Chapter 10: 공간과 움직임

레트로의 유니티 게임프로그래밍 에센스

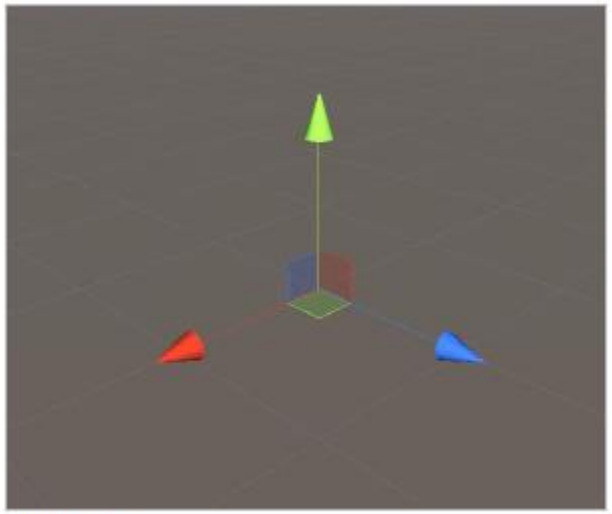
이 제 민 지음

Contents

CHAPTER 10 공간과 움직임

10.1 유니티 공간

- 3D 공간에 배치하는 3D 오브젝트는 위치를 표현한 값, 즉 좌표를 가짐.
  - 좌표를 측정할 기준이 될 원점의 위치와 X, Y, Z축 방향을 설정하여 물체가 어디에 배치되어 있는지 표현하는 기준과 체계를 좌표계라 부름.
  - 좌표계는 공간에서 '어떤 방향으로' 얼마만큼 이동한 거리에 배치할 것인지 결정하는 기준임.



▶ 3D 좌표계

10.1 유니티 공간

- 10.1.1 전역 공간
  - 벡터는 위치, 방향, 회전, 속도, 크기를 비롯한 온갖 종류의 계산에 사용됨.
  - 유니티는 3D 벡터를 나타내는 Vector3를 사용해서 3D 공간에서의 x, y, z 좌표를 표현함.

10.1 유니티 공간

◦ 10.1.1 전역 공간

- 전역 공간은 월드의 중심이라는 절대 기준이 존재하는 공간이며 월드 공간이라 부르기도 함.
- 전역 공간에서 X, Y, Z 방향을 정하고 좌표를 계산하는 기준을 전역 좌표계라고 함.

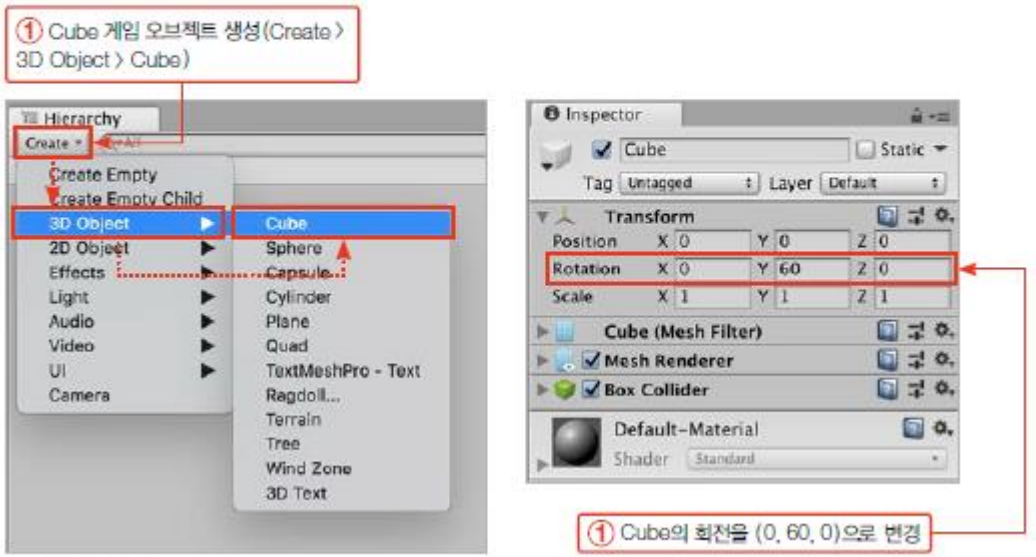
[과정 01] 새로운 프로젝트 생성하기

① 새로운 3D 유니티 프로젝트를 만듭니다.

