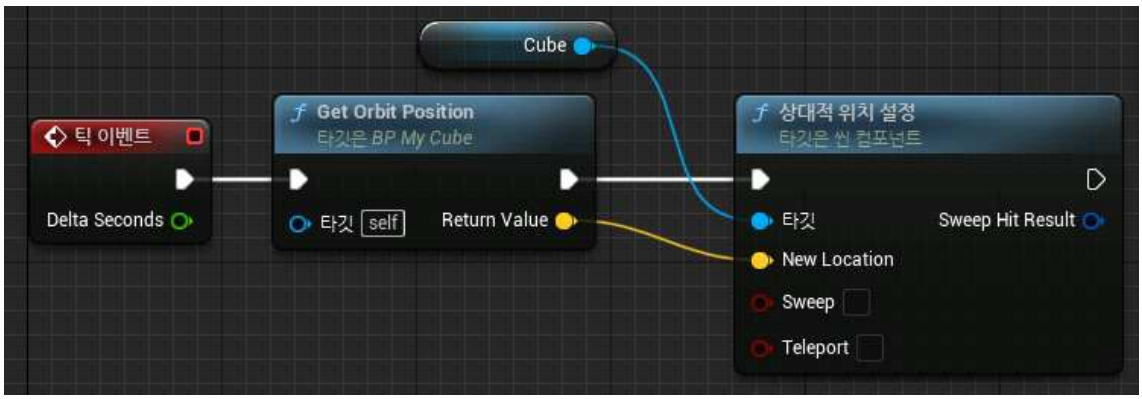


7. 또한, **GetOrbitPosition** 함수 그래프가 아래와 같이 생성되었을 것이다. 자동으로 배치되므로 정렬 형태는 다소 차이가 있을 것이다.



지금까지 **함수로 접기** 기능에 대해서 알아보았다. 편리한 기능이므로 숙지해두자.

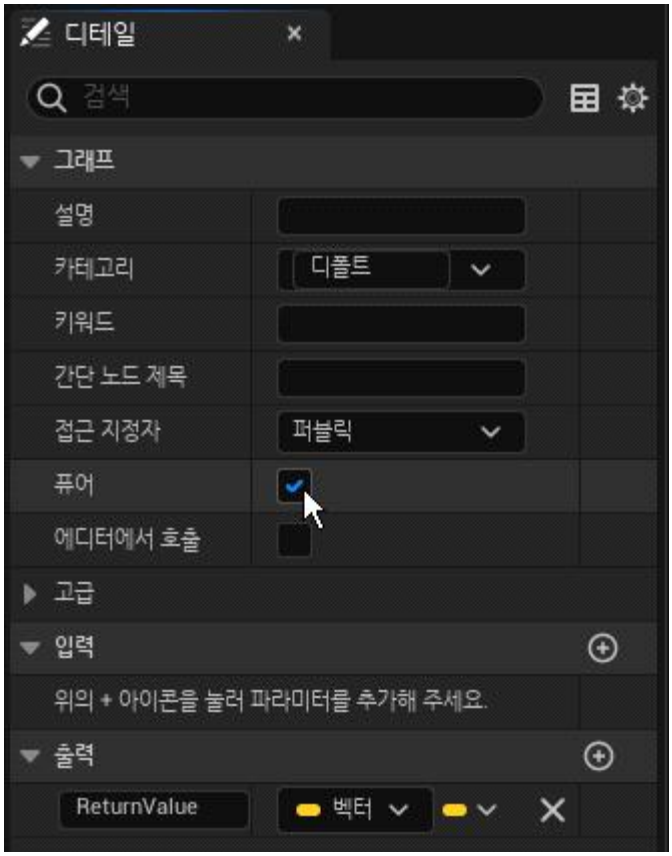
8. 컴파일해보자. 새로 배치된 **GetOrbitPosition** 함수 노드의 실행핀이 연결되지 않아서 경고가 발생할 것이다. 아래와 같이 실행핀을 연결하여 경고가 없어지도록 조치하자.



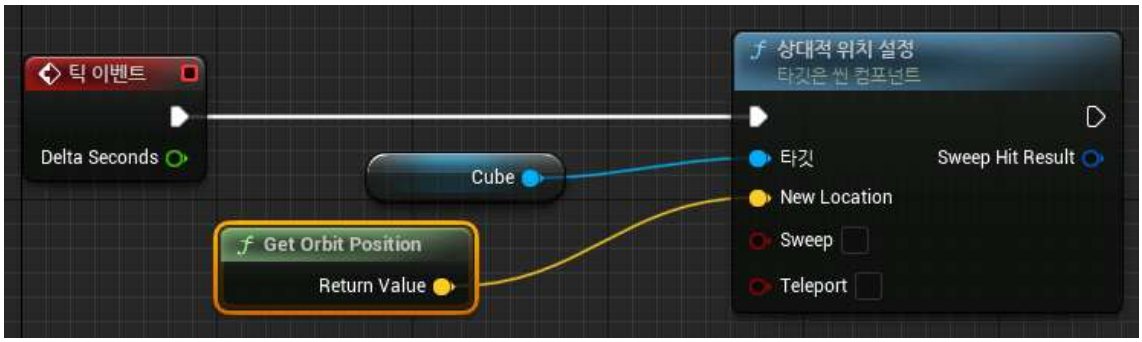
9. 이제부터 **순수 함수**에 대해서 학습해보자.

위의 예에서 **GetOrbitPosition** 함수 그래프를 살펴보자. 변수값을 읽기만 하고 수정하지는 않는다. 따라서 순수 함수에 해당한다.

