게임엔진프로그래밍응용

5. 탄막 슈팅 게임 2

청강문화산업대학교 게임콘텐츠스쿨

반 경 진

코로나 유의사항

코로나 유의 사항

- 2023학년도 1학기 방역 및 학사운영 방안
  https://www.ck.ac.kr/archives/193175
- 2023학년도 1학기 국가공휴일 및 대학 행사 수업 대체 일정 공지
  https://www.ck.ac.kr/archives/193109

온라인 수업 저작권 유의 사항

온라인 수업 저작권 유의 사항

온라인수업 저작권 유의사항 안내

강의 저작물을 다운로드, 캡처하여
교외로 유출하는 행위는
불 법 입 니 다

저작권자의 허락 없이 저작물을 복제, 공중송신 또는 배포하는 것은
저작권 침해에 해당하며 저작권법에 처벌받을 수 있습니다.

강의 동영상과 자료 일체는 교수 및 학교의 저작물로서 저작권이 보호됩니다.
수업자료를 무단 복제 또는 배포, 전송 시 민형사상 책임을 질 수 있습니다.

Index

확장 메서드

확장 메서드

- **확장 메서드**

특수한 종류의 static 메서드, 마치 다른 클래스의 매서드인 것처럼 호출해 사용할 수 있다.

1. 확장 메서드는 static 클래스안에 static 메서드로 정의한다.

2. 확장 메서드의 첫번째 매개 변수가 바로 그 다른 클래스의 메서드인 것처럼 호출할 수 있는 그 호출의 주체로 정의한다.(첫번째 매개변수 앞에 this를 써준다)

확장 메서드

```
using System;

namespace ExtensionMethods
{
    public static class MyExtensions
    {
        public static int WordCount(this string str)
        {
            return str.Split(
                new char[] { ' ', '.', '?' },
                StringSplitOptions.RemoveEmptyEntries)
                .Length;
        }
    }
}
```

확장 메서드

```
using ExtensionMethods;
using UnityEngine;

public class Examples : MonoBehaviour
{
    void Start()
    {
        string s = "Hello Extension Methods";
        Debug.Log(s.WordCount());

        int i = MyExtensions.WordCount(s);
        Debug.Log(i);

    }
}
```

확장 메서드

```
using UnityEngine;

public class Util
{
    public static T GetOrAddComponent<T>(GameObject go) where T : Component
    {
        T component = go.GetComponent<T>();
        if (component == null)
        {
            component = go.AddComponent<T>();
        }

        return component;
    }
}
```

확장 메서드

```
using UnityEngine;

public static class Extension
{
    public static T GetOrAddComponent<T>(this GameObject go) where T : UnityEngine.Component
    {
        return Util.GetOrAddComponent<T>(go);
    }
}
```

확장 메서드

```
using ExtensionMethods;
//using Unity.VisualScripting;
using UnityEngine;

public class Examples : MonoBehaviour
{
    void Start()
    {
        string s = "Hello Extension Methods";
        Debug.Log(s.WordCount());

        int i = MyExtensions.WordCount(s);
        Debug.Log(i);

        AudioSource audioSource = gameObject.GetOrAddComponent<AudioSource>();
        audioSource = Util.GetOrAddComponent<AudioSource>(gameObject);
        //audioSource = transform.GetOrAddComponent<AudioSource>();
    }
}
```

유니티 디자인 패턴

유니티 디자인 패턴

- **컴포넌트 패턴**

  다양한 기능의 로직을 필요에 따라 분리하여 재사용 가능한 개별적인 기능의 컴포넌트로 만드는 패턴

유니티 디자인 패턴

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class Moveable : MonoBehaviour
{
    public float speed;
    private Rigidbody rb;

    private void Start()
    {
        rb = gameObject.GetOrAddComponent<Rigidbody>();

        rb.velocity = transform.forward * speed;
    }
}
```

유니티 디자인 패턴

```
using UnityEngine;

public class SelfDestructable : MonoBehaviour
{
    public float lifeTime;

    private void Start()
    {
        Destroy(this.gameObject, lifeTime);
    }
}
```

유니티 디자인 패턴

```
using UnityEngine.Events;
using UnityEngine;

