15_ 배열과 라이브러리

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서는 배열과 라이브러리에 대해서 학습한다.

블루프린트에서 지원하는 변수 유형에는 **부울**, **인티저**, **플로트**, **벡터**, 구**조체**, **오브젝트 타입** 등이 있다. 이들을 **단일** 유형이라고 한다. 블루프린트에서는 **단일** 유형 외에도 **배열** 유형, **세트** 유형과 **맵** 유형을 지원한다. **배열**, **세트**, **맵**의 세 유형은 한 **단일** 유형의 여러 데이터 요소를 가지는 컨테이너에 해당한다. **세트** 유형과 **맵** 유형은 STL 컨테이너에서의 세트와 맵의 개념과 동일하다.
배열은 순차적으로 나열된 동일 유형의 데이터를 저장하는 데이터 구조이다. 배열 내부에는 다수개의 데이터를 넣을 수 있다. 이들 각각의 데이터를 요소(element)라고 한다. 배열 내부의 각 요소는 순서대로 배치되어 있고 순서에 따라 번호가 매겨져 있다. 그 번호를 인덱스(index)라고 한다. 배열의 인덱스는 다른 대부분의 언어에서와 같이 0부터 시작한다. 즉 0번 요소가 첫 번째 요소이다.

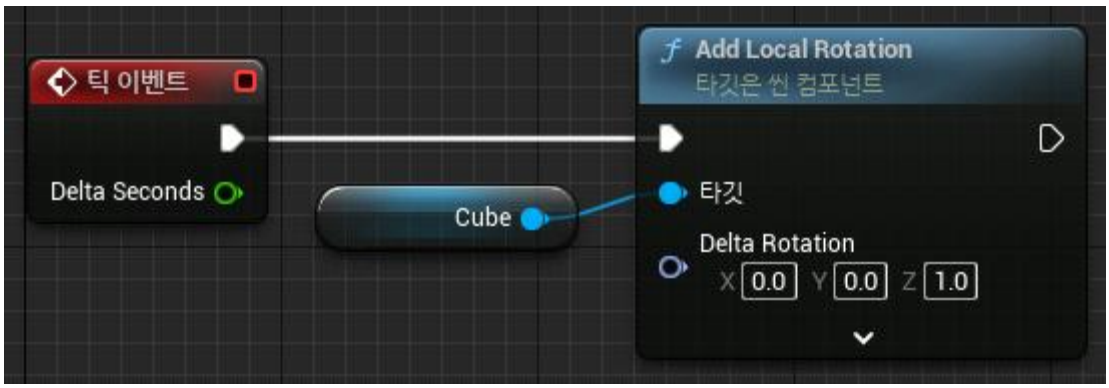
2. 배열 사용하기

이 절에서는 배열을 만들고 활용하는 방법에 대해서 학습한다.
이제부터 배열을 만들고 사용하는 방법을 배워보자.

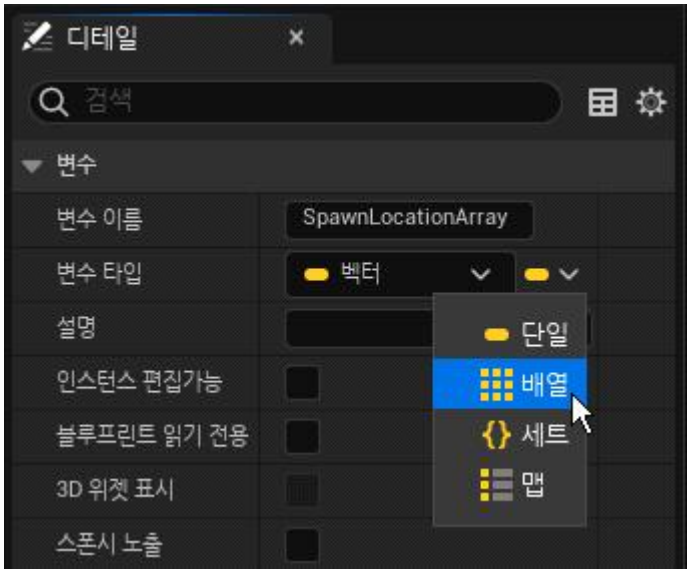
---

1. 새 프로젝트 Parray를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Parray으로 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.

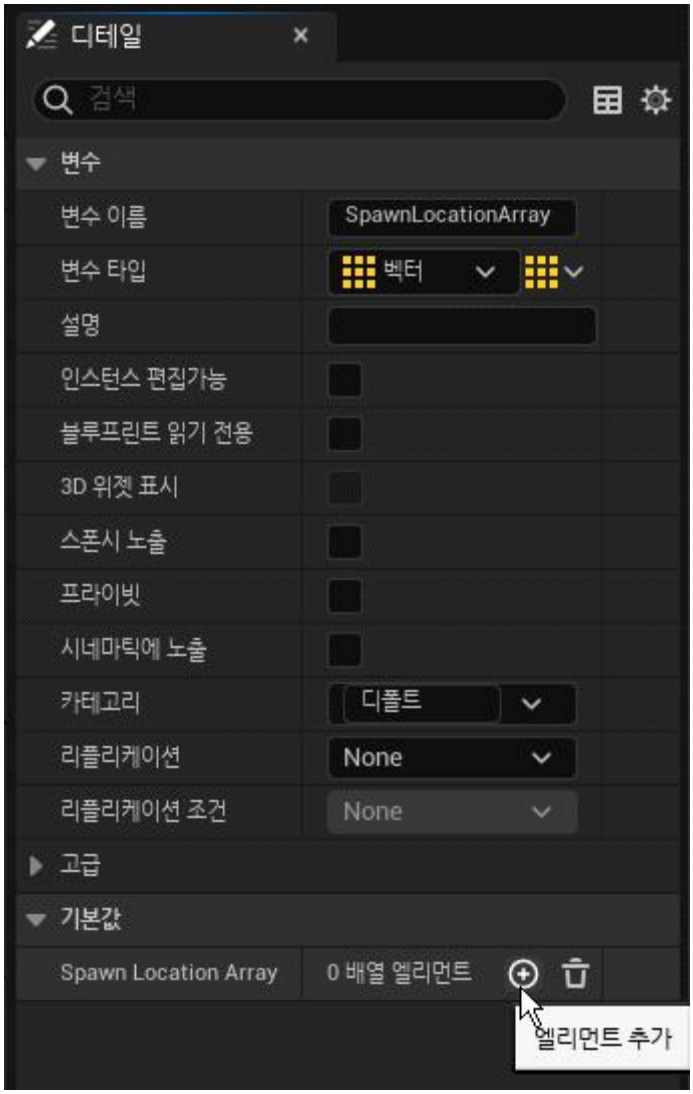
2. 이전 예제에서 만들었던 스핀되는 큐브인 **BP_MyCube**를 동일한 방법으로 만들어보자.
먼저, **콘텐츠 브라우저**에서 툴바에서 **+추가**를 클릭하고, 드롭다운 메뉴에서 **블루프린트 클래스**를 선택하고, **부모 클래스 선택** 창에서 **Actor**를 부모 클래스로 선택하자. 이름은 **BP_MyCube**으로 수정하자.
그다음, **BP_MyCube**을 더블클릭하여 **블루프린트 에디터**를 열자. **컴포넌트** 탭에서 **+추가**를 클릭하고 **큐브**를 선택하자. 큐브 스태틱 메시 컴포넌트가 추가될 것이다. 디폴트로 **Cube**라는 이름으로 그대로 두자.
그다음, 블루프린트 에디터에서 이벤트 그래프 탭을 선택하고 격자판의 빈 곳에서 **AddLocalRotation (Cube)** 노드를 검색해서 배치하자. 그리고 노드의 입력핀 **DeltaRotation**의 **Z**값에 1을 입력하자.
그다음, **Tick 이벤트** 노드의 출력 실행핀을 **AddLocalRotation** 노드의 입력 실행핀에 연결하자. 완성된 모습은 아래와 같다.



3. 이제, 배열에 대해서 학습을 시작해보자.
먼저 배열 변수를 준비해야 한다. 레벨 에디터의 툴바에서 **블루프린트** » **레벨 블루프린트 열기**를 선택하여 **레벨 블루프린트**를 열자.
내 블루프린트 탭에서 변수를 추가하자. 변수명을 **SpawnLocationArray**라고 하자. 유형은 **벡터(Vector)**로 하자. 그다음, **디테일** 탭에서 **변수 유형**의 리스트박스 오른쪽의 아이콘을 클릭하자. 디폴트인 **단일** 이외에 **배열**, **세트**, **맵** 중의 하나를 선택할 수 있다. 우리는 **배열**을 선택하자.

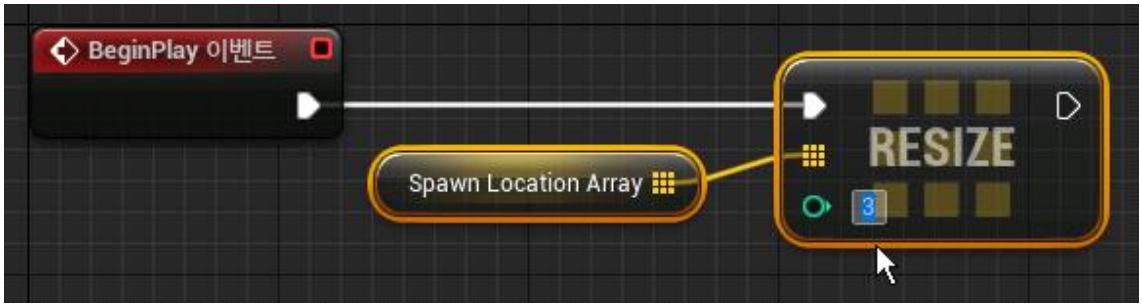


4. 이제, 아이콘 모양이 배열을 나타내는 모양으로 바뀌었다.
컴파일하자. 컴파일한 후에는 디테일 탭의 아래에 **기본값** 영역이 나타난다.



배열 변수는 디폴트로 비어있는 배열로 생성된다. **기본값** 영역 아래의 + 아이콘을 클릭하면 초기 배열의 크기와 각 배열 요소의 디폴트 값을 지정할 수 있다.
디테일 탭의 **기본값** 영역에서 배열 변수의 디폴트 설정을 할 수도 있지만, 여기서는 디폴트로 그대로 두도록 하자. 우리는 이벤트 그래프에서 스크립트로 작성해보자.

5. 이제, 레벨 블루프린트 에디터의 이벤트 그래프 탭에서 노드 네트워크를 작성해보자.
스핀되는 큐브인 **BP_MyCube** 액터를 세 지점 (300,-200,100), (300,0,100), (300,200,100)에 스폰해보자.
먼저, 배열 변수 **SpawnLocationArray**의 **Get** 노드를 배치하자. 그다음, **Get** 노드의 출력핀을 드래그하고 **Resize** 노드를 배치하자. **Resize** 노드는 배열의 크기를 수정해준다. 초기 배열의 크기가 0인데 이를 3으로 변경하도록 하자.
그리고, **Resize** 노드의 실행핀을 **BeginPlay** 이벤트 노드의 실행핀에 연결하자.