[과정 02] 3D 큐브 추가하기

① Cube 게임 오브젝트 생성(**하이어라키** 창에서 Create > 3D Object > Cube 선택)

② Cube 게임 오브젝트의 **회전**을 (0, 60, 0)으로 변경



▶ 3D 큐브 추가

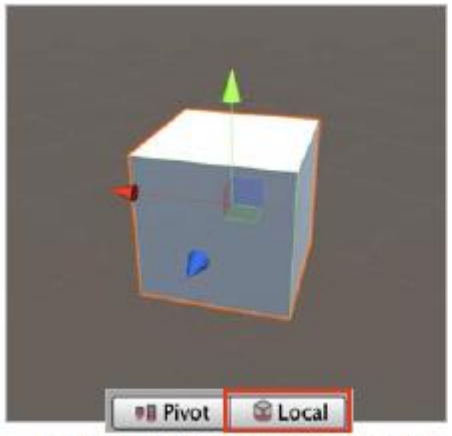
▶ 트랜스폼 기즈모 전환 버튼

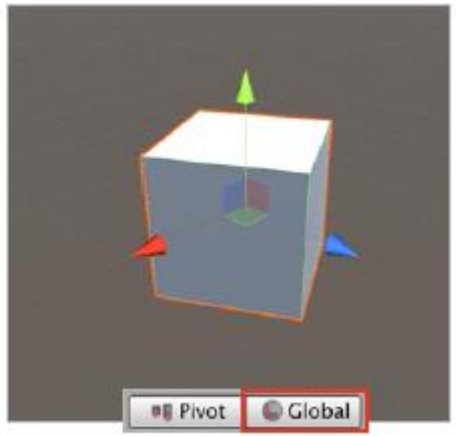
10.1 유니티 공간

[과정 03] 전역 공간 모드로 전환

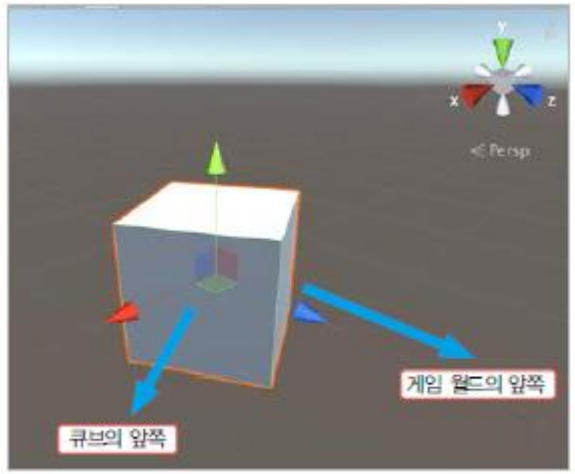
① Local/Global 전환 버튼을 클릭하여 Global로 전환

▶ 전역 공간으로 전환





▶ 오브젝트 공간 모드와 전역 공간 모드의 차이



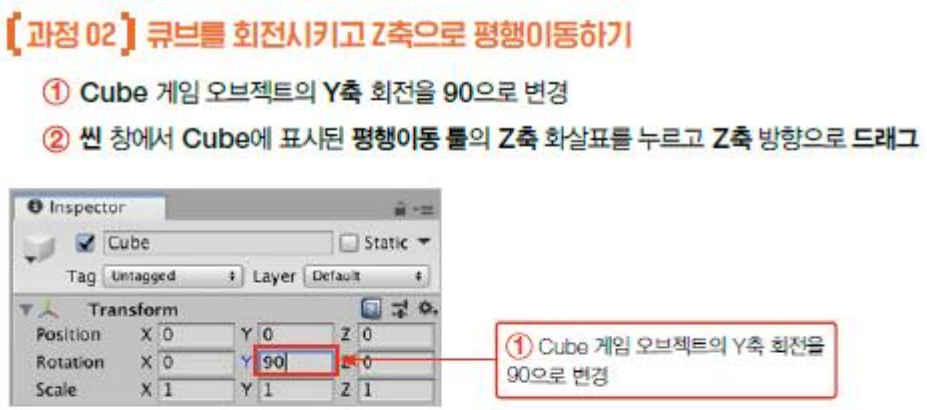
▶ 전역 공간과 오브젝트 공간에서의 앞쪽

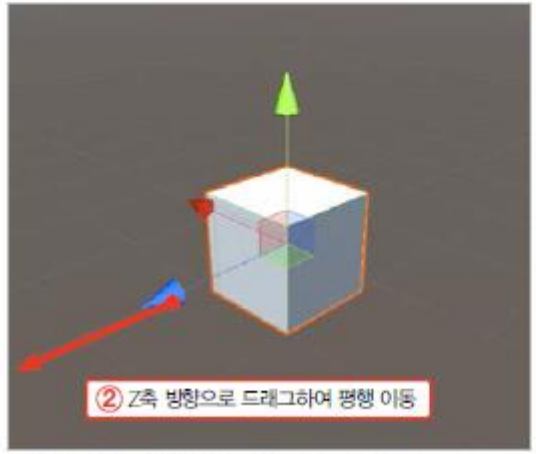
10.1 유니티 공간

◦ 10.1.2 오브젝트 공간

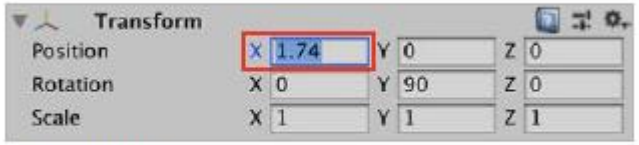
- 전역 좌표계와 원점을 기준으로 배치하는 전역 공간과 반대로 오브젝트 공간은 오브젝트 자신의 X, Y, Z 방향 (오브젝트 좌표계)을 배치 기준으로 사용함.