[System.Serializable]
public class TriggerEvent : UnityEvent<Collider>
{
}
```

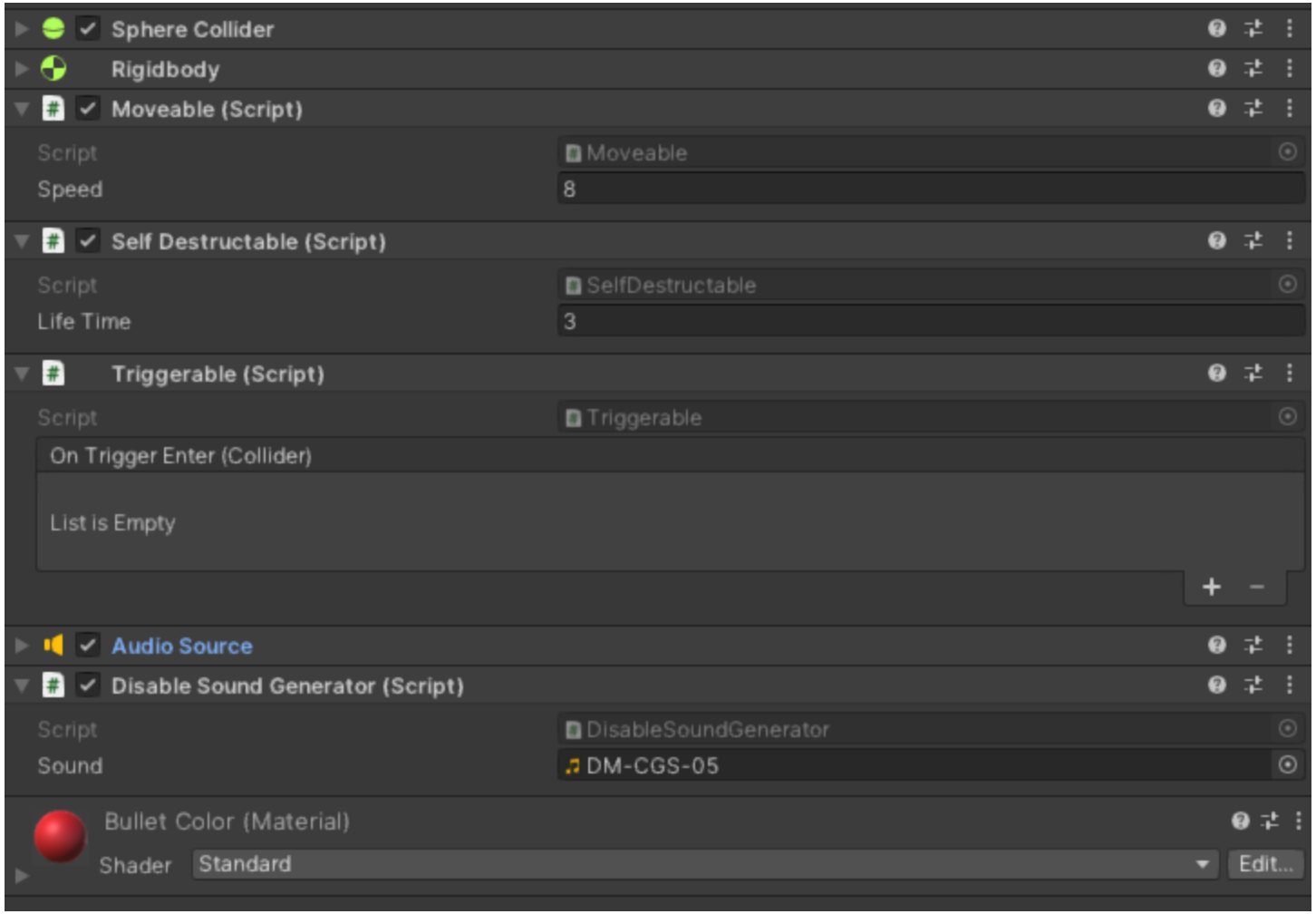
유니티 디자인 패턴

```
public class Triggerable : MonoBehaviour
{
    private Rigidbody rb;
    public TriggerEvent onTriggerEnter;

    private void Awake()
    {
        rb = gameObject.GetOrAddComponent<Rigidbody>();
        gameObject.GetOrAddComponent<Collider>().isTrigger = true;
    }

    private void OnTriggerEnter(Collider other)
    {
        onTriggerEnter.Invoke(other);
    }

    private void OnDestroy()
    {
        onTriggerEnter.RemoveAllListeners();
    }
}
```

유니티 디자인 패턴

```
using UnityEngine;

[RequireComponent(typeof(AudioSource))]
public class DisableSoundGenerator : MonoBehaviour
{
    public AudioClip sound;
    private AudioSource audioSource;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    private void OnDisable()
    {
        audioSource.clip = sound;
        audioSource.Play();
    }
}
```

유니티 디자인 패턴


Sphere Collider
Rigidbody
Moveable (Script)
Script
Moveable
Speed
8
Self Destructable (Script)
Script
SelfDestructable
Life Time
3
Triggerable (Script)
Script
Triggerable
On Trigger Enter (Collider)
List is Empty
Audio Source
Disable Sound Generator (Script)
Script
DisableSoundGenerator
Sound
DM-CGS-05
Bullet Color (Material)
Shader
Standard
Edit...

유니티 디자인 패턴

- **싱글턴 패턴**

다양한 기능의 로직을 필요에 따라 분리하여 재사용 가능한 개별적인 기능의 컴포넌트로 만드는 패턴

싱글턴 클래스
Singleton.cs