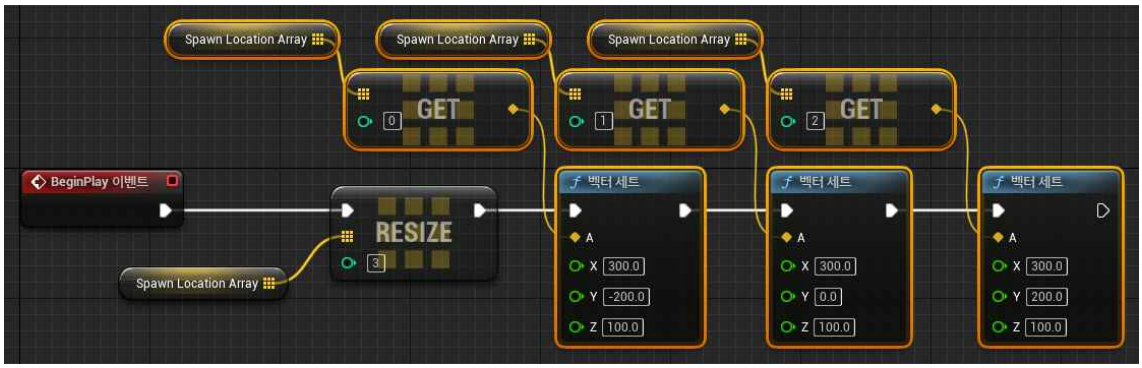
6. 이제, 배열 요소의 **Get** 노드를 배치하자.

배열 변수의 **Get** 노드와 배열 요소의 **Get** 노드는 전혀 다른 노드이니 정확히 구분하자. 배열 변수의 **Get** 노드는 이미 배치해둔 **SpawnLocationArray**의 **Get** 노드와 같이 배열 변수의 레퍼런스 노드이다. 배열 요소의 **Get** 노드는 배열 변수의 멤버 연산자이고 배열 변수 내에서 주어진 인덱스값에 해당하는 요소를 찾아서 그 레퍼런스를 리턴하는 노드이다.

배열 요소의 **Get** 노드를 배치하는 방법은 **SpawnLocationArray**의 **Get** 노드의 출력핀을 당기고 **Get**을 검색하여 배치하면 된다.

배열 요소의 **Get** 노드에는 **Get (사본)**과 **Get (참조)**의 두 버전이 있다. **Get (참조)**는 배열의 해당 요소의 레퍼런스를 바로 리턴한다. **Get (사본)**은 해당 요소의 복사본을 만들고 이를 리턴한다. 이것은 해당 요소의 원본 값이 변경되는 것을 방지하기 위한 목적으로 사용된다. 대부분의 경우 **Get (참조)**를 선택하면 된다. 우리도 **Get (참조)**를 선택하자.

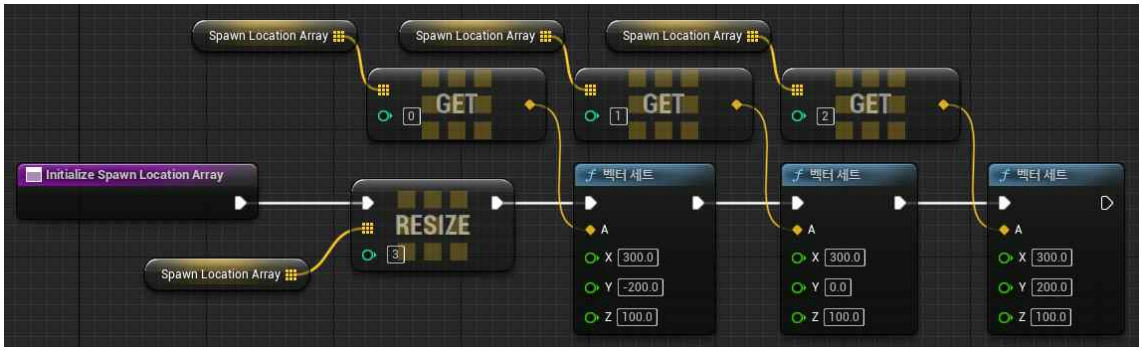
그다음, **벡터 세트(VectorSet)** 노드를 배치하자. 이 노드는 벡터에 직접 값을 지정하는 노드이다. 배열 요소 0,1,2에 대해서 각각 (300,-200,100), (300,0,100), (300,200,100)의 좌표를 지정하자. 아래의 그래프와 같다.



참고로, 위의 그래프에서 **SpawnLocationArray**의 변수 **Get** 노드가 여러분 반복해서 등장한다. 사실상 하나만 배치하고 필요한 모든 곳에 연결하여 사용해도 된다. 그러나 연결선이 너무 복잡해지는 것을 방지하기 위해서 **Get** 노드를 중복해서 배치하였다. 어느 경우든 성능 등에 큰 차이가 없으니 선호하는 방식으로 사용하면 된다.

7. 이제, **BeginPlay 이벤트** 노드 뒤의 모든 노드를 함수로 접자. 함수 이름은 **InitializeSpawnLocationArray**로 하자.

8. **InitializeSpawnLocationArray** 함수 그래프는 다음과 같이 생성되었을 것이다.

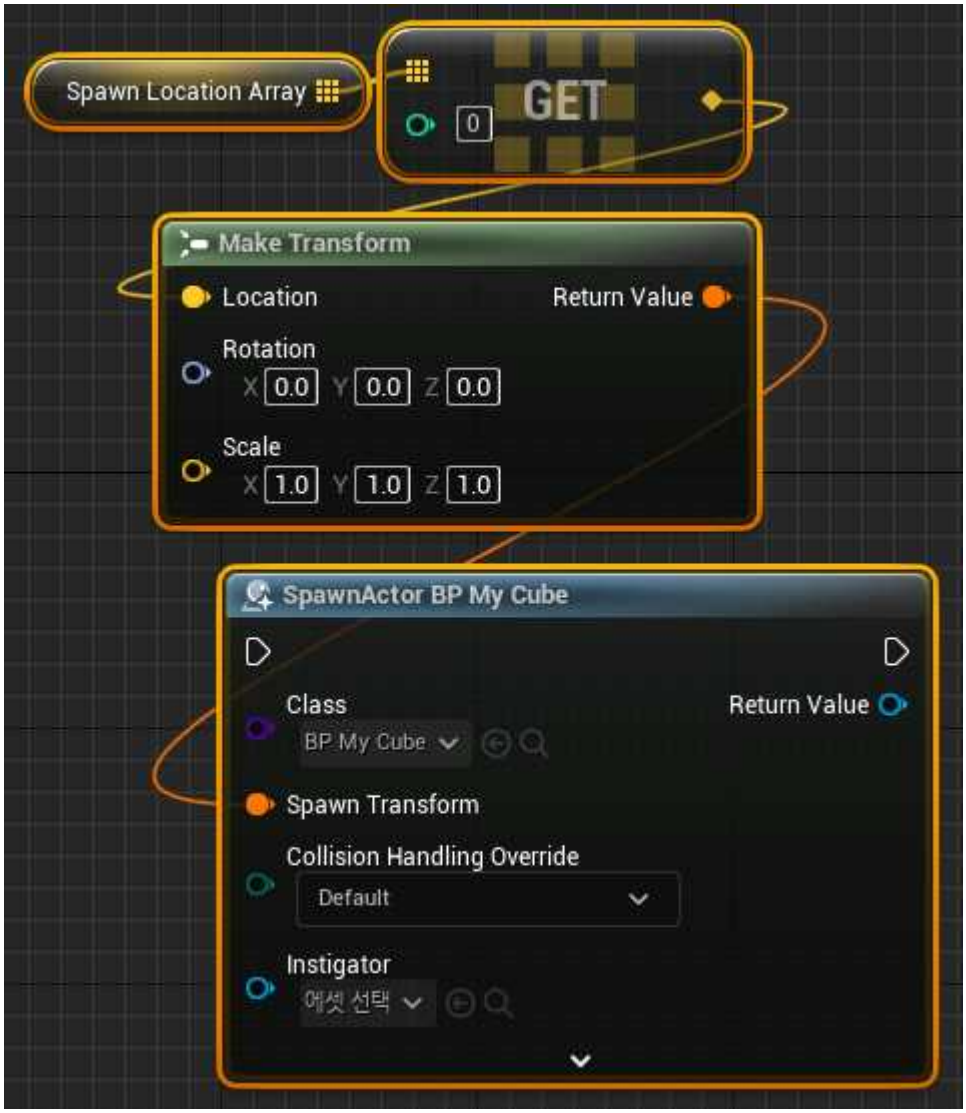


9. 이제, 배열의 인덱스 0의 위치에 **BP_MyCube**를 스폰하자.

먼저, **SpawnLocationArray** 배열 변수의 **Get** 노드를 배치하고 그 노드로부터 다시 배열 요소의 **Get** 노드를 배치하자. 그다음, 배열 요소의 **Get** 노드의 출력핀을 드래그하여 **MakeTransform** 노드를 배치하자.

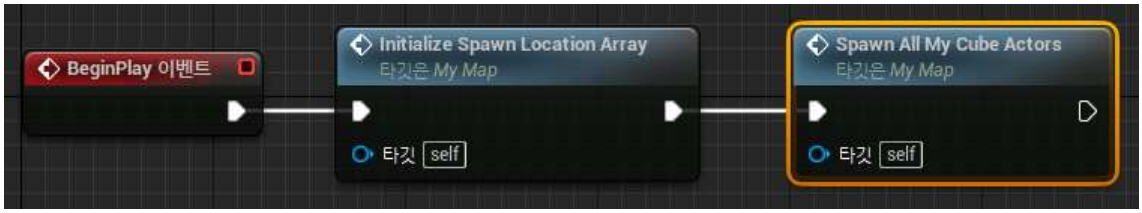
그다음, **SpawnActor fromClass** 노드 검색하여 배치하고 노드의 **Class** 입력핀에 **BP_MyCube**를 명시하자.

그다음, **MakeTransform** 노드의 출력값을 **SpawnActor** 노드의 **SpawnTransform** 입력핀에 연결하자.



10. 이 과정을 인덱스 1,2에 대해서도 동일한 방법으로 반복하자. 다음과 같은 그래프의 모습이 될 것이다.