GetOrbitPosition 함수 그래프를 열고 **GetOrbitPosition** 함수 노드를 클릭하고 디테일 탭을 보자. 순수 함수로 지정하는 체크박스인 **퓨어** 속성이 있을 것이다. 이것에 체크하자. 컴파일하자.

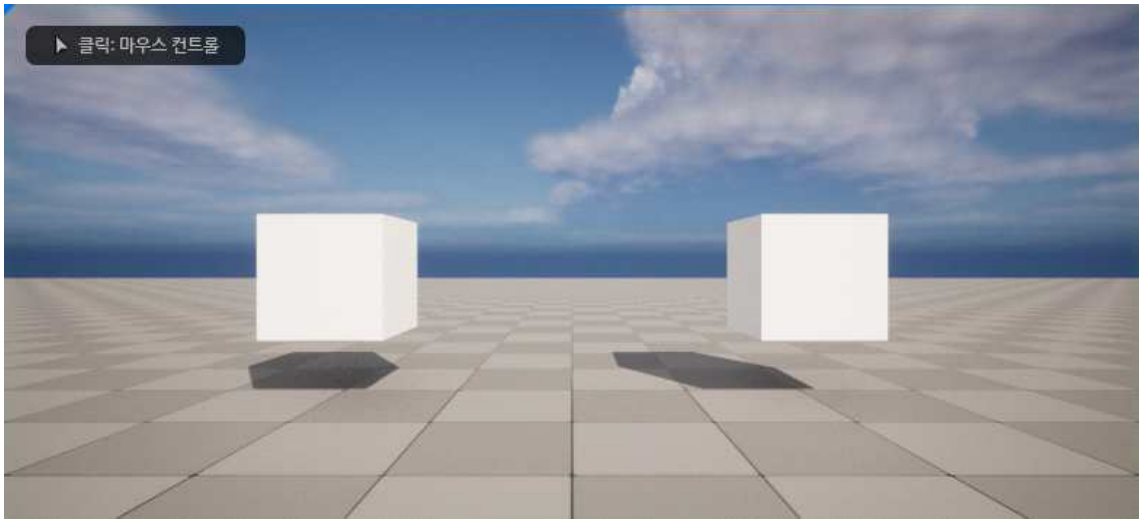


10. 이벤트 그래프로 이동해서 **Tick 이벤트** 그래프를 살펴보자. 아래와 같이 **GetOrbitPosition** 함수 노드의 실행핀이 사라진 것을 볼 수 있다. 또한 파란색이었던 노드 컬러가 녹색으로 바뀌었다.



컴파일하고 저장하자.
지금까지 순수 함수에 대해서 학습하였다.

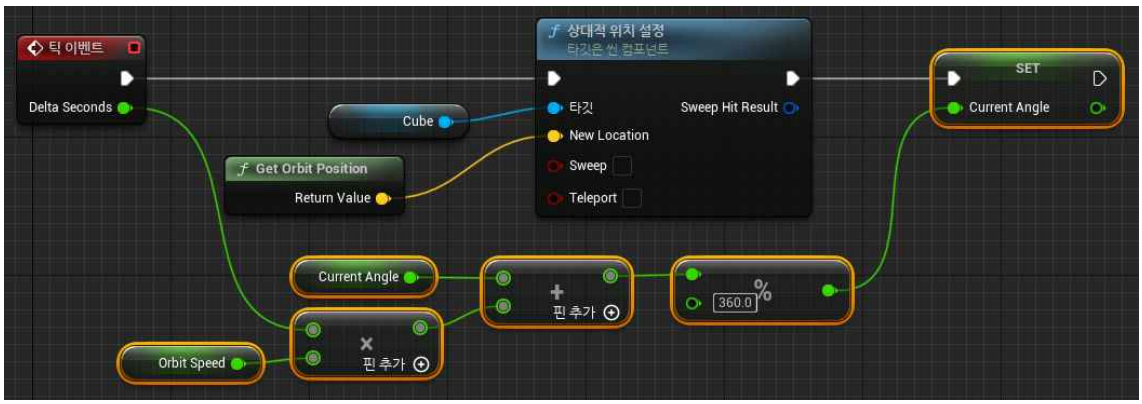
11. 레벨 에디터의 콘텐츠 브라우저에서 **BP_MyCube**를 드래그하여 레벨에 배치하자. 두 개를 배치하자. 각각 이름을 **BP_MyCube1**과 **BP_MyCube2**로 정하자.
그리고, 각각의 **위치**는 (300,-200,100)와 (300,200,100)으로 하자.
실행해보자. 배치된 큐브가 움직이지는 않을 것이다. 아직 **CurrentAngle**을 증가시키지 않았기 때문이다.



12. 이제, **CurrentAngle**을 증가시켜 움직이도록 해보자.

BP_MyCube 블루프린트 에디터로 가자.

Tick 이벤트 노드는 **DeltaSeconds**라는 출력핀을 제공한다. 이 값은 초 단위의 경과 시간이다. 하드웨어 성능과 무관하게 일정 속도의 움직임을 구현하려면 이 값을 사용하면 된다.



그래프를 살펴보자. 먼저 **OrbitSpeed**를 **DeltaSeconds**와 곱하면 초당 회전해야 할 각도가 계산된다. 그다음, 현재 회전 각도를 증가시킨다. 그다음, 360도가 넘으면 다시 0으로 되도록 나머지 연산자인 **%** 노드를 배치하고 두 번째 인자로 360을 지정한다. 그다음, 그 결과값을 **CurrentAngle**의 **Set** 노드에 지정한다.