▶ 오브젝트 공간 모드로 전환





▶ 3D 큐브를 회전시키고 Z축으로 평행이동



▶ X 값이 변경됨

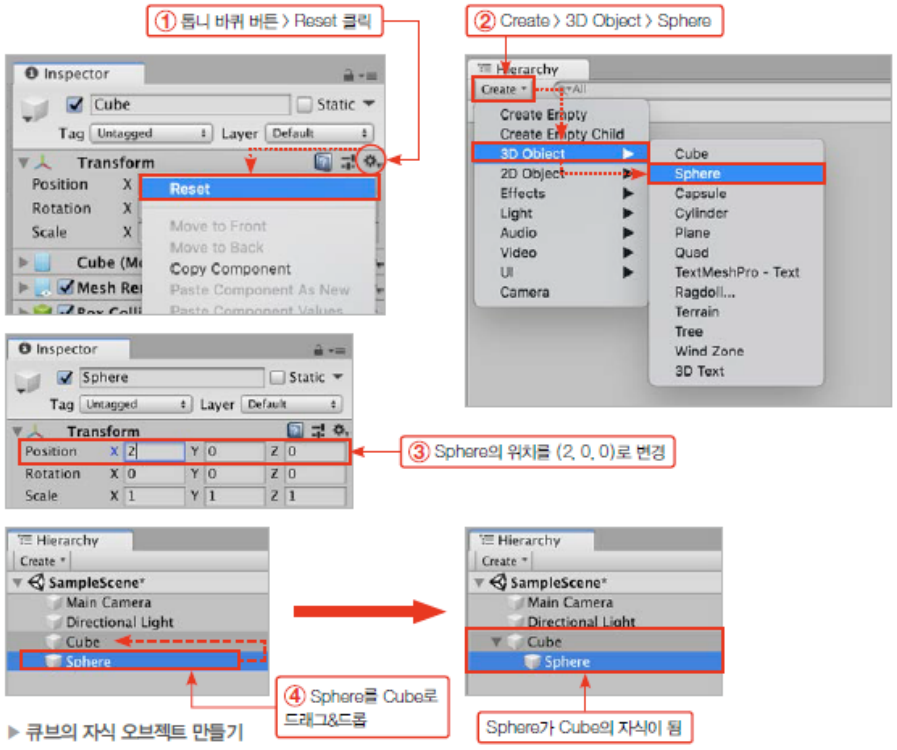
10.1 유니티 공간

◦ 10.1.3 지역 공간

- 지역 공간은 게임 월드나 오브젝트 자신이 아닌 자신의 부모 오브젝트를 기준으로 한 지역 좌표계로 좌표를 측정함.
- 인스펙터 창에 표시되는 게임 오브젝트의 위치, 회전, 스케일은 모두 지역 공간에서 측정된 값임.

[과정 01] 큐브의 자식 만들기

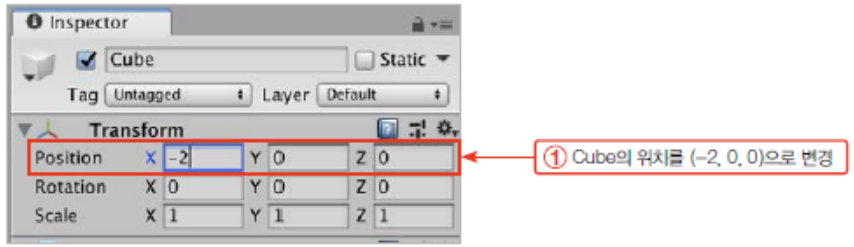
① **인스펙터** 창에서 Cube 게임 오브젝트의 Transform 컴포넌트의 **톱니바퀴** 버튼 > **Reset** 클릭
② Sphere 게임 오브젝트 생성(**하이어라키** 창에서 **Create** > **3D Object** > **Sphere** 클릭)
③ Sphere 게임 오브젝트의 위치를 (2, 0, 0)으로 변경
④ **하이어라키** 창에서 **Sphere**를 **Cube**로 **드래그&드롭**