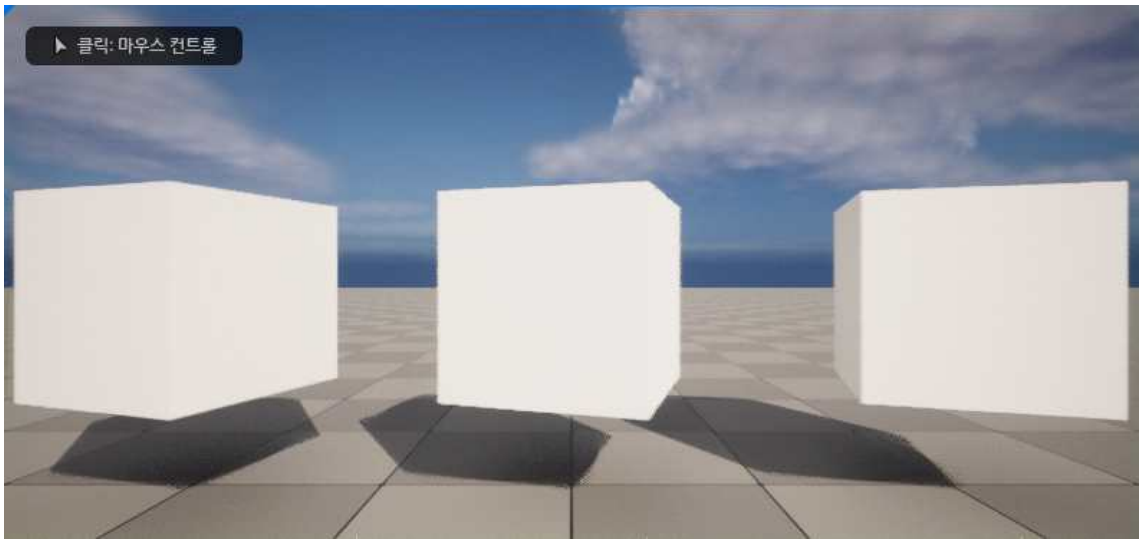
11. 이제, **InitializeSpawnLocationArray** 노드 뒤의 모든 노드를 함수로 접자. 함수 이름은 **SpawnAllMyCubeActors**로 하자.



컴파일하자.

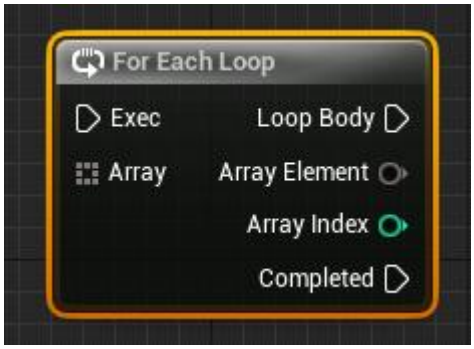
12. **SpawnAllMyCubeActors** 함수 그래프는 다음과 같이 생성되었을 것이다.

13. 플레이해보자. 세 큐브가 스폰되어 스핀하는 것을 확인할 수 있을 것이다.



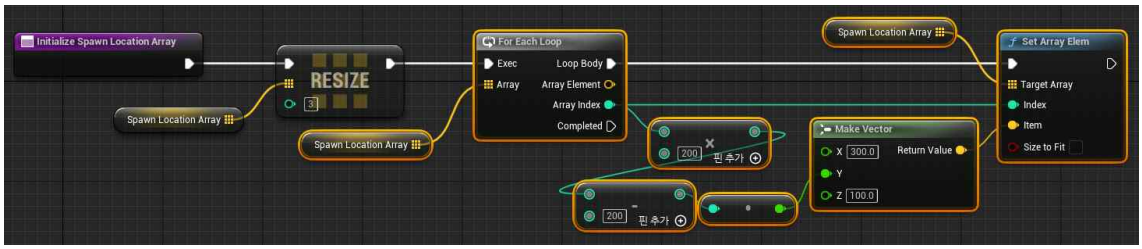
14. 우리는 위에서 각 요소에 대한 처리를 하나씩 나열하여 반복하였다. 이러한 방식은 배열이 크기가 커지면 적절하지 않다. 그보다는 인덱스 변수를 사용하여 반복문으로 처리하는 것이 적당하다.

ForEachLoop 노드에 대해서 알아보자.

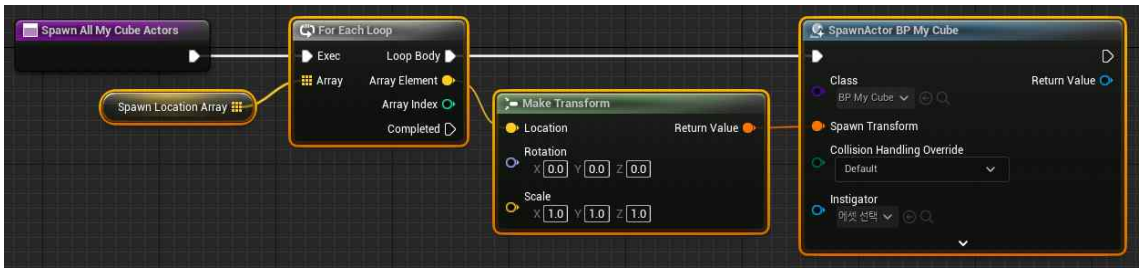


ForEachLoop 노드를 사용하면 각 배열의 요소에 대해서 반복할 수 있다. **ForEachLoop** 노드의 **Array** 입력핀에 지정되는 배열에 대하여 그 크기만큼 각 요소에 대해서 반복한다. 반복 시마다 **LoopBody**에 출력 실행핀에 연결된 그래프를 실행해준다. 그 그래프에서 사용할 수 있도록 각 요소의 값과 인덱스를 **ArrayElement**와 **ArrayIndex**의 출력핀으로 제공한다. 모든 반복이 종료된 후에는 **Completed** 실행핀으로 펄스를 보낸다.

15. 이제부터 **ForEachLoop** 노드를 사용하여 이전의 함수를 하나씩 수정해보자.
먼저, 레벨 블루프린트 에디터의 **InitializeSpawnLocationArray** 함수 그래프로 이동하자.
세 지점 (300,-200,100), (300,0,100), (300,200,100)의 좌표가 Y값만 바뀌고 있으므로 Y값만 인덱스값으로부터 계산해주면 된다. 다음과 같이 함수 그래프를 수정하자.
ForEachLoop 노드를 검색하여 배치하자. 그다음, **ForEachLoop** 노드의 **ArrayIndex** 출력핀을 사용하여 Y좌표값을 계산한다. 그다음, **MakeVector**로 위치 벡터를 만든다. 그다음, 배열의 요소값을 지정하는 **SetArrayElem** 노드를 배치하고 요소의 인덱스와 요소값을 노드의 **Index**와 **Item** 입력핀에 지정하면 된다.



16. 이제, **SpawnAllMyCubeActors** 함수 그래프도 반복문으로 처리해보자.
이전과 유사하게 **ForEachLoop** 노드를 배치하여 **MakeTransform** 노드와 **SpawnActor** 노드를 이으면 된다.



지금까지, 배열 사용하기에 대해서 학습하였다.

3. 레벨 에디터에서 배열 초기화하기

이 절에서는 레벨 에디터에서 배열을 초기화하는 방법에 대해서 학습한다.

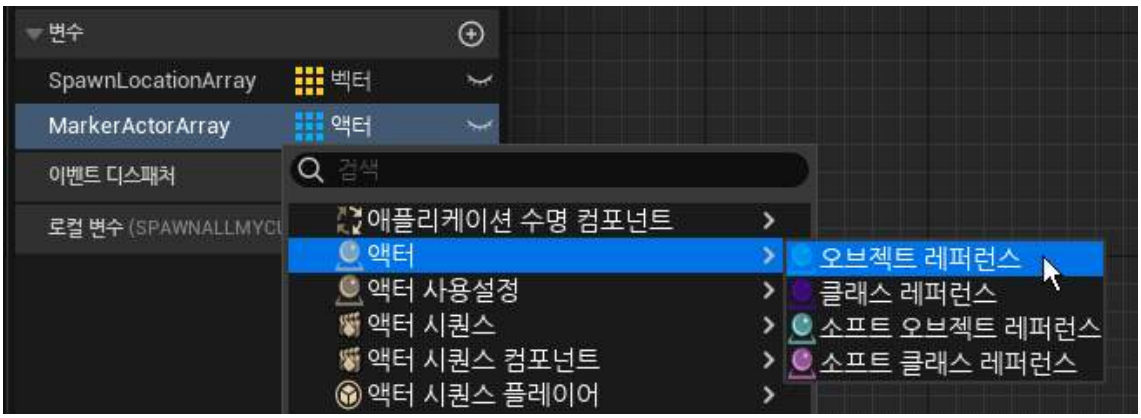
레벨 블루프린트에서는 레벨에 배치된 모든 액터를 자유롭게 접근할 수 있다. 따라서 레벨 블루프린트에서 추가된 액터 유형의 배열 변수에 대해서는 레벨에 배치된 액터를 배열의 초기값으로 사용될 기본값을 지정해줄 수 있다.
레벨 블루프린트 편집기에서 배열의 초기값을 쉽게 지정할 수 있도록 인터페이스를 제공한다. 배열의 각 요소마다 선택상자를 통해서 레벨에 배치된 액터 인스턴스를 선택할 수 있다. 배열의 각 엘리먼트의 선택상자를 클릭하면 레벨에 배치된 액터 인스턴스들이 나열된다. 원하는 인스턴스를 선택하면 기본값으로 지정된다.
또한 스포이드 모양의 선택 도구를 사용하여 배열 엘리먼트에 기본값을 지정할 수도 있다. 배열 엘리먼트의 오른쪽에 스포이드 모양의 아이콘을 클릭한 후에 레벨 에디터의 뷰포트로 마우스를 이동하면 커서 모양이 스포이트 모양으로 바뀌면서 배치된 액터를 선택할 수 있다. 툴팁을 확인하면서 원하는 액터를 선택하면 된다. 선택된 액터는 배열 엘리먼트의 기본값으로 지정된다.
한편, 클래스 블루프린트 에디터에서는 특정 레벨에 종속되어 있지 않으므로 이러한 초기화가 불가능하다.

이제부터 레벨 블루프린트에서 배열의 기본값을 지정하는 방법을 알아보자.

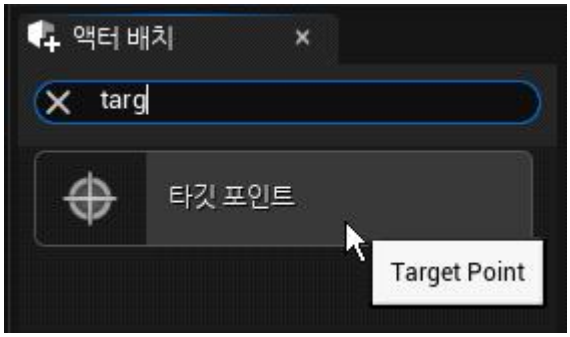
---

1. 이전 프로젝트 Parray에서 계속하자.

2. 먼저 새로운 배열 변수를 준비하자.
레벨 에디터의 툴바에서 **블루프린트** » **레벨 블루프린트 열기**를 선택하여 **레벨 블루프린트**를 열자. **내 블루프린트** 탭에서 변수를 추가하자. 변수명을 **MarkerActorArray**라고 하자. 유형은 **오브젝트 타입**의 **액터**(**Actor**)의 **오브젝트 레퍼런스**로 설정하자. 배열 아이콘이 아니라면, **디테일** 탭에서 **변수 유형**의 리스트박스 오른쪽의 **배열**을 선택하자. 컴파일하고 저장하자.