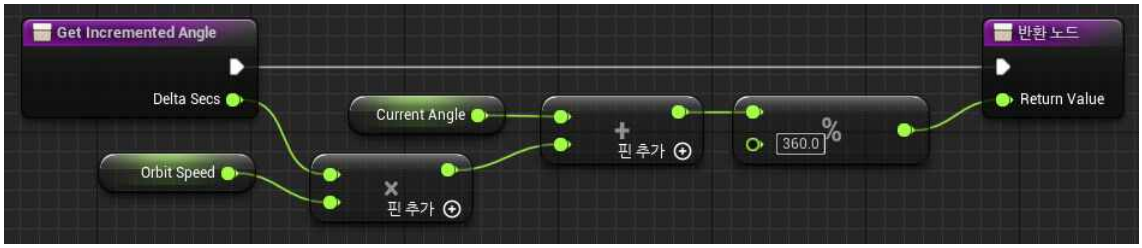
컴파일하고 저장하자.

13. 노드가 복잡해졌으므로 다시 함수로 만들어보자. 추가된 부분에서 **Set** 노드를 제외한 나머지 5개의 노드를 선택하고 우클릭하여 **함수로 접기**를 선택하자. 함수명은 **GetIncrementedAngle**로 하자.

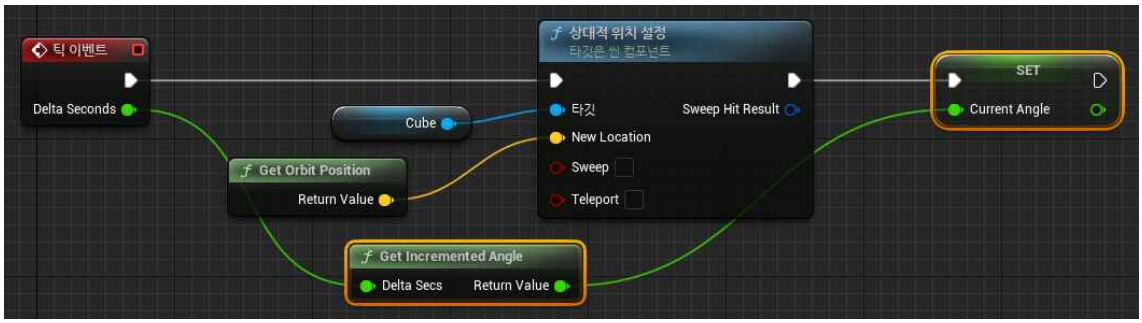
GetIncrementedAngle 함수 그래프로 이동하자.

함수로 접기할 때에 곱하기 노드로 들어오는 외부 입력선은 자동으로 함수의 입력인자로 추가된다. 함수 노드를 클릭하고 오른쪽 **디테일** 탭에서 디폴트로 **A**로 되어있는 입력 인자명을 의미를 쉽게 알 수 있도록 **DeltaSecs**로 바꾸어주자.

또한, 이 함수도 순수 함수에 해당한다. 따라서 디테일 탭에서 **퓨어** 속성에 체크하자.



14. 이제 **Tick 이벤트** 노드 그래프를 보자. 다음과 같은 모습이 되었을 것이다.



15. 레벨 에디터로 가자.

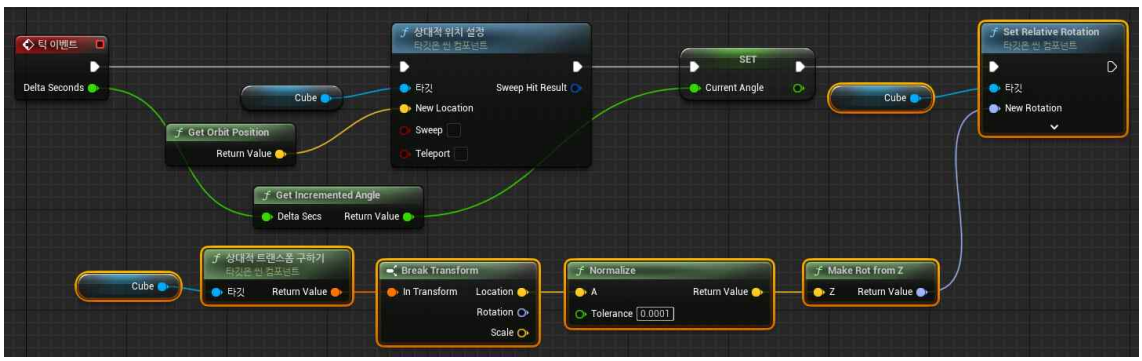
레벨 에디터에서 **BP_MyCube1**과 **BP_MyCube2**의 **디테일** 탭에서 **디폴트** 영역에 **OrbitSpeed** 디폴트 값을 수정할 수 있다. 각각 180, -90으로 수정하자. 값은 초당 회전 각도이고 음수는 반대 방향으로 진행한다.

플레이해보자. 이제 서로 반대 방향으로 스핀되는 두 큐브가 보일 것이다.

16. 한편, 두 큐브가 회전할 때 항상 전면을 보고 있을 것이다.

이제 전면이 아니라 중심을 보면서 회전하도록 수정해보자. 우리는 월드 공간이 아니라 로컬 공간에서 처리할 것이므로 간단하게 처리할 수 있다. 중심점에서 현재 큐브의 위치로 가는 벡터를 생각해보자. 큐브가 향하는 방향을 이 벡터 방향으로 지정해주면 된다. 로컬 공간에서의 중심점의 위치가 원점이므로 현재 위치를 단위벡터로 만들고 그로부터 회전 각도를 얻으면 된다.

BP_MyCube 블루프린트 에디터로 가자. 아래와 같이 **Tick 이벤트** 노드 그래프를 작성하자.