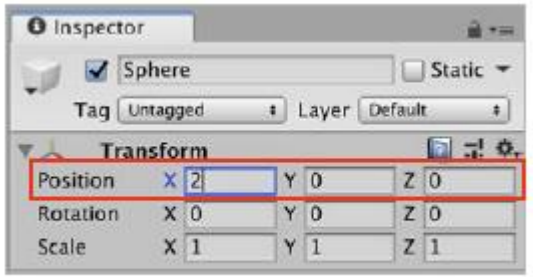
▶ 큐브의 자식 오브젝트 만들기

[과정 02] 큐브 위치 옮기기

① **하이어라키** 창에서 **Cube** 선택 > **Cube**의 위치를 **(−2, 0, 0)**으로 변경



▶ 큐브 위치 옮기기

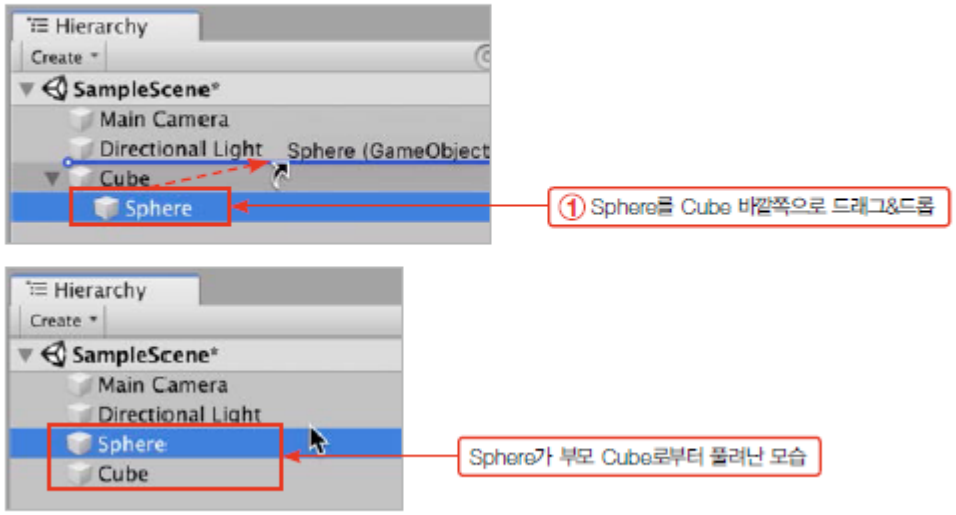


▶ 변경되지 않은 Sphere의 위치

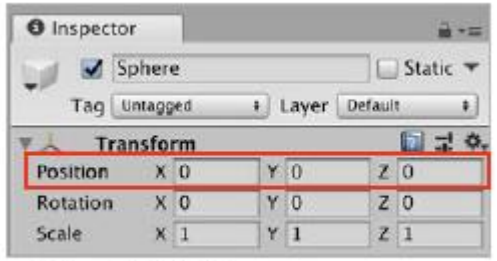
10.1 유니티 공간

[과정 03] 구를 부모 Cube로부터 풀어주기

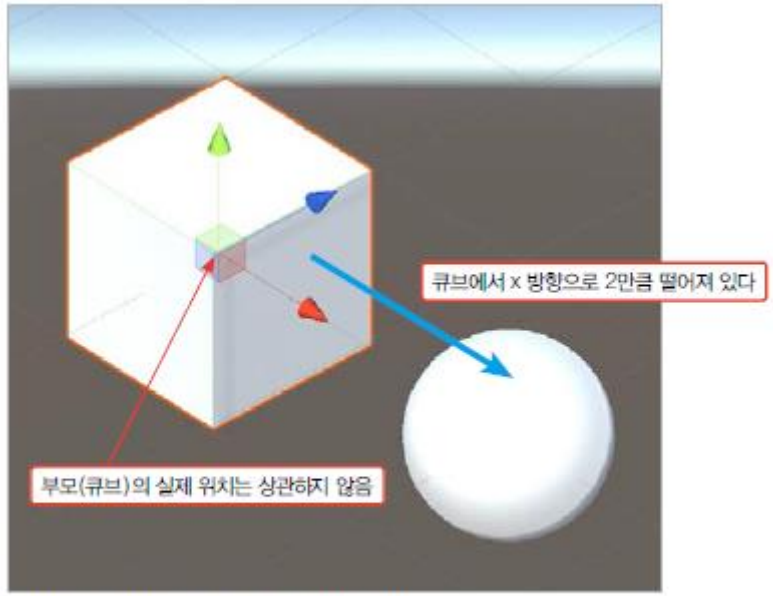
① **하이어라키** 창에서 **Sphere**를 **Cube** 바깥쪽으로 **드래그&드롭**하여 부모로부터 풀어주기