3. 이제, 레벨 에디터에서 **액터 배치** 탭을 열고 **TargetPoint**를 검색하여 찾자.

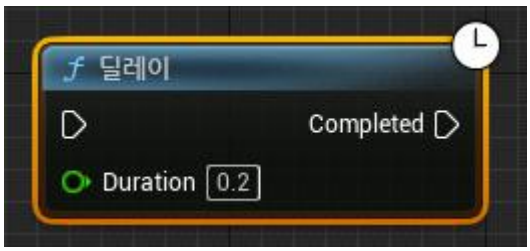


찾은 **타깃 포인트** 액터를 드래그하여 레벨에 배치하자. 3개를 배치하고 각각의 이름을 **TargetPoint1**, **TargetPoint2**, **TargetPoint3**으로 하자. 이들의 위치를 (300,-200,200), (300,0,200), (300,200,200)에 배치하자.

4. 이제, 레벨 블루프린트 에디터로 가자.
MarkerActorArray 변수를 선택하고 **디테일** 탭의 기본값 영역에서 **MarkerActorArray**의 + 버튼을 클릭하여 3개의 **배열 엘리먼트**를 추가하자.
없음으로 되어 있는 선택상자를 클릭하고 목록에서 각각 **TargetPoint1**, **TargetPoint2**, **TargetPoint3**를 선택하여 지정하자. 컴파일하자.

또한, 선택상자 대신에 그 오른쪽에 있는 스포이드 선택 도구를 사용해도 액터 인스턴스를 선택할 수 있다. 스포이드 아이콘을 클릭하고 뷰포트에서 해당 액터를 클릭하여 선택해보자.

5. 이제, 스폰할 위치를 나타내는 **타깃 포인트** 액터가 준비되었다. 스폰하기 전에 잠시 시간 간격을 두도록 해보자.
잠시 대기하는 기능은 **Delay** 노드를 사용하면 된다.



Delay 노드는 실행핀의 흐름을 일정 시간동안 막아서 펄스가 흐르지 못하도록 중지해주는 기능을 한다. 자주 사용하는 노드이다. 검색할 필요 없이 키보드 **D** 키를 누르고 좌클릭하면 바로 배치된다.
한편, 함수 그래프에서는 **Delay** 노드의 사용이 불가능하다. 이벤트 노드 그래프에서만 사용이 가능하다.

<참고> 노드 배치에 대한 단축키는 아래의 치트키 문서를 참조하자.

https://docs.unrealengine.com/blueprint-editor-cheat-sheet-in-unreal-engine/

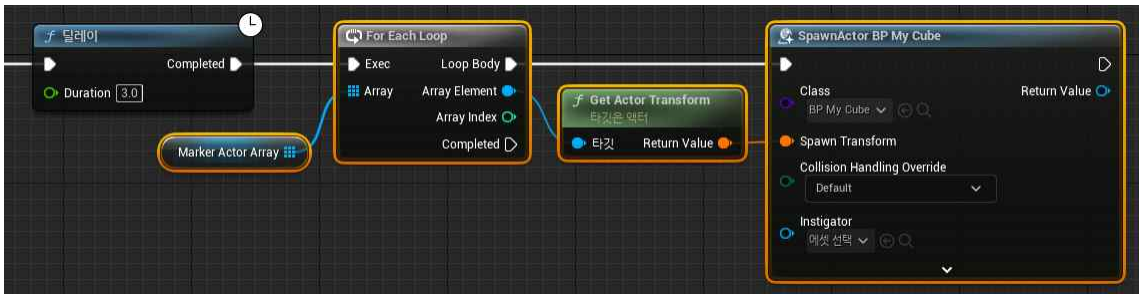
6. 이제, 레벨 블루프린트 에디터로 가자.

Delay 노드를 배치하자. **Duration**을 3으로 하자. 이를 **BeginPlay** 이벤트 노드의 마지막에 연결하자.

7. 이제, 각 **타깃 포인트** 액터에 **BP_MyCube**를 스폰하자.

BeginPlay 이벤트 그래프의 마지막에 있는 **Delay** 노드에 이어서 작성하자.

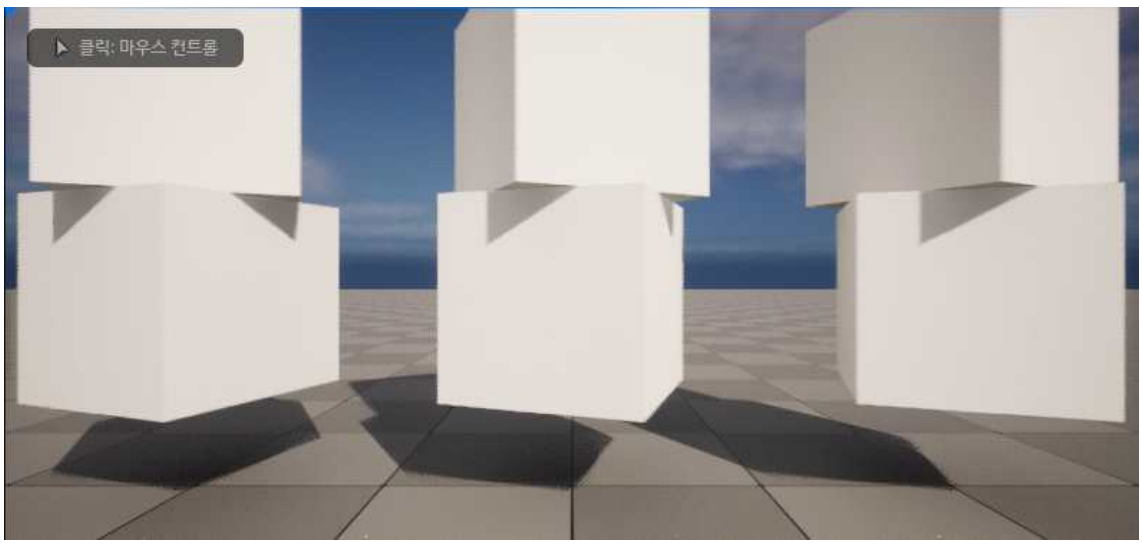
먼저, **ForEachLoop** 노드를 배치하여 **MarkerActorArray** 배열의 각 요소값인 **타깃 포인트** 액터를 가져오자. 그다음, **GetActorTransform** 노드를 사용하여 **타깃 포인트** 액터의 트랜스폼을 얻자. 그다음, **SpawnActor** 노드를 배치하고 이전의 트랜스폼을 **SpawnTransform** 입력핀으로 넣어주자.



8. **Delay** 노드 뒤의 노드들을 함수로 접자. 함수명을 **SpawnAtMarkerLocations**라고 하자.

한편, **Delay** 노드를 포함하여 함수로 접을 수는 없다. 왜냐하면 **Delay** 노드는 함수 그래프에 포함될 수 없기 때문이다.

9. 실행해보자. 3초 후에 기존 큐브의 위에 새로운 큐브가 스폰될 것이다.



지금까지, 이 절에서는 레벨 에디터에서 배열을 초기화하는 방법에 대해서 학습하였다.

4. 함수 라이브러리 만들고 사용하기

이 절에서는 **함수 라이브러리**를 만들고 활용하는 방법에 대해서 학습한다.

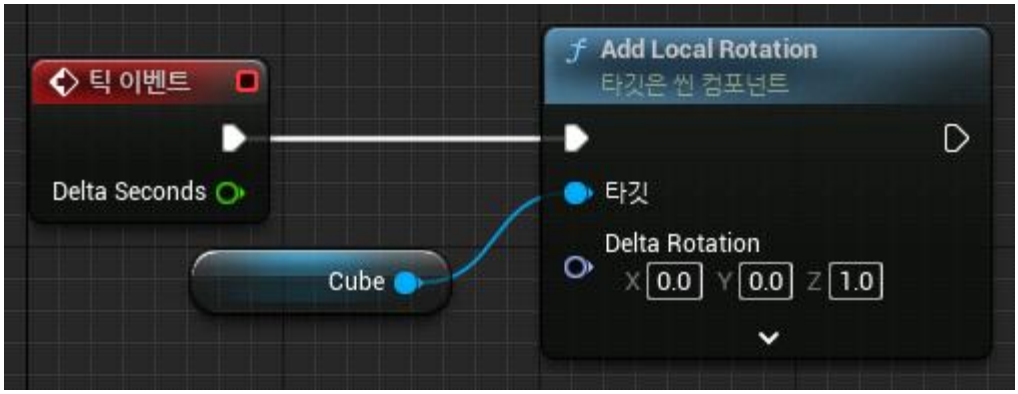
각 클래스 내부에서 함수를 만들고 사용할 수 있다. 그러나 클래스 내부에서 만든 함수는 그 클래스나 파생 클래스에서만 사용할 수 있다. 한편 함수를 라이브러리로 만들어두면 모든 클래스에서 사용할 수 있다.
함수 라이브러리는 정적 함수들의 모음이다. 정적 함수는 특정 클래스 내부에 존재하는 멤버 함수가 아니다. 따라서 특정 클래스에 의존되어 있지 않으며 따라서 어떠한 클래스에서도 자유롭게 사용할 수 있다. 유용하게 활용할 수 있는 유틸리티 함수를 **함수 라이브러리**에 구현해두면 편리하게 사용할 수 있다.

이제부터 함수 라이브러리에 대해서 학습해보자.