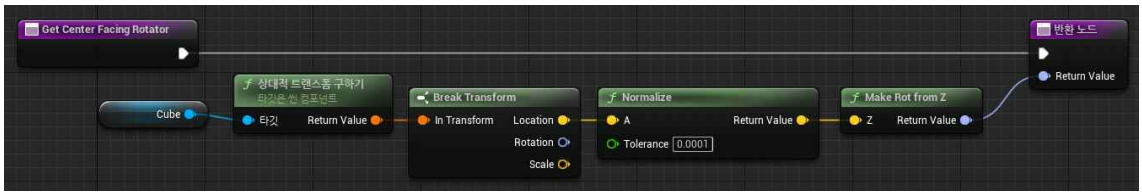


위의 그래프를 살펴보자. 먼저, 큐브의 **상대적 트랜스폼 구하기**(**GetRelativeTransform**) 함수와 **BreakTransform** 함수를 호출하여 큐브의 현재 위치를 얻는다. 그다음, **Normalize**로 정규화하여 방향벡터를 얻는다. 그다음, **MakeRotfromZ**로 Z축 회전을 가정하여 방향벡터로부터 **Rotator** 회전행렬을 얻는다. 마지막으로, **SetRelativeRotation**으로 얻어진 회전행렬을 **Cube**의 상대 회전각으로 지정한다.

실행해보자. 이제 두 큐브가 회전할 것이다.

두 큐브가 회전할 때에 항상 전면을 보고 있을 것이다.

17. 이번에도 **Cube** 레퍼런스 노드부터 **MakeRotfromZ** 노드까지의 5개의 노드를 함수로 접자.
함수명을 **GetCenterFacingRotator**라고 수정하자.
그리고, 함수 노드를 선택하고 **디테일** 탭에서 **퓨어** 속성에 체크하여 순수 함수가 되도록 하자. 함수 그래프는 다음의 모습이 될 것이다.



18. 이제 **Tick 이벤트** 노드 그래프를 보자. 다음과 같은 모습이 되었을 것이다.

19. 이제 **Tick 이벤트** 노드를 제외한 오른쪽의 모든 노드를 선택하고 다시 함수로 접자. 함수 이름을 **OrbitOnce**로 바꾸자. **OrbitOnce** 함수 그래프를 보자. 다음과 같은 모습이 되었을 것이다.

20. 이제 **Tick 이벤트** 노드 그래프를 보자. 다음과 같은 모습이 되었을 것이다.

이제 **Tick 이벤트** 그래프를 모두 완성하였다.

21. 컴파일하고 실행해보자. 이전과 동일하게 실행될 것이다.
지금까지 진행한 것처럼 노드 그래프가 너무 커지지 않게 수시로 함수화하도록 하자.
만약 함수화를 하지 않았다면 **Tick 이벤트** 그래프의 모습은 매우 커져 있을 것이다. 그리고 점점

개발 과정이 힘들어질 것이다. 따라서 항상 간결하게 함수화가 유지되도록 하자.

지금까지, 변수와 함수의 기본적인 내용에 대해서 학습하였다.

3. 형변환과 즐겨찾기

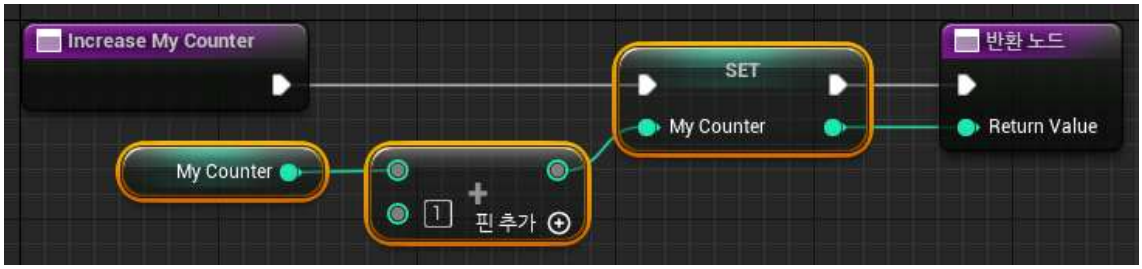
이 절에서는 한 액터의 유형을 다른 유형으로 형변환에 대해서 학습하고, 즐겨찾기 기능에 대해서 학습한다.

먼저, 스크립트 작성에서 매우 중요한 기능인 **형변환**에 대해서 알아보자.
두 액터의 겹침 콜리전이 발생할 때에 각 액터가 상대방 액터의 함수를 호출해주는 상황을 가정해보자.
우리가 만든 액터의 파생 클래스인 **BP_MyCube** 클래스에 **IncreaseMyCounter**라는 함수가 정의되어 있다고 하자. **BP_MyCube** 액터끼리 서로 충돌하여 겹침 콜리전이 발생하면 자신의 함수가 아니라 콜리전된 상대 액터의 **IncreaseMyCounter** 함수를 호출해주도록 해보자.
각 **BP_MyCube** 클래스에는 **OnComponentBeginOverlap** 이벤트 노드를 추가해서 겹침 콜리전 시에 호출되도록 하자. **OnComponentBeginOverlap** 이벤트 노드의 **OtherActor** 출력핀으로 콜리전이 발생한 상대 액터가 리턴된다. 그런데 리턴되는 유형이 **Actor**이다.
실제적으로 콜리전이 발생하는 액터의 유형은 **BP_MyCube** 클래스이지만 부모 클래스 유형인 **Actor** 클래스로 리턴되는 것이다. 그 이유는 **OnComponentBeginOverlap**가 컴포넌트 클래스에서 지원하는 범용으로 사용되는 함수이기 때문에 사용자 정의 타입을 가정하고 출력해줄 수 없기 때문이다. 따라서 **OtherActor** 출력핀을 사용해서는 **IncreaseMyCounter** 함수를 호출할 수 없다. 부모 클래스에서는 자식 클래스의 멤버를 접근할 수 없기 때문이다.
이런 경우에는 우리가 부모 클래스 유형인 **Actor** 유형으로 리턴되는 레퍼런스를 우리의 자식 클래스 유형인 **BP_MyCube** 유형의 레퍼런스로 강제로 타입을 바꾸어주어야 한다. **BP_MyCube** 유형의 레퍼런스로 바꾸어준 후에는 **IncreaseMyCounter** 함수를 호출할 수 있게 된다. 이렇게 한 유형의 레퍼런스를 다른 유형의 레퍼런스로 바꾸어주는 연산을 **형변환**(type casting)이라고 한다. 한편, 시스템에서 자동으로 형변환해주는 기능도 제공한다. 사용자가 강제로 하는 명시적(explicit) 형변환과 시스템에서 자동으로 하는 묵시적(implicit) 형변환으로 구분하기도 한다.