▶ Sphere를 자신의 부모 Cube로부터 풀어주기



▶ Sphere의 전역 위치



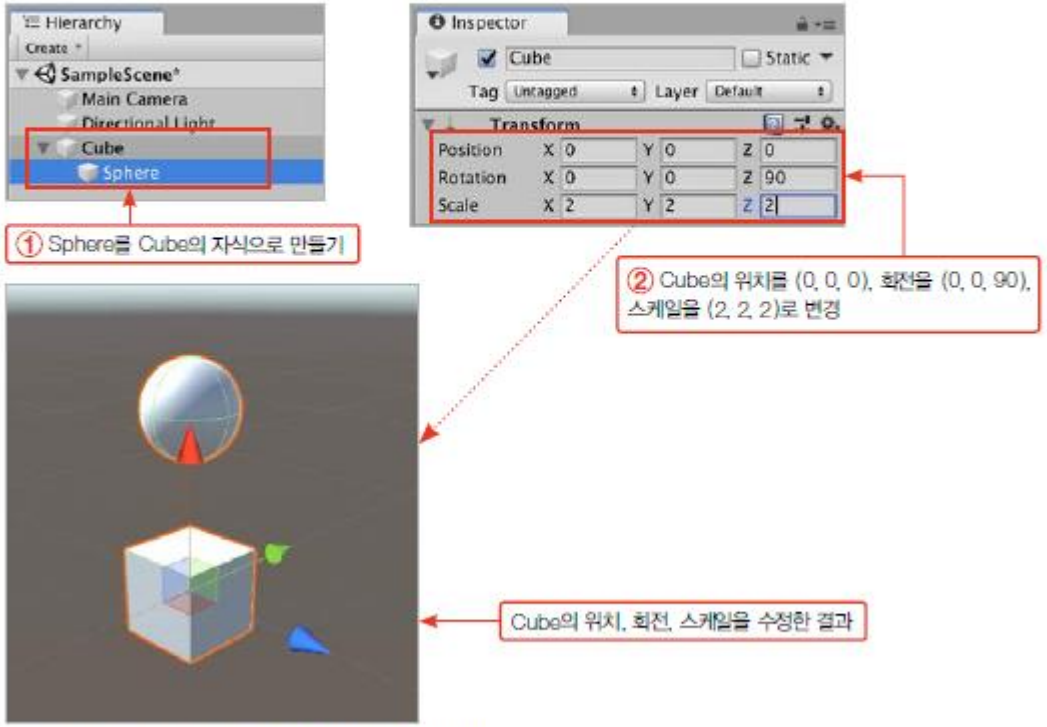
▶ 모든 것을 부모를 기준으로 생각하게 된다

10.1 유니티 공간

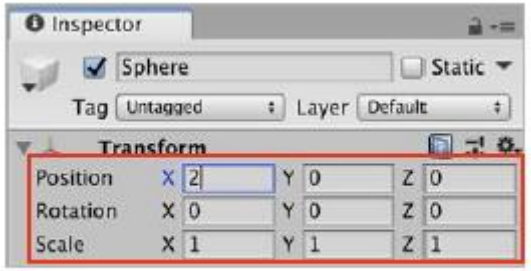
[과정 04] 부모에 의한 자식의 회전과 스케일 변경 확인하기

① Sphere를 Cube의 자식으로 만들기(**하이어라키** 창에서 Sphere를 Cube로 **드래그&드롭**)

② **하이어라키** 창에서 Cube 선택 〉 Cube의 **위치**를 (0, 0, 0), **회전**을 (0, 0, 90), **스케일**을 (2, 2, 2)로 **변경**



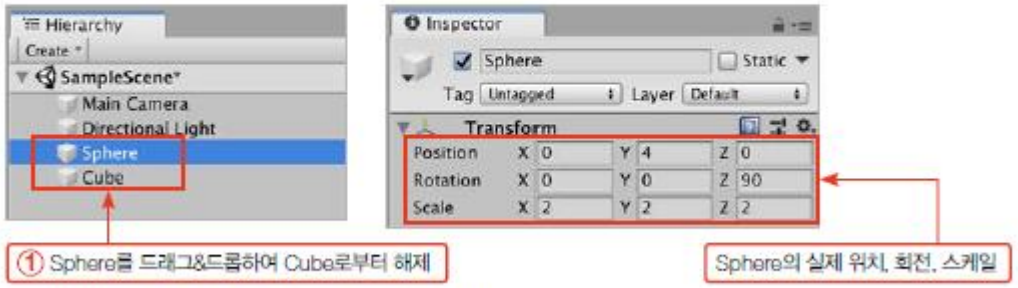
▶ 부모에 의한 자식의 회전과 스케일 변경 확인하기



▶ 변하지 않은 Sphere의 위치, 회전, 스케일

[과정 05] 구를 부모 Cube로부터 풀어주기

① **하이어라키** 창에서 Sphere를 Cube 바깥쪽으로 **드래그&드롭**하여 부모로부터 풀어주기



▶ Sphere를 자신의 부모 Cube로부터 풀어주기

- 지역 위치 (2, 0, 0) → 전역 위치 (0, 4, 0)
- 지역 회전 (0, 0, 0) → 전역 회전 (0, 0, 90)
- 지역 스케일 (1, 1, 1) → 전역 스케일 (2, 2, 2)

10.1 유니티 공간

- 10.1.4 지역 공간과 오브젝트 공간의 차이

  - 유니티는 편의상 지역 공간과 오브젝트 공간을 합쳐 지역 공간으로 부르고 있음.

- **전역 공간 :** 게임 월드의 원점을 기준으로 위치를 측정합니다.
- **지역 공간 :** 자신의 부모 게임 오브젝트를 기준으로 위치를 측정합니다.
- **오브젝트 공간 :** 자기 자신을 기준으로 위치를 측정합니다.

- 지역 공간에서 위치, 회전, 스케일값 측정 : 부모 게임 오브젝트를 기준으로 측정(지역 공간)
- 지역 공간에서 평행이동 : 게임 오브젝트 자신의 방향을 기준으로 평행이동(오브젝트 공간)

10.2 오브젝트의 이동과 회전

- 공간에 관해 모두 설명했으니 이제 코드로 공간 내부에서 물체를 배치하고 움직여봄.
  - 물리 처리를 거치지 않고 트랜스폼 컴포넌트의 위치나 회전 값을 직접 변경하여 게임 오브젝트를 움직이는 것이 더 좋은 경우도 있음.

10.2 오브젝트의 이동과 회전

◦ 10.2.1 스크립트 작성하기

- 스크립트로 전역 공간과 지역 공간을 구분하여 Sphere 게임 오브젝트와 Cube 게임 오브젝트를 움직여보겠음.