```
선언
public class GameManager : Singleton<GameManager>
{

사용
public void Die()
{
        gameObject.SetActive(false);
        GameManager.Instance.EndGame();
}
```

유니티 디자인 패턴

```
using UnityEngine;

[RequireComponent(typeof(AudioSource))]
public class AudioManager : MonoBehaviour
{
    public static AudioManager Instance;
    private AudioSource audioSource;

    private void Awake()
    {
        Instance = this;
        audioSource = GetComponent<AudioSource>();
    }

    public void PlaySound(AudioClip clip)
    {
        Debug.Log("<color=yellow>Sound Play!!</color> : " + clip.name);
        audioSource.PlayOneShot(clip);
    }
}
```

유니티 디자인 패턴

```
using UnityEngine;

public class DisableSoundGenerator : MonoBehaviour
{
    public AudioClip sound;

    private void OnDisable()
    {
        AudioManager.Instance.PlaySound(sound);
    }
}
```

유니티 디자인 패턴

- **팩토리 패턴**

  객체를 생성하고자 할 때 사용하는 패턴 (Factory - 객체 생성을 처리하는 클래스)

  Simple Factory Pattern

  주어진 입력을 기반으로 다른 유형의 객체를 리턴하는 메소드가 있는 팩토리 클래스

  Factory Method Pattern

  어떤식으로 생성할지에 관한 생성의 형태는 팩토리 클래스를 상속하는 서브 팩토리 클래스들에게 맡긴다.
  생성 형태는 여러가지가 있으므로 서브 팩토리 클래스들은 여러 개가 있을 수 있다.

  Abstract Factory Pattern

  관련 있는 여러 종류의 객체를 특정 그룹으로 묶어 한번에 생성

유니티 디자인 패턴 (Simple Factory Pattern)

```
using UnityEngine;

public abstract class SpawnerBase : MonoBehaviour
{
    public abstract void Spawn();
}
```

유니티 디자인 패턴 (Simple Factory Pattern)

```
using UnityEngine;

public class CubeSpawner : SpawnerBase
{
    public override void Spawn()
    {
        Debug.Log("CubeSpawner.Spawn");
        GameObject obj = GameObject.CreatePrimitive(PrimitiveType.Cube);
    }
}
```

유니티 디자인 패턴 (Simple Factory Pattern)

```
using UnityEngine;

public class CapsuleSpawner : SpawnerBase
{
    public override void Spawn()
    {
        Debug.Log("CapsuleSpawner.Spawn");
        GameObject obj = GameObject.CreatePrimitive(PrimitiveType.Capsule);
    }
}
```

유니티 디자인 패턴 (Simple Factory Pattern)

```
using UnityEngine;

public class SphereSpawner : SpawnerBase
{
    public override void Spawn()
    {
        Debug.Log("SphereSpawner.Spawn");
        GameObject obj = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    }
}
```

유니티 디자인 패턴 (Simple Factory Pattern)

```
using UnityEngine;

public enum SpawnerType
{
    cube,
    capsule,
    sphere
}
```

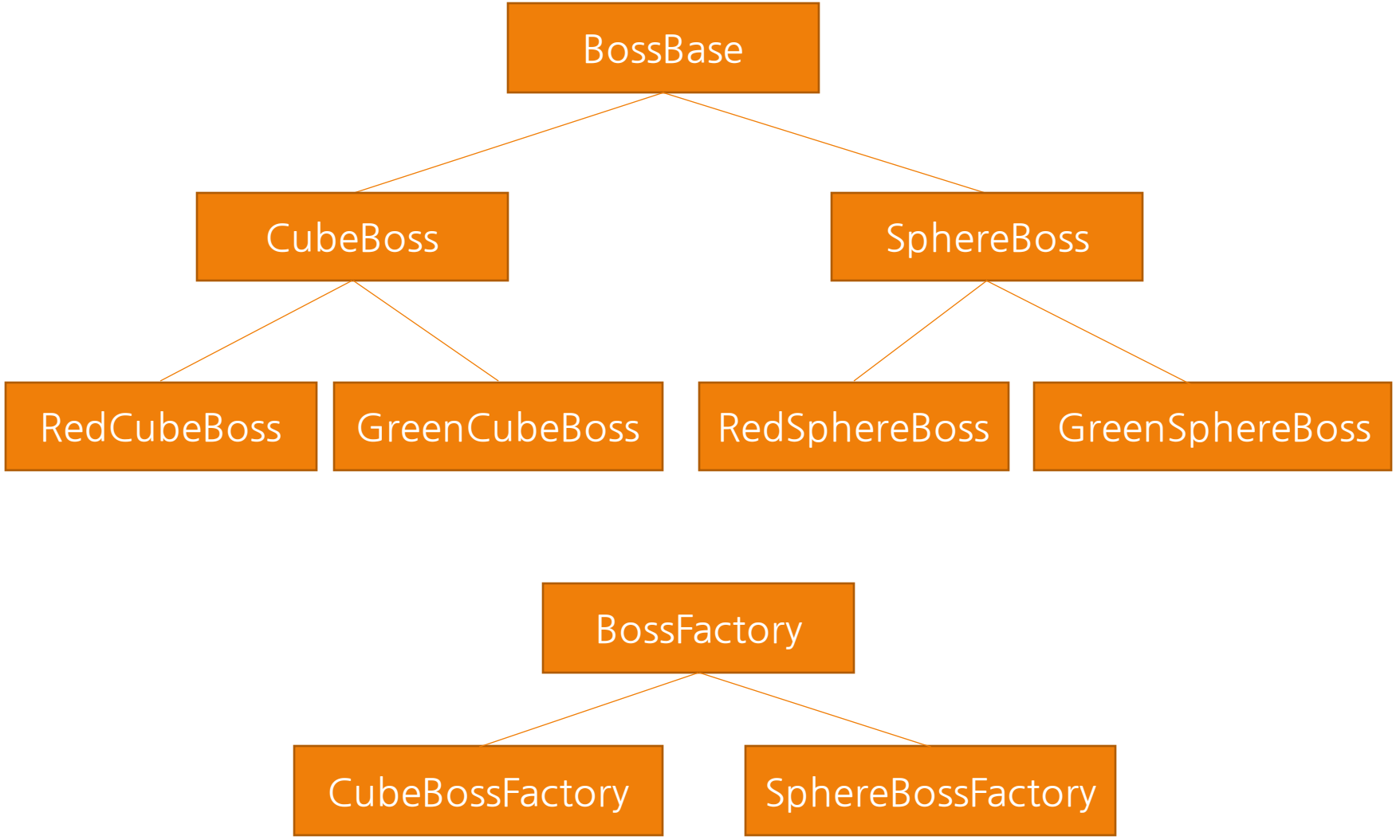
유니티 디자인 패턴 (Simple Factory Pattern)