---

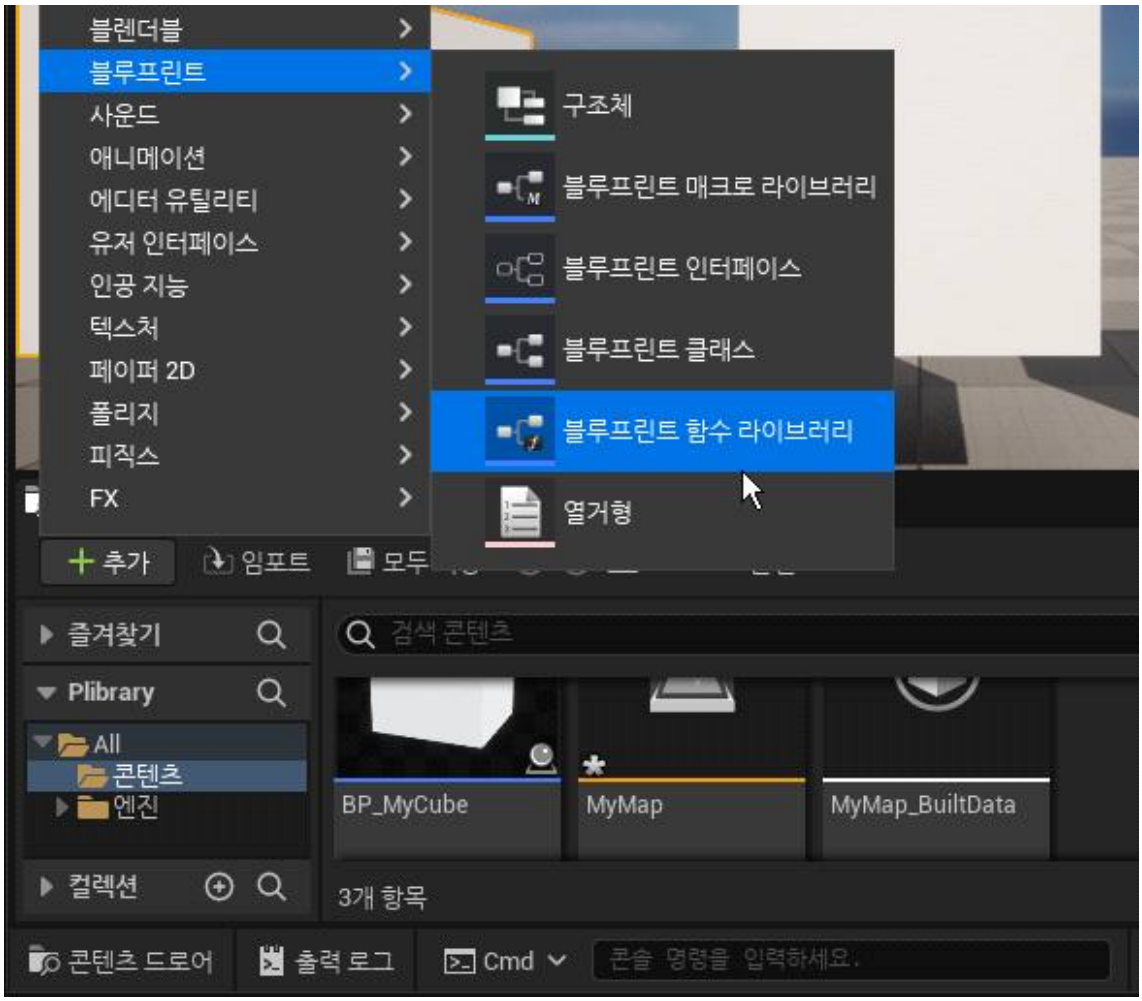
1. 새 프로젝트 `Plibrary`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Plibrary`로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자.
그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 `MyMap`으로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

2. 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.
그다음, 블루프린트 에디터에서 이벤트 그래프 탭을 선택하고 격자판의 빈 곳에서 **AddLocalRotation (Cube)** 노드를 검색해서 배치하자. 그리고 노드의 입력핀 **DeltaRotation**의 **Z**값에 1을 입력하자.
그다음, **Tick 이벤트** 노드의 출력 실행핀을 **AddLocalRotation** 노드의 입력 실행핀에 연결하자. 완성된 모습은 아래와 같다.



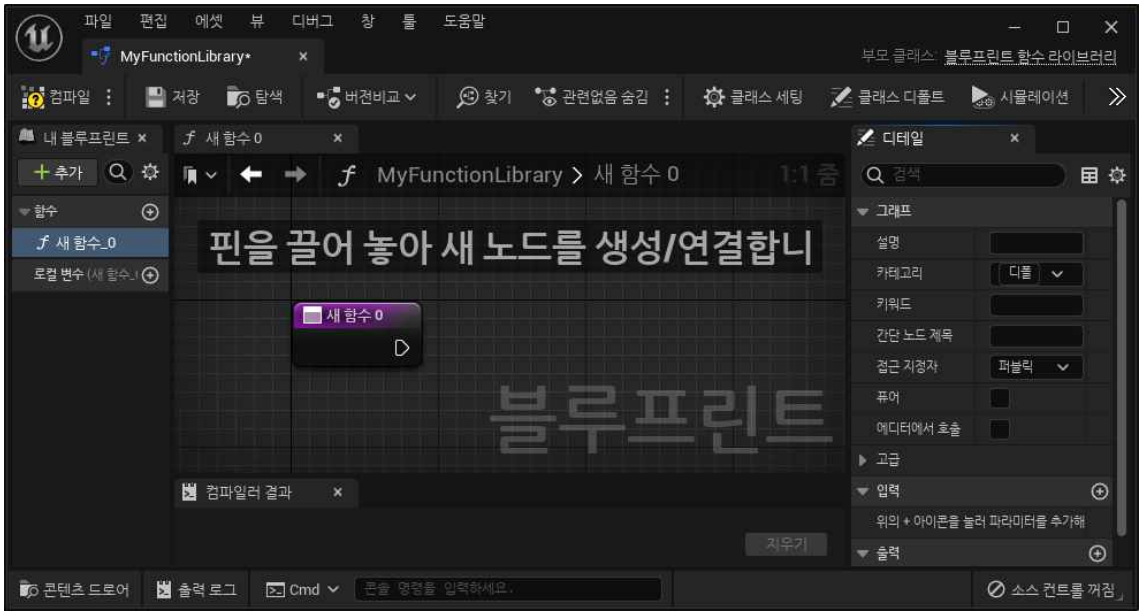
그다음, **BP_MyCube** 액터를 레벨에서 세 개를 배치하자. 각각 (300,-200,100), (300,0,100), (300,200,100)에 배치하자

3. 이제, 라이브러리에 대해서 학습을 시작해보자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 누르고 **블루프린트** » **블루프린트 함수 라이브러리**를 선택하자.



이름을 **MyFunctionLibrary**로 지정하자.

4. **MyFunctionLibrary**를 더블클릭하면 블루프린트 에디터가 열린다. **내 블루프린트** 탭에서는 **함수** 영역과 **로컬 변수** 영역만 있는 간소화된 형태로 나타난다. 디폴트로 하나의 함수가 생성되어 있다.



5. 이제부터 함수 라이브러리에 함수를 작성해보자.

함수 이름은 **GetMeanLocation**이라고 하자. 기존의 디폴트로 생성된 함수가 있으므로 이 함수의 이름을 수정하자.

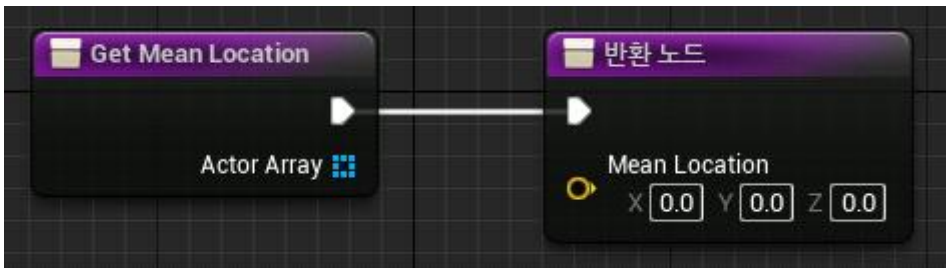
이 함수는 **Actor** 유형의 배열을 입력받아서 그 액터들의 평균 위치를 계산해서 리턴하는 함수로 구현하자.

함수가 생성되면 입력 인자와 출력 인자가 모두 없는 채로 생성된다. 따라서 입력 인자와 출력 인자를 추가해주어야 한다. 함수 노드를 선택한 후에 **디테일** 탭에서 **입력**과 **출력**을 수정하자.

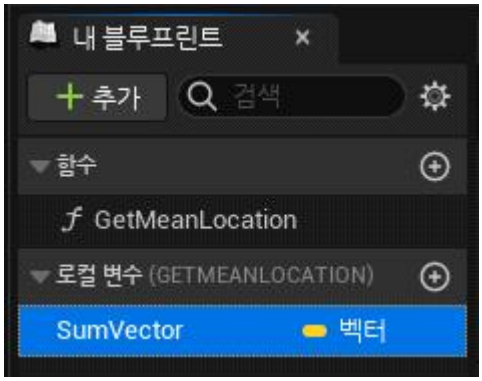
입력 인자는 이름을 **ActorArray**으로 하자. 유형은 **오브젝트 타입** » **액터** » **오브젝트 레퍼런스**를 선택하고 배열을 선택하자.

출력 인자는 이름을 **MeanLocation**으로 하자. 유형은 벡터로 하자.

6. 함수 그래프가 다음의 모습으로 될 것이다.

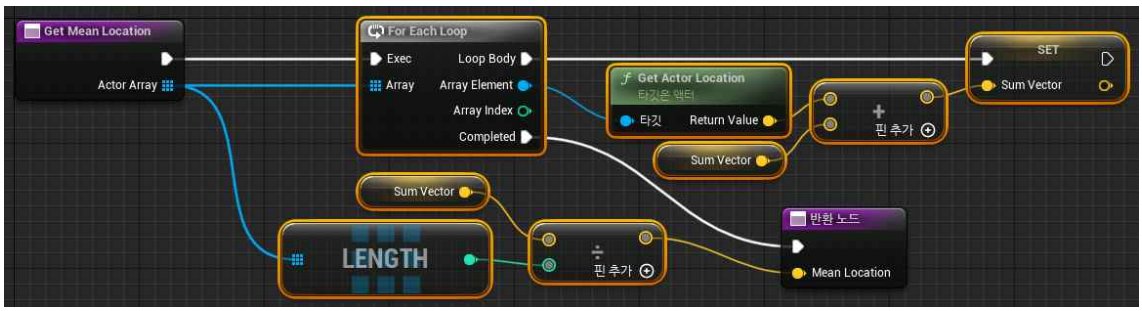


7. 그다음, **로컬 변수**를 하나 추가하자. 이름을 **SumVector**라고 하고 유형은 **Vector**로 하자. 컴파일하자.



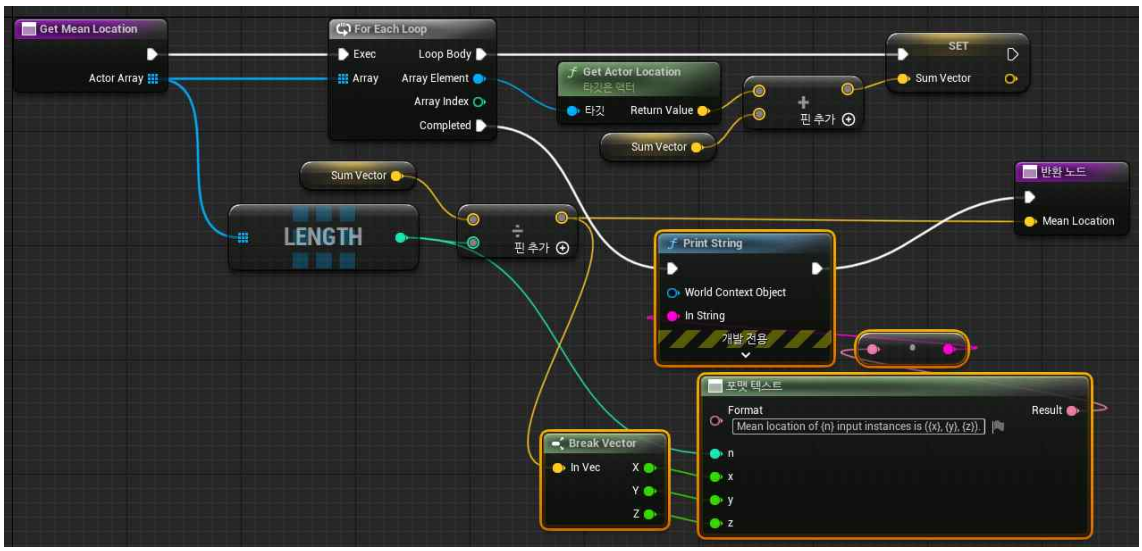
로컬 변수는 한 함수 내에서만 유효한 변수이다. 따라서 함수가 실행되기 시작할 때에 로컬 변수의 존재가 생기고 함수가 리턴된 후에는 로컬 변수의 존재도 사라지게 된다. 로컬 변수는 각 함수마다 배정되는 변수이므로 로컬 변수 뒤에 함수명이 함께 표시되어 있다. 다른 함수 그래프를 선택하면 로컬 변수 목록도 바뀌게 된다.
우리의 경우 로컬 변수 **SumVector**는 함수 **GetMeanLocation**에서만 존재하는 변수이다.