[과정 01] Move 스크립트 생성

① **프로젝트** 창에서 Create > C# Script 클릭

② 생성된 스크립트의 이름을 **Move**로 변경하고 열기

[과정 02] Move 스크립트 작성

① **Move** 스크립트를 다음과 같이 완성

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Move : MonoBehaviour {
    public Transform childTransform; // 움직일 자식 게임 오브젝트의 트랜스폼

    void Start() {
        // 자신의 전역 위치를 (0, -1, 0)으로 변경
        transform.position = new Vector3(0, -1, 0);
        // 자식의 지역 위치를 (0, 2, 0)으로 변경
        childTransform.localPosition = new Vector3(0, 2, 0);

        // 자신의 전역 회전을 (0, 0, 30)으로 변경
        transform.rotation = Quaternion.Euler(new Vector3(0, 0, 30));
        // 자식의 지역 회전을 (0, 60, 0)으로 변경
        childTransform.localRotation = Quaternion.Euler(new Vector3(0, 60, 0));
    }

    void Update() {
        if (Input.GetKey(KeyCode.UpArrow))
        {
            // 위쪽 방향키를 누르면 초당 (0, 1, 0)의 속도로 평행이동
            transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.DownArrow))
        {
            // 아래쪽 방향키를 누르면 초당 (0, -1, 0)의 속도로 평행이동
            transform.Translate(new Vector3(0, -1, 0) * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.LeftArrow))
        {
            // 왼쪽 방향키를 누르면
            // 자신을 초당 (0, 0, 180) 회전
            transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime);
            // 자식을 초당 (0, 180, 0) 회전
            childTransform.Rotate(new Vector3(0, 180, 0) * Time.deltaTime);
        }

        if (Input.GetKey(KeyCode.RightArrow))
        {
            // 오른쪽 방향키를 누르면
            // 자신을 초당 (0, 0, -180) 회전
            transform.Rotate(new Vector3(0, 0, -180) * Time.deltaTime);
            // 자식을 초당 (0, -180, 0) 회전
            childTransform.Rotate(new Vector3(0, -180, 0) * Time.deltaTime);
        }
    }
}
```

10.2 오브젝트의 이동과 회전

◦ 10.2.2 위치와 회전 결정하기

- Start( ) 메서드에서는 트랜스폼 컴포넌트를 나타내는 Transform 타입에 내장된 변수들의 값을 변경하여 Sphere 게임 오브젝트와 Cube 게임 오브젝트의 위치와 회전을 변경함.

```
void Start() {
    // 자신의 전역 위치를 (0, -1, 0)으로 변경
    transform.position = new Vector3(0, -1, 0);
    // 자식의 지역 위치를 (0, 2, 0)으로 변경
    childTransform.localPosition = new Vector3(0, 2, 0);

    // 자신의 전역 회전을 (0, 0, 30)으로 변경
    transform.rotation = Quaternion.Euler(new Vector3(0, 0, 30));
    // 자식의 지역 회전을 (0, 60, 0)으로 변경
    childTransform.localRotation = Quaternion.Euler(new Vector3(0, 60, 0));
}
```

10.2 오브젝트의 이동과 회전

◦ 10.2.3 Update( ) 메서드 구현

- Update( ) 메서드에서는 키보드 입력에 따라 평행이동과 회전을 함.

```
void Update() {
    if (Input.GetKey(KeyCode.UpArrow))
    {
        // 위쪽 방향키를 누르면 초당 (0, 1, 0)의 속도로 평행이동
        transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime);
    }

    if (Input.GetKey(KeyCode.DownArrow))
    {
        // 아래쪽 방향키를 누르면 초당 (0, -1, 0)의 속도로 평행이동
        transform.Translate(new Vector3(0, -1, 0) * Time.deltaTime);
    }

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        // 왼쪽 방향키를 누르면
        // 자신을 초당 (0, 0, 180) 회전
        transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime);
        // 자식을 초당 (0, 180, 0) 회전
        childTransform.Rotate(new Vector3(0, 180, 0) * Time.deltaTime);
    }

    if (Input.GetKey(KeyCode.RightArrow))
    {
        // 오른쪽 방향키를 누르면
        // 자신을 초당 (0, 0, -180) 회전
        transform.Rotate(new Vector3(0, 0, -180) * Time.deltaTime);
        // 자식을 초당 (0, -180, 0) 회전
        childTransform.Rotate(new Vector3(0, -180, 0) * Time.deltaTime);
    }
}
```

키보드 입력에 따라 다음과 같이 평행이동과 회전을 실행합니다.

- **키보드 위쪽 방향키**
  - 자신의 Y 방향으로 초당 1의 속도로 평행이동
- **키보드 아래쪽 방향키**
  - 자신의 Y 방향으로 초당 −1의 속도로 평행이동
- **키보드 왼쪽 방향키**
  - 자신을 Z축 기준으로 초당 180도 반시계 방향으로 회전
  - 자식을 Y축 기준으로 초당 180도 반시계 방향으로 회전
- **키보드 오른쪽 방향키**
  - 자신을 Z축 기준으로 초당 180도 시계 방향으로 회전
  - 자식을 Y축 기준으로 초당 180도 시계 방향으로 회전

10.2 오브젝트의 이동과 회전

◦ 10.2.4 평행이동

- 평행이동은 Transform 타입이 제공하는 Translate( ) 메서드로 실행할 수 있음.