```
public class SpwanerFactory : MonoBehaviour
{
    public static SpawnerBase createSpwaner(SpawnerType type)
    {
        GameObject obj = null;
        SpawnerBase spawner = null;
        switch (type)
        {
            case SpawnerType.cube:
                obj = Instantiate(Resources.Load<GameObject>("CubeSpawner"));
                spawner = obj.GetComponent<CubeSpawner>();
                break;

            case SpawnerType.capsule:
                obj = Instantiate(Resources.Load<GameObject>("CapsuleSpawner"));
                spawner = obj.GetComponent<CapsuleSpawner>();
                break;

            case SpawnerType.sphere:
                obj = Instantiate(Resources.Load<GameObject>("SphereSpawner"));
                spawner = obj.GetComponent<SphereSpawner>();
                break;
        }
        return spawner;
    }
}
```

유니티 디자인 패턴 (Simple Factory Pattern)

```
public void Die(Collider other)
{
    if (other.tag != "Player")
    {
        return;
    }

    // 자신의 게임 오브젝트를 비활성화
    gameObject.SetActive(false);

    SpwanerFactory.createSpwaner(
        (SpawnerType)Random.Range((int)SpawnerType.cube,
        (int)SpawnerType.sphere))
        .Spawn();

    GameManager.Instance.EndGame();
}
```

유니티 디자인 패턴 (Factory Method Pattern)



유니티 디자인 패턴 (Factory Method Pattern)

```
using UnityEngine;

public class BossBase : MonoBehaviour
{

}
```

유니티 디자인 패턴 (Factory Method Pattern)

```
public class CubeBoss : BossBase
{

}

public class RedCubeBoss : CubeBoss
{

}

public class GreenCubeBoss : CubeBoss
{

}
```

유니티 디자인 패턴 (Factory Method Pattern)