---

1. 이전 프로젝트 Pfuncapp에서 이어서 계속하자.

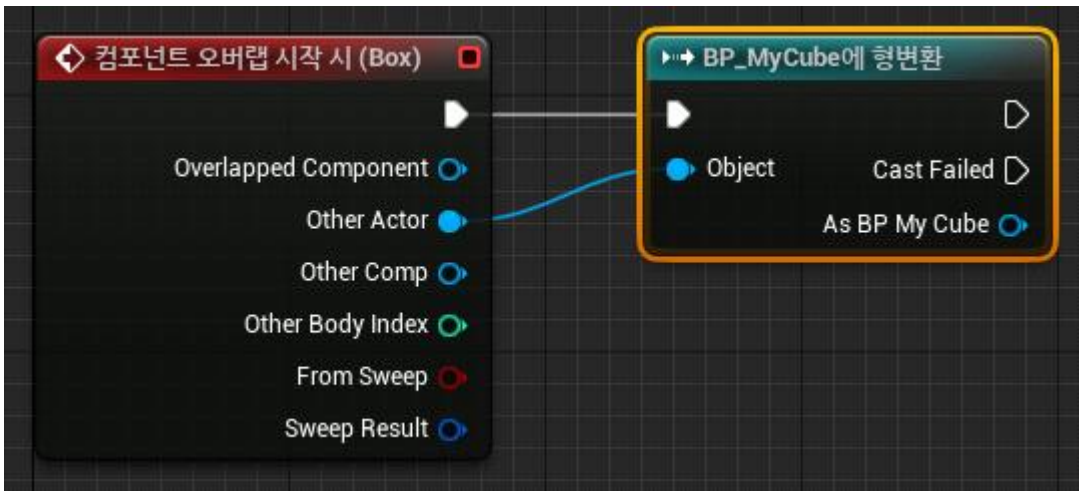
2. 이제부터, 형변환에 대해서 알아보자.
먼저, 준비 과정을 진행하자. 각 큐브는 자신에게서 겹침 이벤트가 발생한 회수를 세는 카운터 변수를 가지도록 하자.
먼저, **BP_MyCube** 블루프린트 에디터로 가서 변수를 추가해보자.
내 블루프린트 탭에서 변수 영역의 오른쪽의 + 아이콘을 클릭하여 추가하자. **MyCounter** 변수를 추가하자. 유형은 **인티저**로 하자.
컴파일하자.
그다음, 카운터를 1씩 증가시키는 함수를 추가해보자. **내 블루프린트** 탭에서 **함수** 영역의 오른쪽에 있는 + 아이콘을 클릭하자. 함수이름을 **IncreaseMyCounter**로 하자.
그리고, 증가된 후의 값을 리턴하도록 하자. 함수 그래프에서 **IncreaseMyCounter** 함수 노드를 클릭하고 출력 인자를 추가하자. 이름을 **ReturnValue**로 하고 유형을 **인티저**로 하자. 함수 그래프를 다음과 같이 작성하자.



3. **BP_MyCube** 블루프린트 에디터에서 계속하자.
컴포넌트 탭에서 **Cube**를 선택한 후에 **+추가** 버튼을 클릭하고 **Box Collision**을 선택하여 박스 콜리전 컴포넌트를 추가하자. 이름은 **Box**로 그대로 두자. **Box**를 선택하고 **트랜스폼**의 **스케일**을 (2,2,2)로 하자. 이제, 스태택 메시 컴포넌트인 **Cube** 아래에 **Box** 박스 콜리전 컴포넌트가 추가되었다.
그다음, **Box**를 우클릭하고 **이벤트 추가** » **OnComponentBeginOverlap 추가**를 선택하자. **컴포넌트 오버랩 시작 시**(**OnComponentBeginOverlap**) **(Box)** 이벤트 노드가 추가될 것이다.

4. 이제, 우리는 콜리전이 발생하면 자신의 함수가 아니라 콜리전된 상대 액터의 **IncreaseMyCounter** 함수를 호출해주자. **OnComponentBeginOverlap (Box)** 이벤트 노드의 **OtherActor** 출력핀으로 콜리전이 발생한 상대 액터가 리턴된다.
한편 리턴되는 상대 액터가 **Actor** 유형의 레퍼런스이다. 우리는 **OtherActor** 출력핀의 유형을 자식 클래스 유형인 **BP_MyCube**로 형변환을 하자. 액션선택 창에서 검색하여 **BP_MyCube에 형변환** 노드를 배치하자. 매우 많은 형변환 노드가 있게 때문에 형변환 문자열로 검색하기보다는 원하는 유형명으로 검색하는 것이 좋다.