8. 그다음, 다음과 같이 **GetMeanLocation** 함수 그래프를 완성하자.



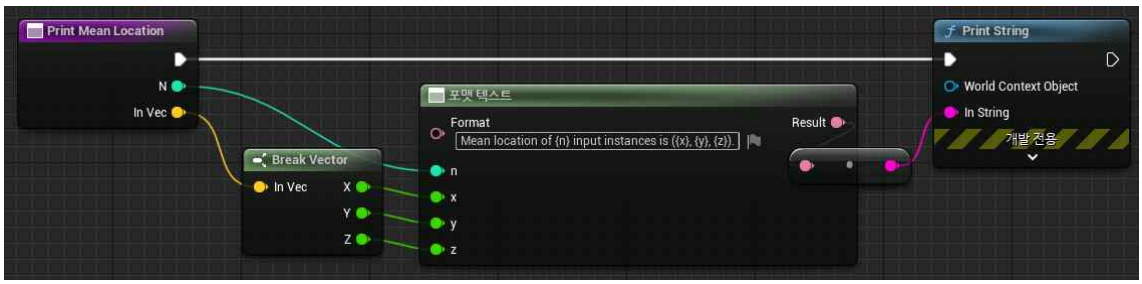
위의 그래프는 **ForEachLoop** 노드로 반복하면서 **SumVector**에 모든 위치 벡터를 합산한다. 그 이후에, 합산 벡터를 배열의 크기로 나누어서 평균 위치 벡터을 구해서 리턴한다.

9. 계산 결과를 확인하기 위해서 정보를 출력하는 노드들을 추가하자.

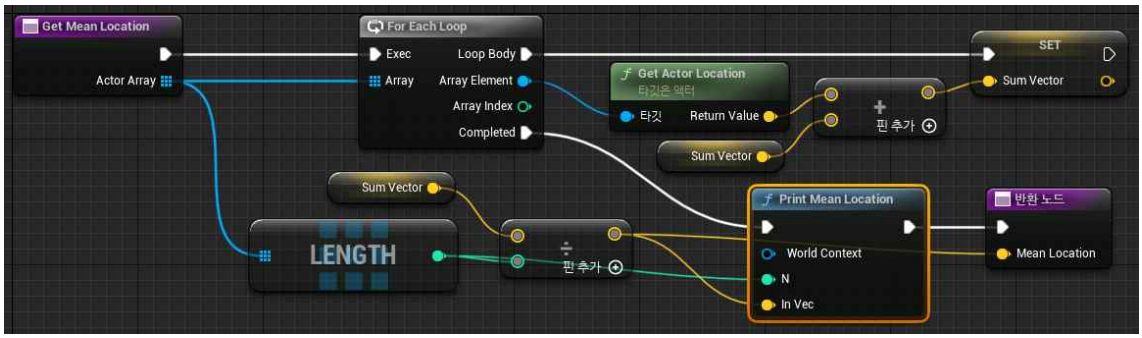


10. 출력과 연관된 노드들이 선택된 채로 우클릭하고 **함수로 접기**를 실행하자. 함수의 이름을 **PrintMe**

anLocation으로 하자.



11. 이제, **GetMeanLocation** 함수 그래프는 다음과 같이 완성되었다.



12. 레벨 블루프린트 에디터를 열자.

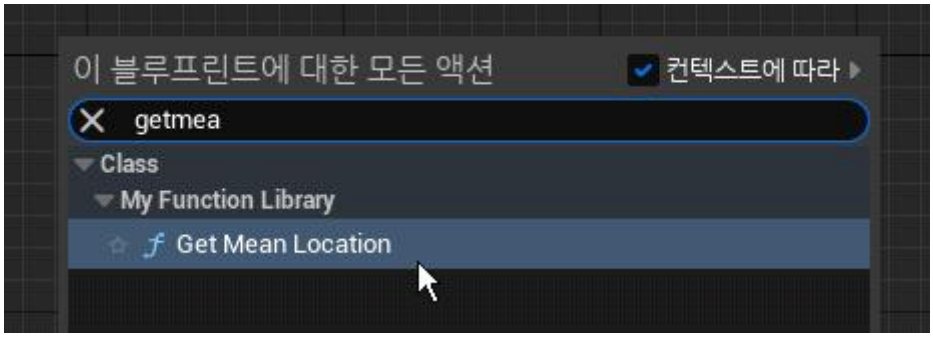
BeginPlay 이벤트 노드에서 실행핀을 드래그해서 **GetAllActorsOfClass** 함수 노드를 배치하자.

GetAllActorsOfClass 함수는 레벨에 배치된 액터들의 목록을 리턴하는 함수이다.

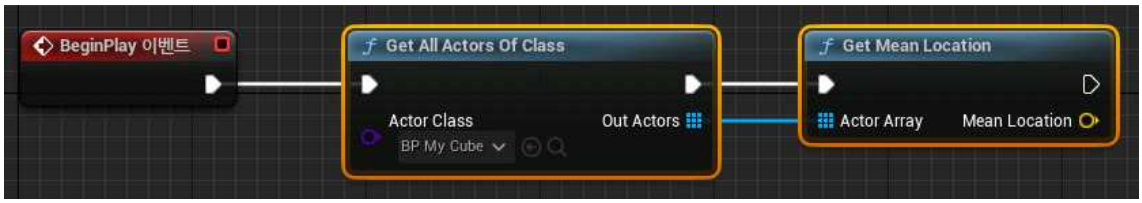
ActorClass 입력핀에는 필터링할 클래스 유형을 명시하면 된다. 우리는 **BP_MyCube**를 지정하자. 이렇게 지정하면 레벨에 배치된 **BP_MyCube** 액터 인스턴스들만 목록으로 만들어서 리턴해준다.

13. 이벤트 그래프 격자판의 빈 곳에서 우클릭하고 액션선택 창을 띄우자.

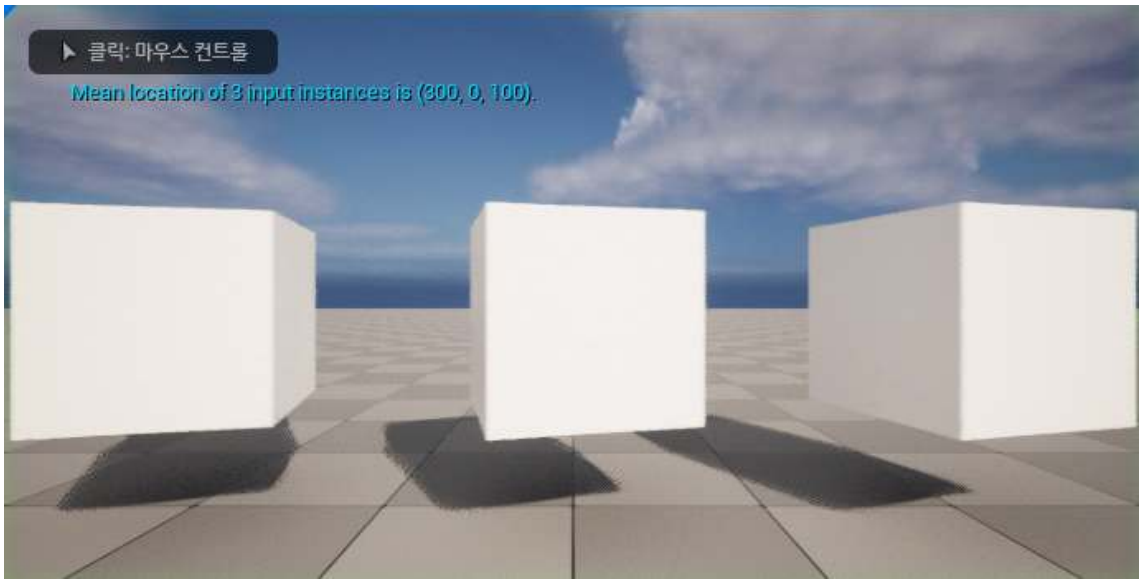
우리의 함수 라이브러리에 있는 함수인 **GetMeanLocation**를 검색해보자. **MyFunctionLibrary** 아래에 **GetMeanLocation** 함수가 나열되는 것을 확인할 수 있다. 이 함수를 선택하여 노드를 배치하자.



14. 그리고, 이전의 **GetAllActorsOfClass** 노드의 실행핀과 배치된 **GetMeanLocation** 노드의 실행핀을 연결하자. 또한, **GetAllActorsOfClass** 노드의 **OutActors** 출력핀을 **GetMeanLocation**의 **ActorArray** 입력핀에 연결하자. 이제 다음과 같이 되었다.



15. 플레이해보자. 다음과 같이 보일 것이다.

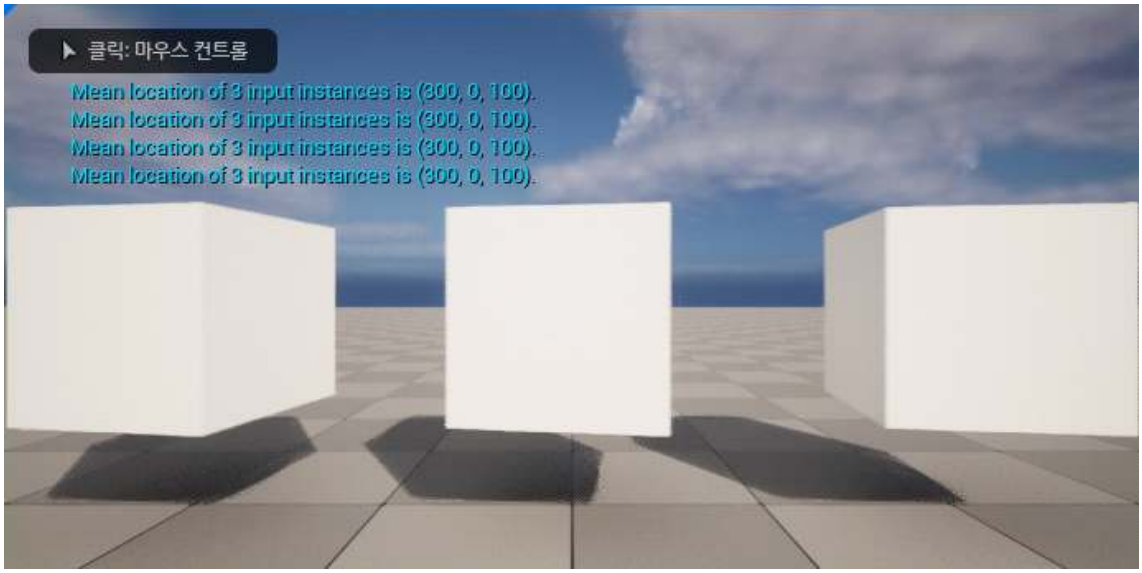


출력되는 문자열을 레벨 블루프린트에서 **BeginPlay 이벤트** 노드가 실행될 때에 한번 출력되는 문자열이다.