```
public class SphereBoss : BossBase
{

}

public class RedSphereBoss : SphereBoss
{

}

public class GreenSphereBoss : SphereBoss
{

}
```

유니티 디자인 패턴 (Factory Method Pattern)

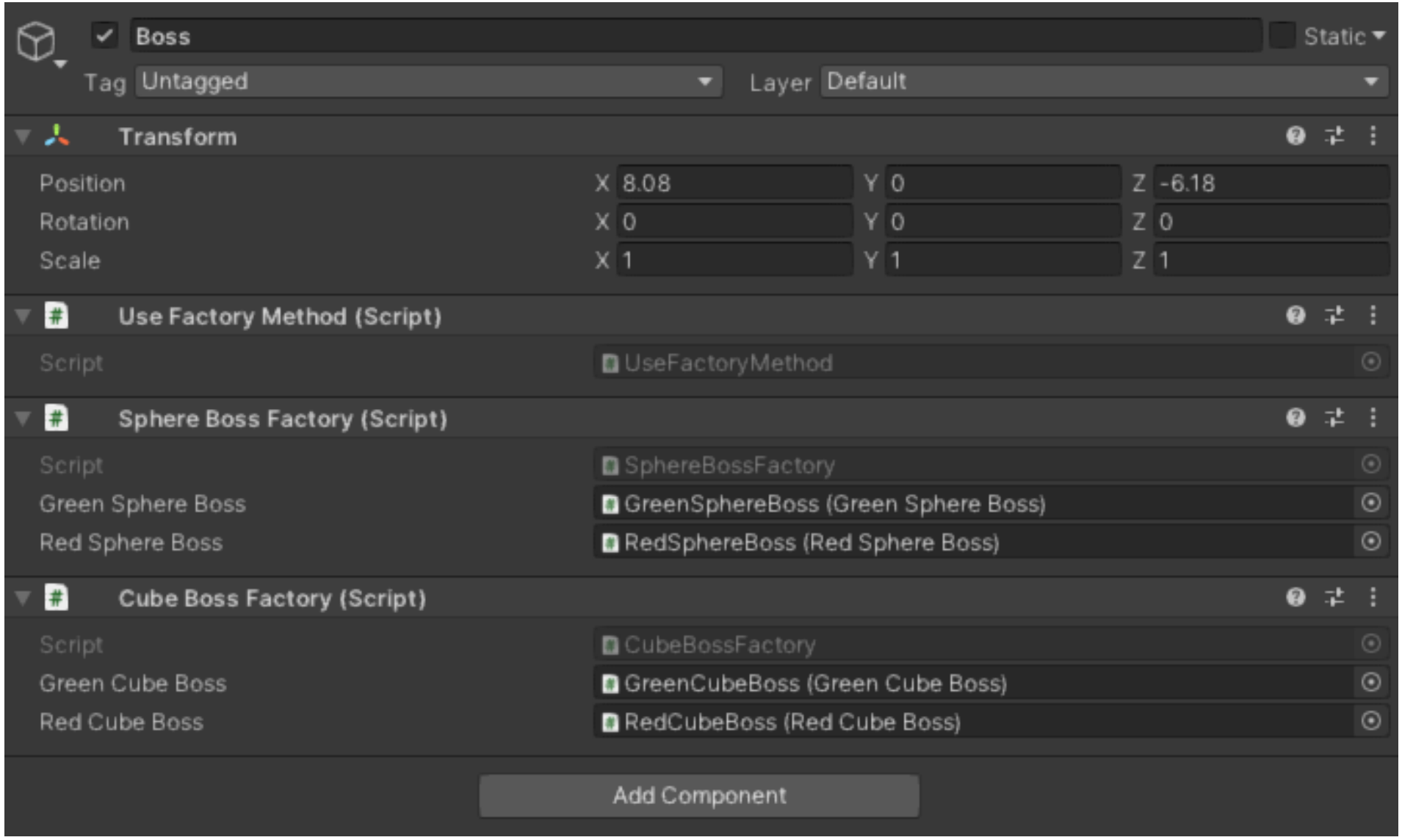
```
using UnityEngine;

public abstract class BossFactory : MonoBehaviour
{
    public abstract BossBase CreateBoss(string type);
}
```

```
public class CubeBossFactory : BossFactory
{
    public GreenCubeBoss greenCubeBoss;
    public RedCubeBoss redCubeBoss;

    public override BossBase CreateBoss(string type)
    {
        BossBase boss = null;
        if (type.Equals("green"))
        {
            boss = Instantiate(greenCubeBoss);
        }
        else if (type.Equals("red"))
        {
            boss = Instantiate(redCubeBoss);
        }

        return boss;
    }
}
```

유니티 디자인 패턴 (Factory Method Pattern)

```
public class SphereBossFactory : BossFactory
{
    public GreenSphereBoss greenSphereBoss;
    public RedSphereBoss redSphereBoss;
    public override BossBase CreateBoss(string type)
    {
        BossBase boss = null;

        if (type.Equals("green"))
        {
            boss = Instantiate(greenSphereBoss);
        }
        else if (type.Equals("red"))
        {
            boss = Instantiate(redSphereBoss);
        }