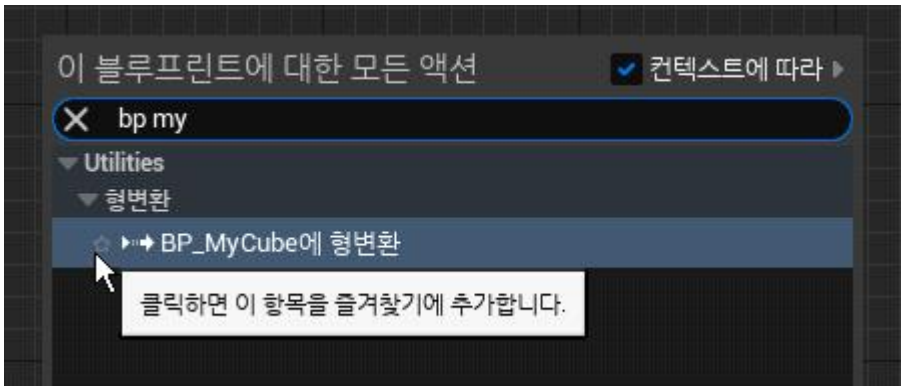
5. 배치된 형변환 노드에서는 출력핀으로 **BP_MyCube** 레퍼런스를 출력해준다. 만약 콜리전이 발생한 액터가 **BP_MyCube** 액터가 아니라 다른 액터라면 형변환 노드는 형변환에 실패할 것이다. 이 경우에는 출력 실행핀으로는 펄스가 흐르지 않고 **Cast Failed** 출력 실행핀으로 펄스가 흐르게 된다. 따라서 우리는 콜리전이 발생 가능한 액터가 **BP_MyCube** 유형의 액터라는 확신이 없더라도 자유롭게 형변환 노드를 사용해도 된다.



지금까지 **형변환**에 대해서 알아보았다.

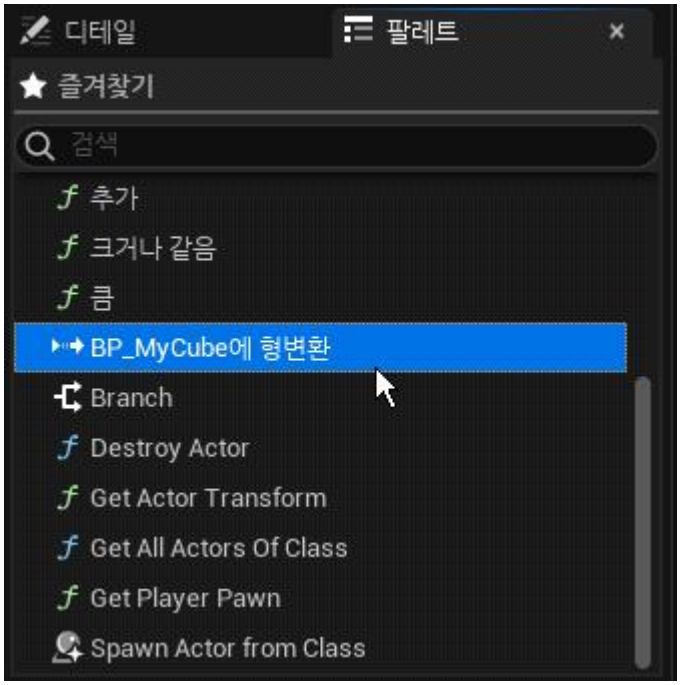
6. 지금부터, 유용한 팁인 즐겨찾기 기능에 대해서 알아보자.
에디터에는 자주 사용되는 노드를 미리 등록해두고 나중에 쉽게 찾아볼 수 있게 하는 즐겨찾기 기능이 있다. 이러한 즐겨찾기 탭을 언리얼에서는 **팔레트**라고 한다. 메뉴바에서 **창** » **팔레트**를 선택하면 **디테일** 탭 옆에 **팔레트** 탭이 생긴다.

7. 위의 단계에서 알아본 **BP_MyCube에 형변환**과 같은 노드는 **BP_MyCube**와 관련하여 자주 사용될 것이다. 이 노드를 즐겨찾기에 추가해보자. 액션선택 창에서 검색하여 **BP_MyCube에 형변환** 노드가 나열될 때에 클릭하지 말고 마우스를 이동시켜보면 노드명 앞에 희미한 별 모양이 보인다. 기본적으로 테두리만 그려진 별 모양이다. 이곳을 클릭하면 채워진 별 모양으로 바뀌며 즐겨찾기에 추가된다.



클릭하여 즐겨찾기에 추가해보자.

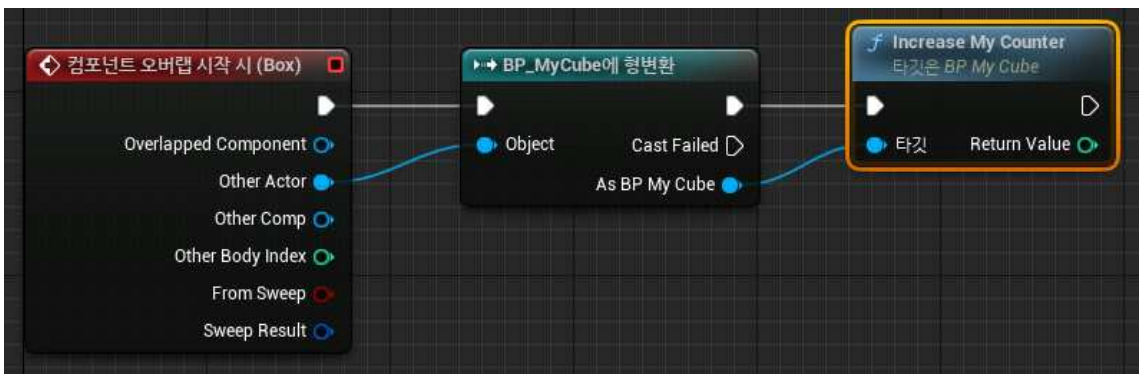
8. 이제 팔레트를 확인해보자. **BP_MyCube에 형변환** 노드가 나열되어 있을 것이다. 이제부터는 액션선택 창을 사용할 필요없이 팔레트 창에서 해당 노드를 드래그해서 배치하면 된다.