16. 이번에는 레벨 블루프린트가 아닌 블루프린트 클래스에서 작업해보자.
BP_MyCube 블루프린트 에디터를 열고 이벤트 그래프 탭에서 다음과 같이 노드를 추가하자.

이번에는 블루프린트 클래스에서 이전의 레벨 블루프린트에서와 동일한 **BeginPlay** 이벤트 그래프를 작성하였다. **GetMeanLocation** 함수를 이전에서와 동일하게 **BP_MyCube** 클래스 내에서도 사용할 수 있음을 알 수 있다.

17. 플레이해보자. 다음과 같이 보일 것이다.



동일한 함수 실행이 서로 다른 환경에서 수행되는 것을 확인할 수 있다.

지금까지, 함수 라이브러리를 만들고 사용하기에 대해서 학습하였다.

5. 매크로 라이브러리 만들고 사용하기

이 절에서는 **매크로 라이브러리**를 만들고 활용하는 방법에 대해서 학습한다.

함수 라이브러리는 다른 클래스의 함수를 사용할 수 없으므로 구현 기능이 제한적이다. 매크로 라이브러리는 이러한 제한점을 해결한다. 매크로 라이브러리에서는 생성 시에 특정 클래스를 지정한다. 그리고 매크로 내에서 지정한 클래스의 함수를 사용할 수 있도록 허용한다. 매크로 라이브러리에서 액터 클래스를 지정하였다면 매크로 내에서 액터 클래스의 모든 함수를 사용할 수 있다. 단 매크로 라이브러리의 매크로는 지정한 클래스의 파생 클래스 내에서만 사용해야 한다.

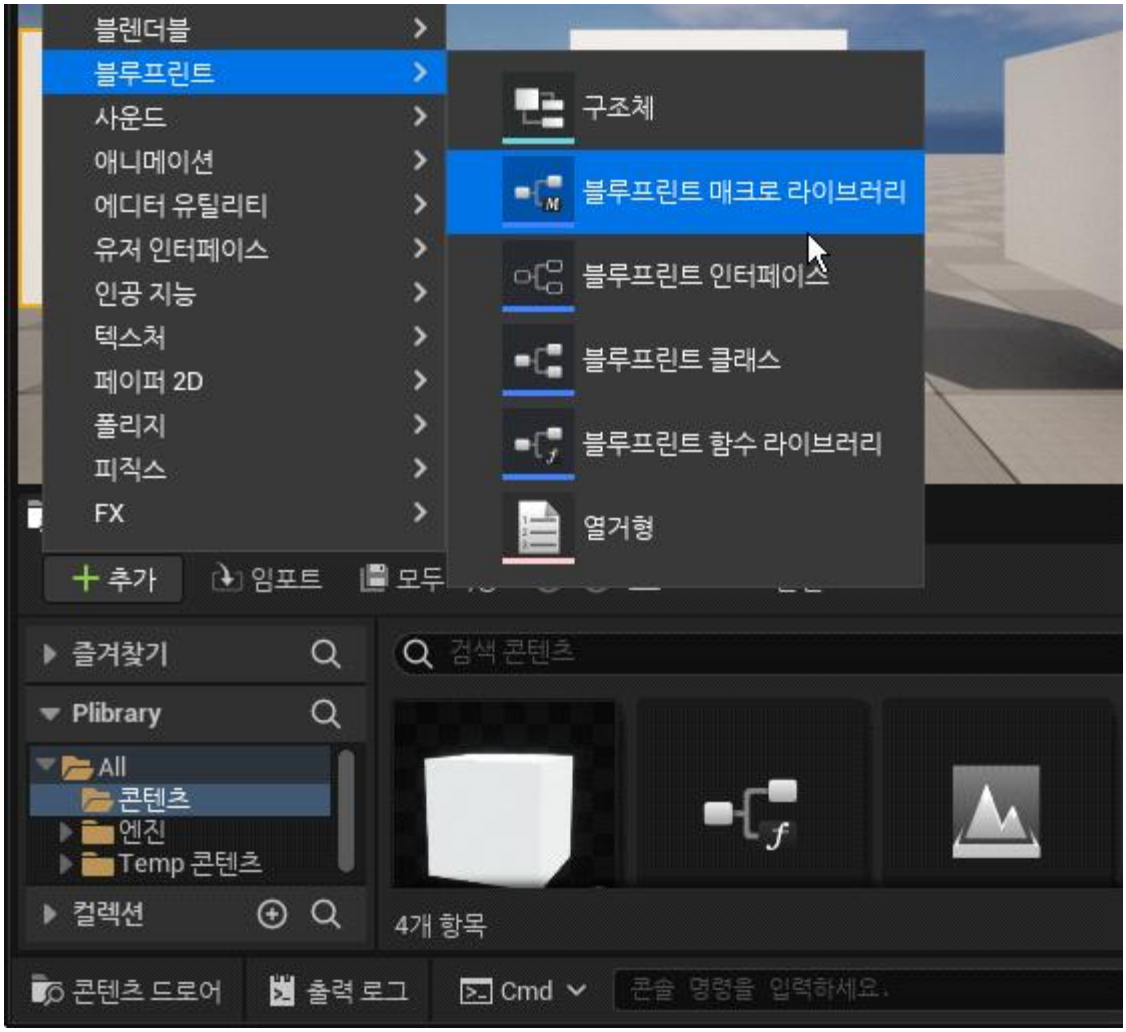
매크로 라이브러리에 대해서 더 알아보자. 매크로 라이브러리도 함수 라이브러리와 같이 여러 클래스에서 공통적으로 사용할 수 있는 매크로를 정의한다. 매크로 라이브러리는 여러 블루프린트에서 사용할 수 있는 매크로들의 모음이다. 자주 사용하는 노드 네트워크의 일부가 있다면 이를 매크로로 저장해두면 편리하게 사용할 수 있다.
매크로와 함수는 모두 유사한 목적으로 사용되지만 근본적인 차이가 있다. 기본적인 차이는 매크로는 컴파일되어서 저장해두는 개념이 아니라 매크로를 사용하는 그래프에 대치된 후에 나중에 컴파일되는 개념이다. 매크로에서는 컴파일 개념이 아니므로 컴파일 버튼이 비활성으로 되어 있다. 매크로는 노드 그래프를 빠르고 편하게 만들 수 있는 편리한 수단일 뿐이다. 따라서 매크로에서는 함수와는 다르게 로컬 변수를 가질 수도 없다.
또한, 매크로 라이브러리를 생성할 때에는 부모 클래스를 선택해야 한다. 매크로 라이브러리의 매크로는 매크로 라이브러리를 생성할 때에 선택한 부모 클래스의 파생 클래스에서만 사용해야 한다. 즉, 함수 라이브러리의 함수는 모든 클래스에서 사용할 수 있지만 특정 클래스이 멤버 함수를 사용할 수 없는 단점이 있다. 매크로 라이브러리의 매크로는 선택한 부모 클래스의 멤버 함수를 사용할 수 있지만 그 파생 클래스에서만 사용해야 한다.

<참고> 매크로 라이브러리에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/blueprint-macro-library-in-unreal-engine/

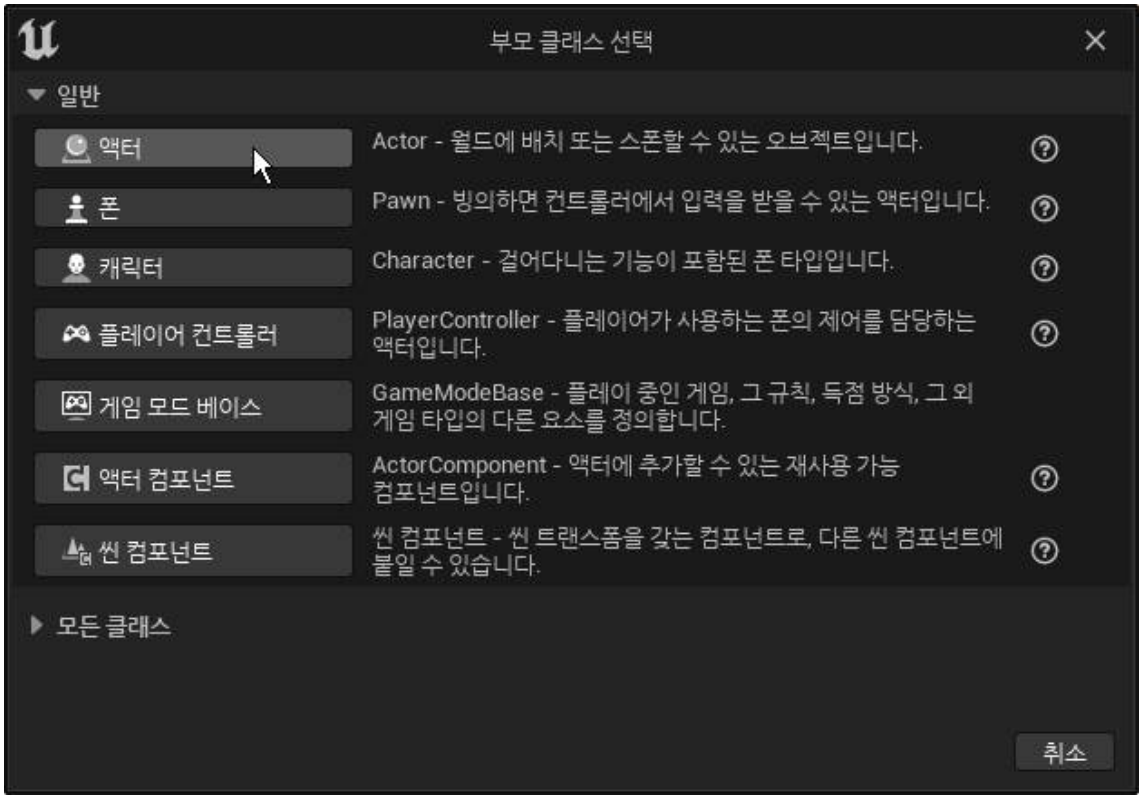
---

1. 이전 프로젝트 `Plibrary`에서 이어서 계속하자.