        return boss;
    }
}
```

유니티 디자인 패턴 (Factory Method Pattern)



유니티 디자인 패턴 (Factory Method Pattern)

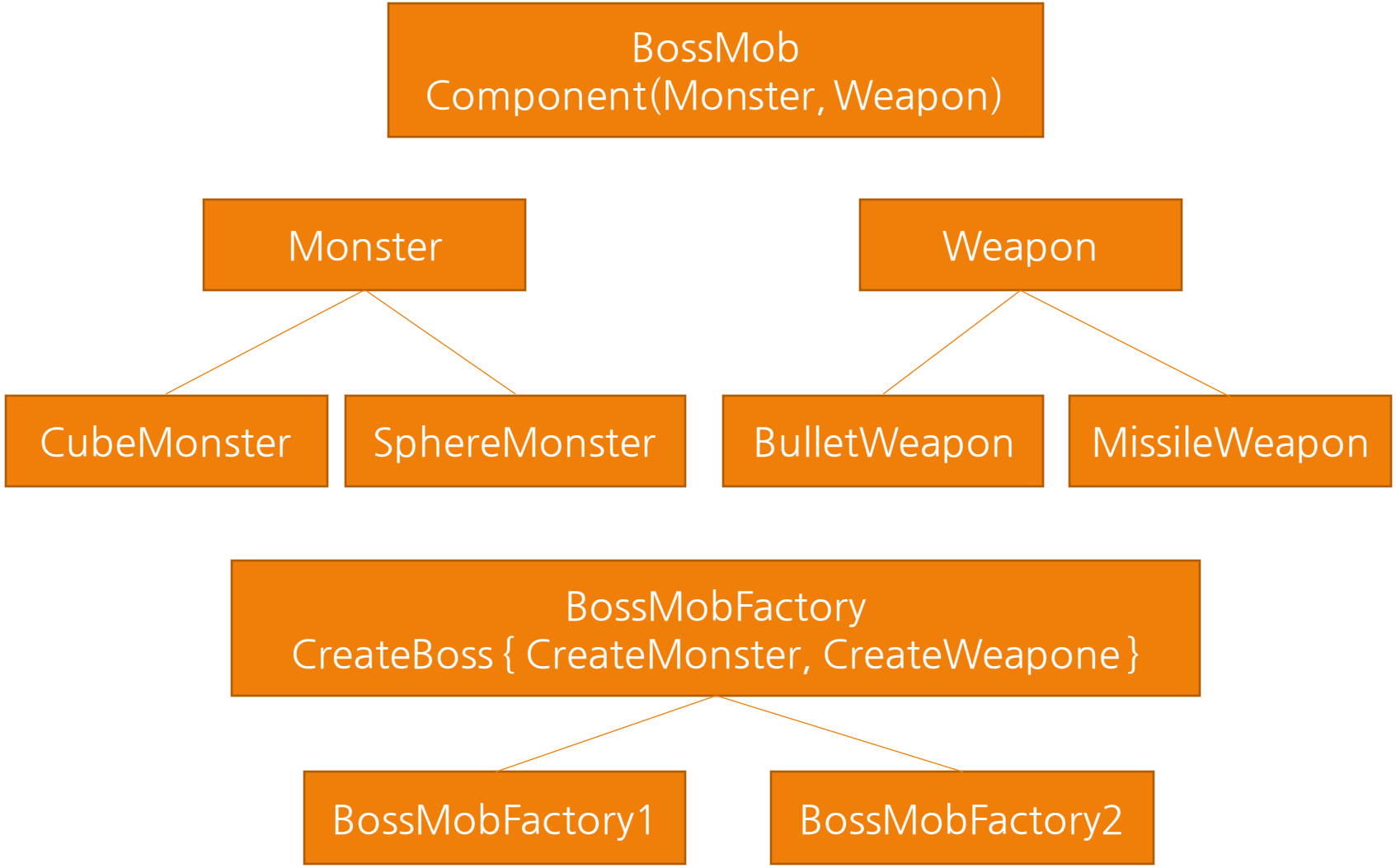
```
using UnityEngine;

public class UseFactoryMethod : MonoBehaviour
{
    public void EventTime()
    {
        BossFactory bf = GetComponent<CubeBossFactory>();
        BossBase boss = bf.CreateBoss("green");
        boss.transform.position = transform.position;
        boss.transform.LookAt(Vector3.zero);
    }
}
```

유니티 디자인 패턴 (Factory Method Pattern)

GameManager.cs

```
[SerializeField] private UseFactoryMethod factoryMethod;
private float eventTime = 10f;
private bool IsEventTime = false;

void Update()
{
    // 게임 오버가 아닌 동안
    if (!isGameover)
    {
        // 생존 시간 갱신
        surviveTime += Time.deltaTime;
        // 갱신한 생존 시간을 timeText 텍스트 컴포넌트를 통해 표시
        timeText.text = "Time: " + (int) surviveTime;

        if (surviveTime > eventTime && !IsEventTime)
        {
            IsEventTime = true;
            factoryMethod.EventTime();
        }
    }
```

유니티 디자인 패턴 (Abstract Factory Pattern)



유니티 디자인 패턴 (Abstract Factory Pattern)

```
using UnityEngine;

public class BossMob : MonoBehaviour
{
    public Monster monster;
    public Weapon weapon;
}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
using UnityEngine;

public class Monster : MonoBehaviour
{
}
```

```
using UnityEngine;

public class Weapon : MonoBehaviour
{
}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
public class CubeMonster : Monster
{