```
if (Input.GetKey(KeyCode.UpArrow))
{
    transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime);
}

if (Input.GetKey(KeyCode.DownArrow))
{
    transform.Translate(new Vector3(0, -1, 0) * Time.deltaTime);
}
```

전역 평행이동과 지역 평행이동

• **전역 공간 기준으로 평행이동**

아래 코드는 자신의 Y축 방향과 상관없이 전역 공간을 기준으로 한 Y축 방향으로 평행이동합니다.

```
transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime, Space.World);
```

• **지역 공간 기준으로 평행이동**

아래 코드는 게임 월드의 Y축 방향과 상관없이 자신의 Y축 방향으로 평행이동합니다.

```
transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime, Space.Self);
```

10.2 오브젝트의 이동과 회전

- 10.2.5 회전

  - Update( ) 메서드에서 키보드 왼쪽 방향키나 오른쪽 방향키를 눌렀을 때 게임 오브젝트를 회전하는 부분을 봄.

```
if (Input.GetKey(KeyCode.LeftArrow))
{
    transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime);
    childTransform.Rotate(new Vector3(0, 180, 0) * Time.deltaTime);
}

if (Input.GetKey(KeyCode.RightArrow))
{
    transform.Rotate(new Vector3(0, 0, -180) * Time.deltaTime);
    childTransform.Rotate(new Vector3(0, -180, 0) * Time.deltaTime);
}
```

전역 회전과 지역 회전

- **전역 공간 기준으로 회전하기**

  아래 코드는 자신의 Z축 기울기와 상관없이 전역 공간의 Z축을 기준으로 1초에 180도 회전합니다.

```
transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime, Space.World);
```

- **지역 공간 기준으로 회전하기**

  아래 코드는 전역 공간의 Z축 방향과 상관없이 자신의 Z축을 기준으로 1초에 180도 회전합니다.

```
transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime, Space.Self);
```

10.2 오브젝트의 이동과 회전

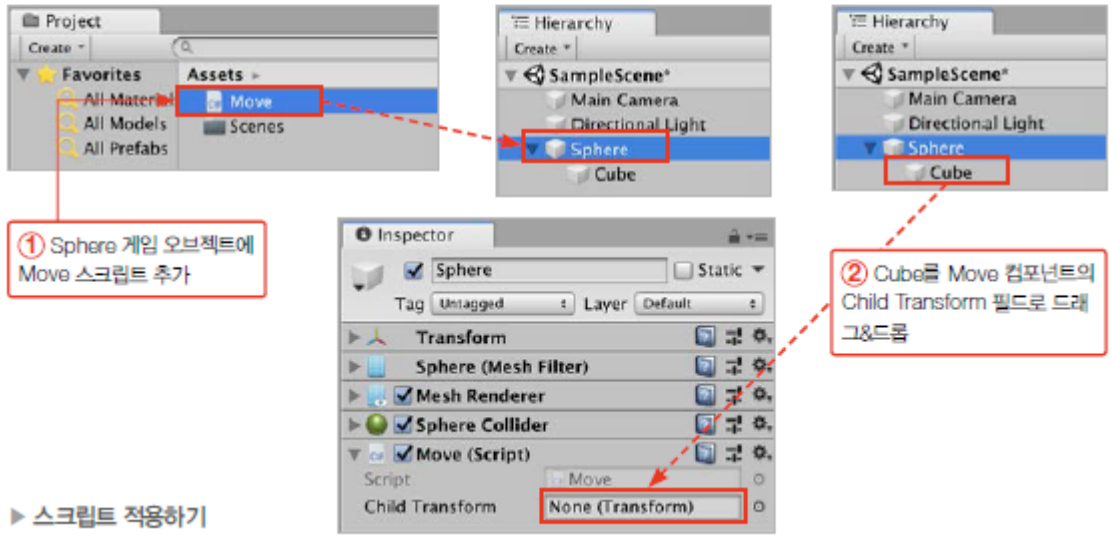
◦ 10.2.6 테스트

- [Ctrl+S]로 작성한 Move 스크립트를 저장하고 유니티 에디터로 돌아감. 작성한 코드를 테스트함.

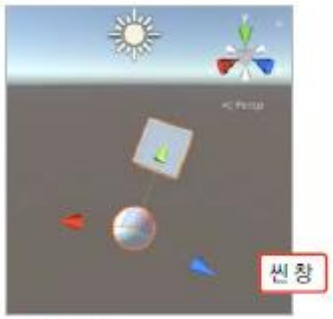
[과정 01] 스크립트 적용하기

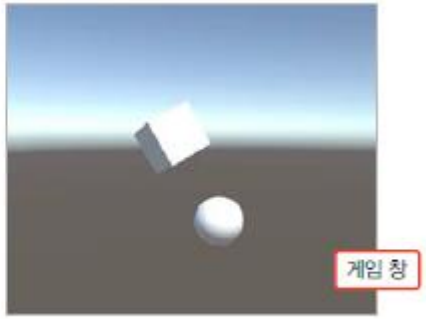
① Sphere 게임 오브젝트에 Move 스크립트 추가(Move 스크립트를 하이어라키 창의 Sphere로 드래그&드롭)

② 하이어라키 창에서 Sphere 게임 오브젝트 선택 > 하이어라키 창의 Cube를 Move 컴포넌트의 Child Transform 필드로 드래그&드롭



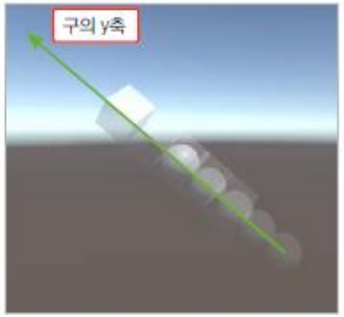
▶ 스크립트 적용하기

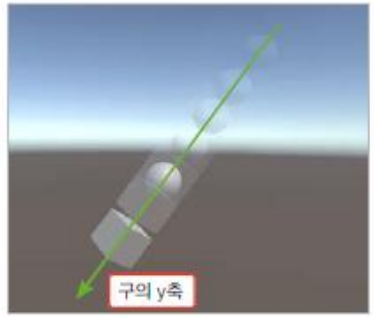




▶ 실행 결과

▶ 키보드 오른쪽 방향키를 눌렀을 때의 회전





▶ 키보드 위쪽 방향키를 눌렀을 때의 평행이동

10.3 벡터 연산으로 평행이동 구현하기

◦ 10.3.1 벡터의 속기

- Vector3 타입에는 속기shorthand라는 미리 만들어진 편리한 변수들이 있음.
- 자주 사용되는 Vector3 값은 속기를 사용해 다음과 같은 형태로 즉시 생성할 수 있음.