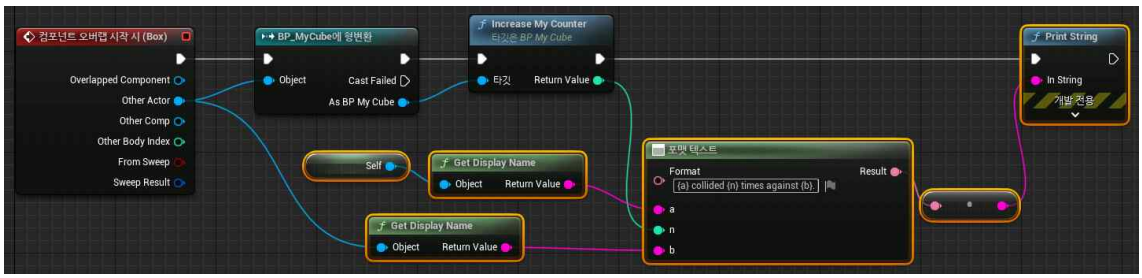
나중에 즐겨찾기에서 제외하고싶은 경우에는 팔레트에서 해당 노드를 우클릭하고 팝업메뉴에서 **즐겨찾기에서 제거**를 선택하면 된다. 또다른 방법으로, 액션선택 창에서 해당 노드를 찾아서 채워진 별 모양을 다시 클릭해서 비워진 별 모양으로 만들어도 즐겨찾기에서 제거된다.
지금까지 즐겨찾기 기능에 대해서 알아보았다.

9. 이제부터, **BP_MyCube** 블루프린트 에디터에서 계속해서 노드 그래프를 작성해보자.
BP_MyCube에 형변환 노드에서 **As BP_MyCube** 출력핀을 당기고 액션선택 창에서 검색해보자. 우리의 커스텀 함수들이 나열될 것이다.

10. 액션선택 창에서 **IncreaseMyCounter** 함수 노드를 검색하여 배치하자.



11. 실행해도 출력이 없으므로 충돌을 확인할 수가 없다. 따라서 출력되도록 다음과 같이 완성하자.

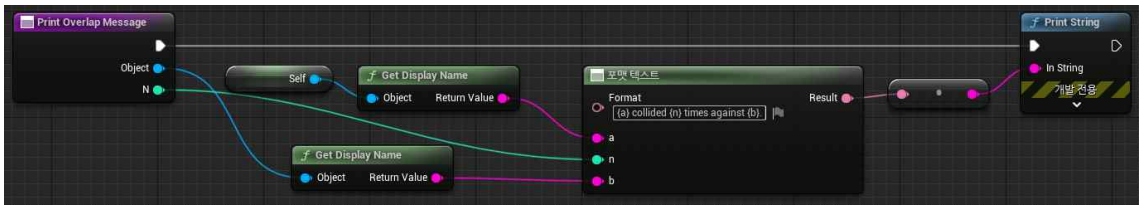


그래프를 살펴보자. 먼저, 액션선택 창에서 검색하여 **FormatText**를 검색하여 **포맷 텍스트** 노드를 배치하자. 그다음, **Format** 입력핀에 '`{a} collided {n} times against {b}.`' 식의 문자열을 입력하자. 입력한 후에는 **a**, **n**, **b**의 세 입력핀이 생성된다. 여기에 입력값을 넣어주면 된다. **a**에는 자신의 인스턴스 이름을 넣어주고 **b**에는 충돌한 상대 액터 인스턴스의 이름을 넣어주자. **n**에는 충돌 카운터 변수값을 넣어주자.

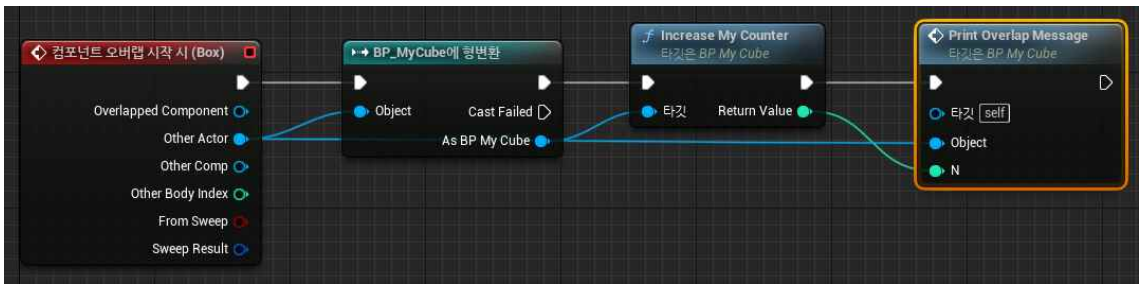
GetDisplayName 노드를 사용하면 액터 인스턴스 이름을 얻을 수 있다. 상대 액터의 레퍼런스는 **OnComponentBeginOverlap (Box)** 이벤트 노드의 **OtherActor** 출력핀으로 얻을 수 있다.

자신의 레퍼런스는 액션선택 창에서 **self**를 검색해서 **셀프 레퍼런스 가져오기**를 선택해서 **Self** 노드를 배치할 수 있다. **PrintString** 노드를 배치하고 **포맷 텍스트** 노드의 결과를 **PrintString** 노드로 연결하자.