2. 블루프린트 매크로 라이브러리를 만들어보자.
먼저, **콘텐츠 브라우저**에서 **+추가**를 누르고 **블루프린트** » **블루프린트 매크로 라이브러리**를 선택하자.

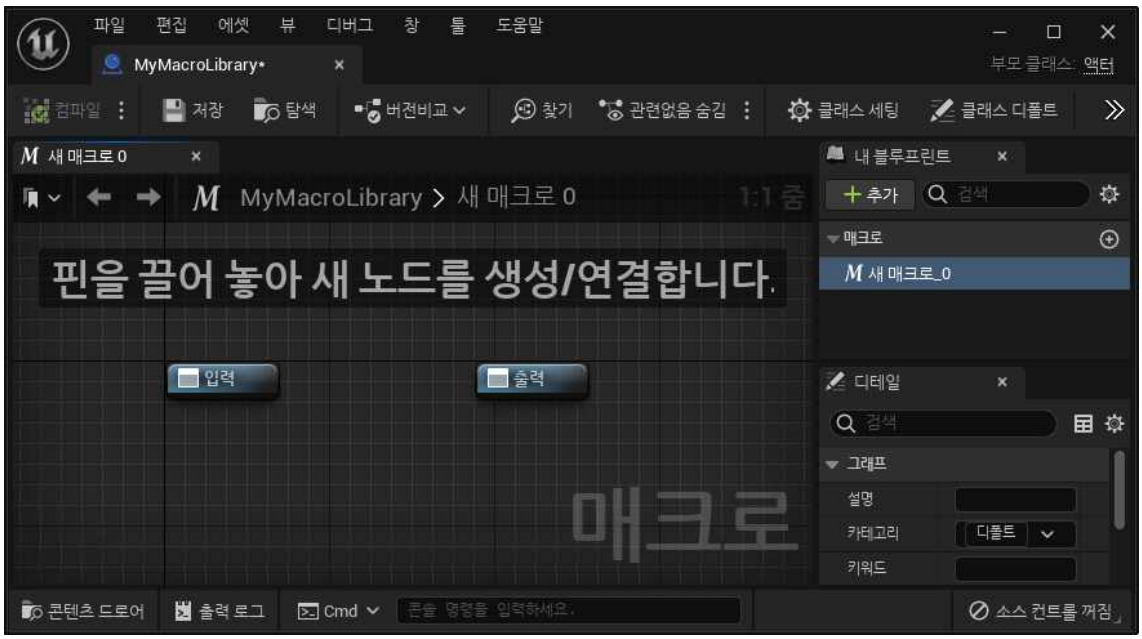


3. 부모 클래스 선택 창이 뜬다. 우리는 **Actor**를 선택하자.



그다음, 생성된 매크로 라이브러리의 이름을 **MyMacroLibrary**로 지정하자.

4. **MyMacroLibrary**를 더블클릭하면 블루프린트 에디터가 열린다.
디폴트로 하나의 매크로가 생성되어 있을 것이다.
매크로에는 입력 노드와 출력 노드가 배치되어 있다.

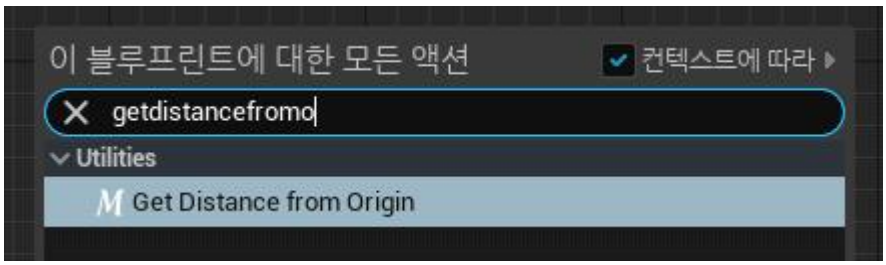


5. 매크로의 이름을 **GetDistanceFromOrigin**로 수정하자.
그다음, **디테일** 탭에서 출력 인자를 추가하자. 이름은 **Distance**로 하고 유형은 **플로트**로 하자.

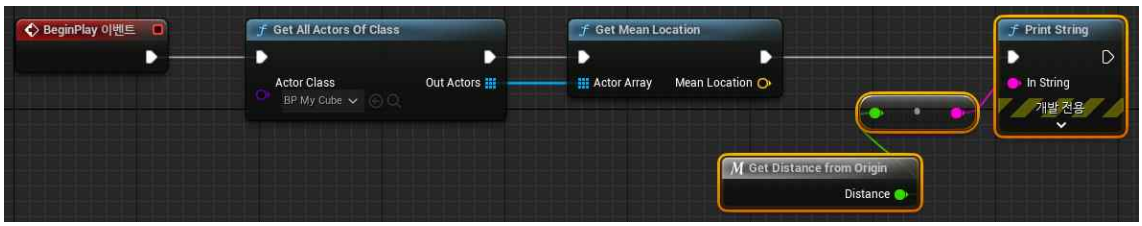
격자판의 빈 곳에서 우클릭하고 **GeActorLocation** 노드를 검색하여 배치하자. 이 노드는 **Actor**의 멤버 함수이다. 우리는 부모 클래스를 **Actor**로 하였으므로 매크로 내에서 **Actor**의 멤버 함수를 사용할 수 있는 것이다.
그다음, **VectorLength** 노드를 배치하고. 다음과 같이 그래프를 완성하자.

입력 인자가 없으므로 입력 노드는 사용할 필요가 없다.
저장하자. 매크로이므로 컴파일할 필요는 없다.

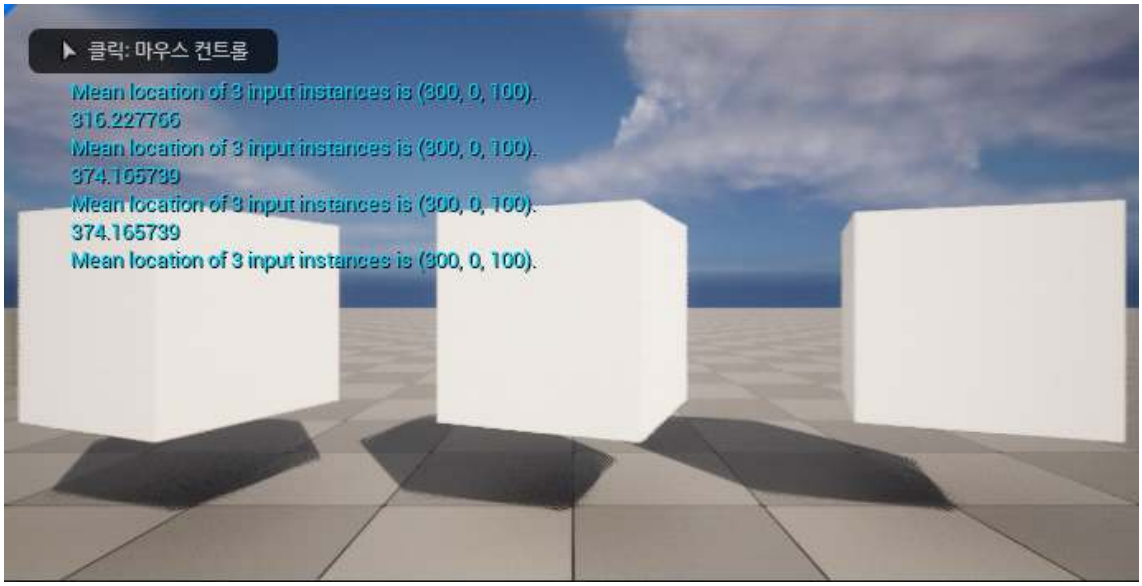
6. 이제 **BP_MyCube** 블루프린트 에디터에서 이벤트 그래프 탭으로 이동하자.
격자판의 빈 곳에서 **GetDistanceFromOrigin**를 검색해보자.



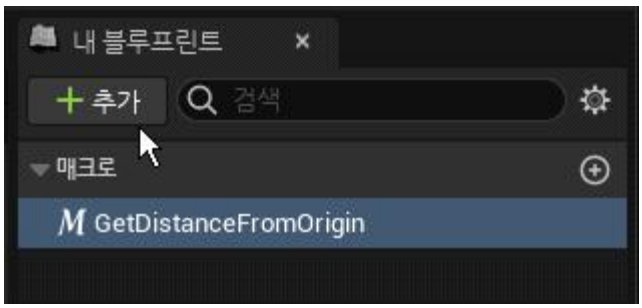
7. **BeginPlay 이벤트** 그래프에서 배치된 매크로 노드를 연결하여 다음과 같이 그래프를 완성하자.



8. 플레이해보자. 다음과 같이 출력될 것이다. 각 **BP_MyCube** 인스턴스가 자신의 거리를 출력하고 있다.



9. 위의 매크로의 경우에서는 실행핀이 없었다. 실행핀이 없어서 순수 함수 형식의 노드만 사용할 수 있었고 실행핀이 있는 노드는 사용할 수 없는 단점이 있었다.
그러나 매크로에 실행핀을 추가하여 더욱 유연한 그래프를 만들 수도 있다.
이제, 또다른 매크로를 추가해보자.
MyMacroLibrary 에디터에서 오른쪽의 **내 블루프린트** 탭에 **+추가**를 클릭하고 **매크로**를 선택하여 새 매크로를 추가하자.



매크로의 이름은 **DisplayDistanceFromOrigin**로 하자.

10. 이번에는 매크로에 입력 인자를 추가해보자.
매크로 **DisplayDistanceFromOrigin**의 디테일 탭에서 **입력** 영역의 + 아이콘을 클릭하여 입력 인자를 추가하자. 이름을 **InExec**로 하고 유형을 **실행**으로 하자.
그다음, 출력 인자를 추가하자. 이름을 **OutExec**로 하고 유형을 **실행**으로 하자.
유형에 기존의 변수에서는 없었던 **실행** 유형이 생겼다. 이 유형은 매크로를 작성할 때에 실행핀이 필요한 경우에 이를 지원하기 위함이다.

11. 이제 **DisplayDistanceFromOrigin** 매크로의 그래프를 작성하자. 실행핀이 있으므로 실행핀이 필요한 노드를 사용할 수 있다. 우리는 **PrintString** 노드를 추가로 배치하여 다음과 같이 그래프가 되도록 하자.

저장하자.

12. 이제 **BP_MyCube** 블루프린트 에디터에서 이벤트 그래프 탭으로 이동하자.
BeginPlay 이벤트 그래프 뒤쪽의 **GetDistanceFromOrigin**를 노드와 **PrintString** 노드를 제거하고 그 자리에 **DisplayDistanceFromOrigin** 매크로를 추가하자.
BeginPlay 이벤트 그래프를 다음과 같이 된다.

이전과 동일하게 실행될 것이다.

지금까지, 매크로 라이브러리를 만들고 사용하기에 대해서 학습하였다.

□