```
Vector3 position = Vector3.up;
```

Vector3의 속기

- Vector3.forward : new Vector3(0, 0, 1)
- Vector3.back : new Vector3(0, 0, -1)
- Vector3.right : new Vector3(1, 0, 0)
- Vector3.left : new Vector3(-1, 0, 0)
- Vector3.up : new Vector3(0, 1, 0)
- Vector3.down : new Vector3(0, -1, 0)

10.3 벡터 연산으로 평행이동 구현하기

◦ 10.3.2 트랜스폼의 방향

- 트랜스폼 컴포넌트를 표현하는 Transform 타입은 자신의 앞쪽, 뒤쪽, 오른쪽 등을 나타내는 방향벡터를 즉시 접근할 수 있는 변수들을 제공함.

Transform 타입의 방향

- transform.forward : 자신의 앞쪽을 가리키는 방향벡터
- transform.right : 자신의 오른쪽을 가리키는 방향벡터
- transform.up : 자신의 위쪽을 가리키는 방향벡터

- 자신의 뒤쪽 : -1 * transform.forward
- 자신의 왼쪽 : -1 * transform.right
- 자신의 아래쪽 : -1 * transform.up

10.3 벡터 연산으로 평행이동 구현하기

- 10.3.3 벡터 연산으로 평행이동
  - 벡터 연산을 응용하면 Translate( ) 메서드를 사용하지 않고 Transform의 position 값을 직접 수정하여 평행이동을 구현할 수 있음.

자신의 앞쪽으로 평행이동

게임 오브젝트가 지역 공간을 기준으로 (0, 1, 0)만큼 평행이동하는 코드는 다음과 같습니다.

```
transform.Translate(new Vector3(0, 1, 0));
```

같은 동작을 다음과 같이 표현할 수 있습니다.

```
transform.position = transform.position + transform.up * 1;
```

전역 공간 앞쪽으로 평행이동

게임 오브젝트가 전역 공간을 기준으로 (0, 1, 0)만큼 평행이동하는 코드는 다음과 같습니다.

```
transform.Translate(new Vector3(0, 1, 0), Space.World);
```

같은 동작을 다음과 같이 표현할 수 있습니다.

```
transform.position = transform.position + Vector3.up * 1;
```

10.4 마치며

이 장에서 배운 내용 요약

- 전역 공간은 게임 월드를 기준으로 함.
- 지역 공간은 부모 게임 오브젝트를 기준으로 함.
- 오브젝트 공간은 자기 자신을 기준으로 함.
- 인스펙터 창에 표시되는 위치, 회전, 스케일은 지역 공간 기준으로 측정된 값임.
- Transform의 Translate( ) 메서드로 평행이동함.
- Transform의 Rotate( ) 메서드로 회전함.
- Translate( ), Rotate( )는 기본적으로 지역 공간 기준으로 동작함.
- Translate( ), Rotate( )에 Space 타입을 입력하여 기준 공간을 지역 공간이나 전역 공간으로 결정할 수 있음.
- Vector3.up 등의 속기를 이용하면 자주 사용되는 Vector3 값을 즉시 생성할 수 있음.
- Transform 타입이 제공하는 방향 관련 변수 (transform.forward 등)로 게임 오브젝트의 방향을 쉽게 알 수 있음.