12. 이전 단계에서 추가한 부분을 함수로 접자. 함수명은 **PrintOverlapMessage**로 지정하자. 다음과 같은 함수 그래프가 만들어질 것이다.

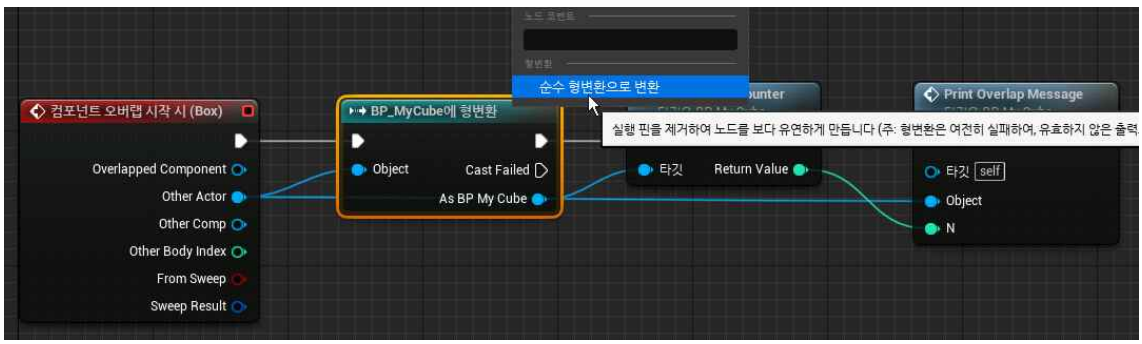


13. **OnComponentBeginOverlap (Box)** 이벤트 노드 그래프는 다음과 같은 모습이 될 것이다.

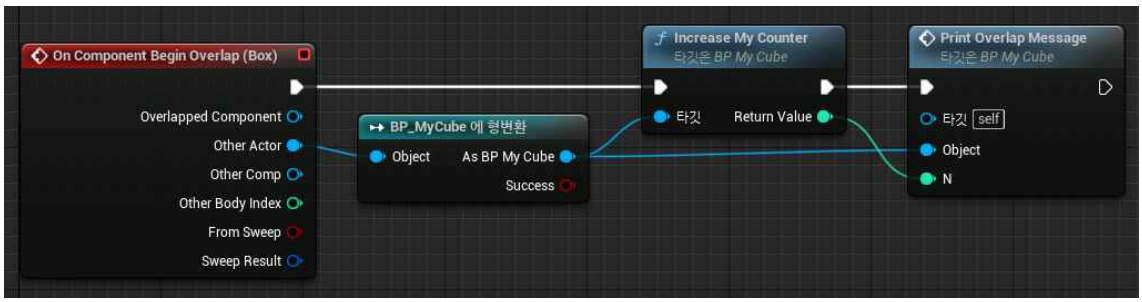


14. 플레이해보자. 다음과 같이 실행될 것이다.

15. 한편, 형변환 노드도 멤버 변수를 수정하지 않으므로 순수 형변환으로 바꿀 수 있다. 형변환 노드를 우클릭하고 팝업메뉴에서 **순수 형변환으로 변환**을 선택하자.



16. 변환 노드의 실행핀이 없어질 것이다. 기존 실행핀은 다음 노드의 실행핀으로 바로 연결된다. 그리고, **PrintOverlapMessage** 노드의 **Object** 입력핀에는 최상위 클래스인 **Object** 유형을 가정하므로 형변환 전의 객체를 연결하거나 또는 형변환 후의 객체를 연결하거나 모두 동일하게 동작할 것이다. 따라서 좀더 간단한 노드 네트워크 모양을 위해서 형변환 후의 출력을 연결하도록 수정하자.

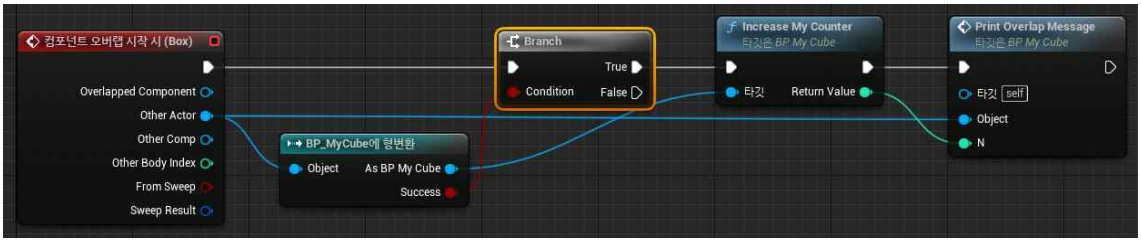


17. 한편, 순수 형변환 노드로 바꾸었을 때는 실행핀이 없어지므로 형변환이 실패하는 경우에 대한 고려가 따로 있어야 한다.

순수 형변환으로 바꾸기 전의 실행핀이 있는 경우에는 형변환이 성공할 때에만 형변환 노드의 출력 실행핀에 펄스가 흐르므로 문제가 없었다. 또한 **CastFailed** 출력 실행핀을 사용하여 형변환이 실패하는 경우에 대한 조치를 할 수 있었다.

순수 형변환의 경우에는 출력 실행핀을 대신하여 **Success** 출력핀이 제공된다. 조건에 따라서 분기하

는 **Branch** 노드를 형변환 노드 뒤에 배치하고 형변환 노드의 **Success** 출력핀을 **Branch** 노드의 **Condition** 입력핀에 연결하자. 아래와 같이 그래프를 완성하자.



사실 순수 형변환으로 항상 바꿀 필요는 없다. 노드 네트워크가 더 간결하고 분명하게 되도록 활용하면 된다.
플레이해보자. 기존과 동일하게 실행될 것이다.

지금까지, 형변환과 즐겨찾기에 대해서 학습하였다.

□