}

public class SphereMonster : Monster
{

}

public class BulletWeapon : Weapon
{

}

public class MissileWeapon : Weapon
{

}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
using UnityEngine;

public abstract class BossMobFactory : MonoBehaviour
{
    public BossMob CreateBoss()
    {
        BossMob boss = new BossMob
        {
            monster = CreateMonster(),
            weapon = CreateWeapon()
        };

        return boss;
    }

    public abstract Monster CreateMonster();
    public abstract Weapon CreateWeapon();
}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
public class BossMobFactory1 : BossMobFactory
{
    public override Monster CreateMonster()
    {
        Monster monster = new CubeMonster();
        return monster;
    }

    public override Weapon CreateWeapon()
    {
        Weapon weapon = new MissileWeapon();
        return weapon;
    }
}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
public class BossMobFactory2 : BossMobFactory
{
    public override Monster CreateMonster()
    {
        Monster monster = new SphereMonster();
        return monster;
    }

    public override Weapon CreateWeapon()
    {
        Weapon weapon = new BulletWeapon();
        return weapon;
    }
}
```

유니티 디자인 패턴 (Abstract Factory Pattern)

```
using UnityEngine;

public class BattleGenerator : MonoBehaviour
{
    void Start()
    {
        BossMobFactory factory1 = new BossMobFactory1();
        BossMob boss1 = factory1.CreateBoss();
    }
}
```

유니티 디자인 패턴

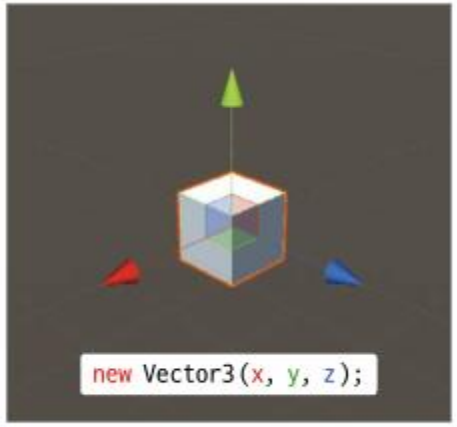
- 메멘토 패턴
- 옵저버 패턴
- 커맨드 패턴
- 빌더 패턴
- 오브젝트 풀
- FSM과 스테이트 패턴
- 데코레이터 패턴

방향, 크기, 회전

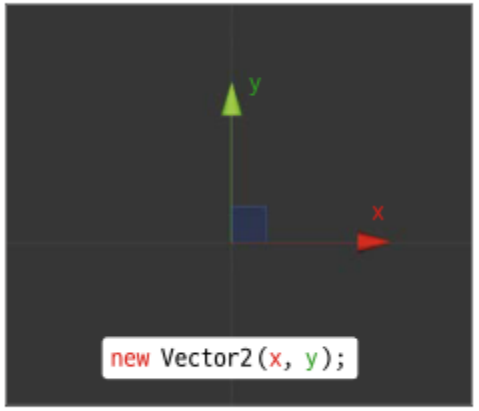
벡터(Vector)

게임 개발에서 벡터는 주로 위치, 방향, 속도(크기 또는 거리)를 나타내는데 사용

유니티는 Vector3를 사용하여 3D공간에서의 x, y, z 좌표를 표현



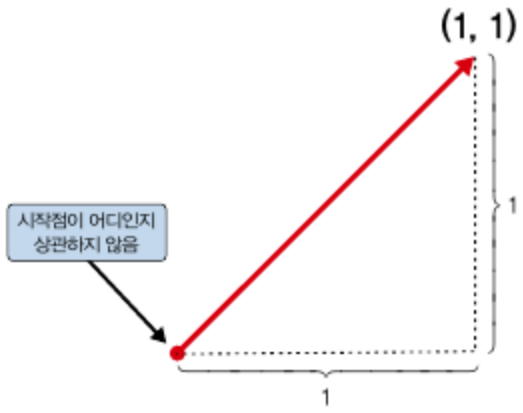
▶ Vector3는 3차원 공간에 대응



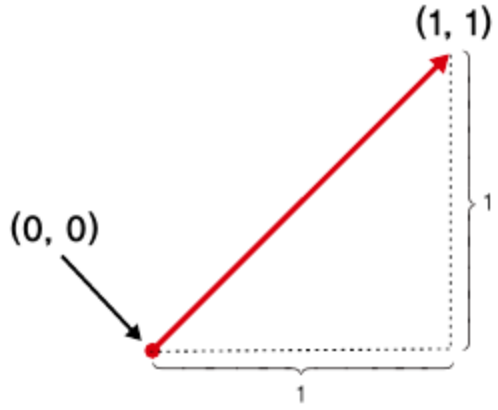
▶ Vector2는 2차원 공간에 대응

벡터(Vector)

절대 위치와 상대 위치



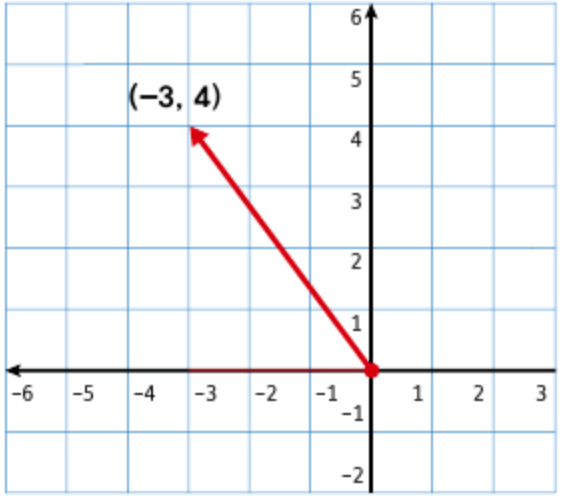
▶ 상대적인 방향과 크기로서의 (1, 1)



▶ 절대적인 좌표로서의 (1, 1)

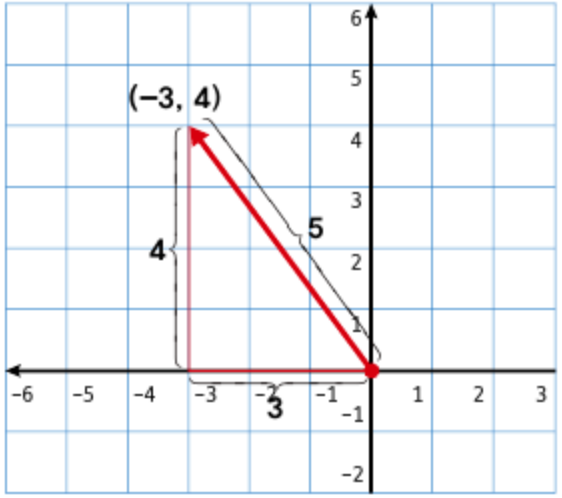
벡터(Vector)

벡터의 크기

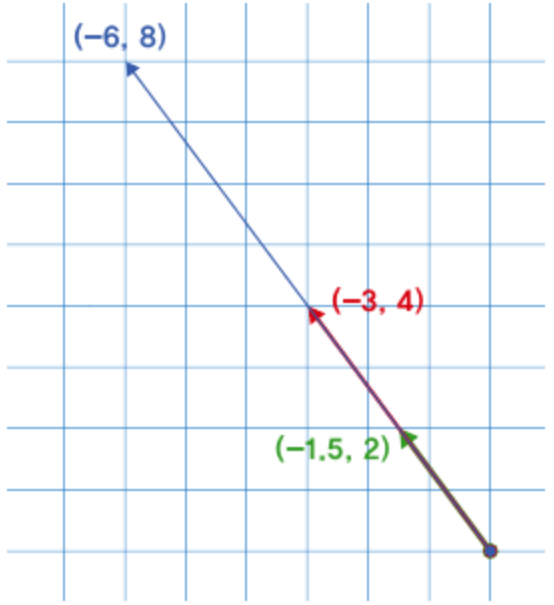


▶ (−3, 4)를 나타낸 벡터

$(-3, 4)$의 크기 $= \sqrt{(-3)^2 + 4^2} = 5$



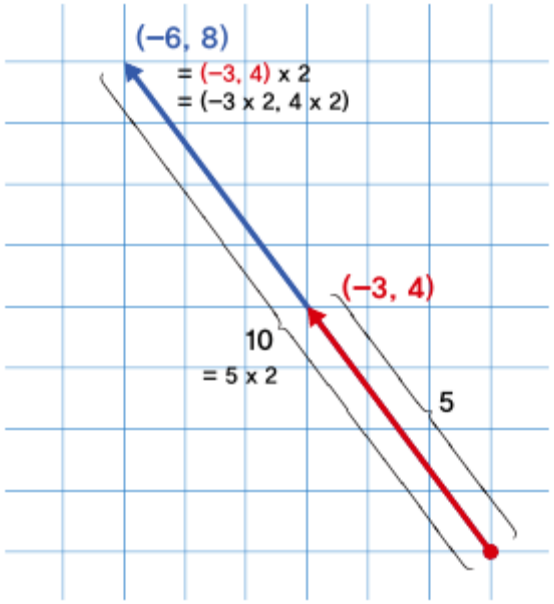
▶ Vector2 (−3, 4)의 크기



▶ (−3, 4)와 방향은 같지만 크기는 다른 벡터들

벡터(Vector)

벡터의 스칼라곱

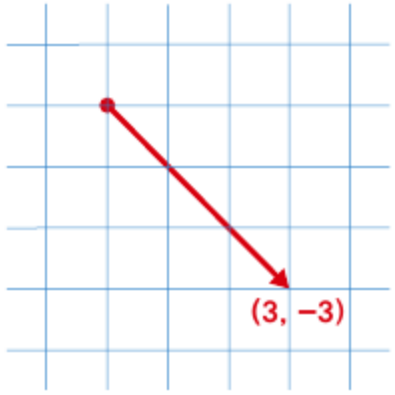


▶ 벡터에 배수 취하기

벡터(Vector)

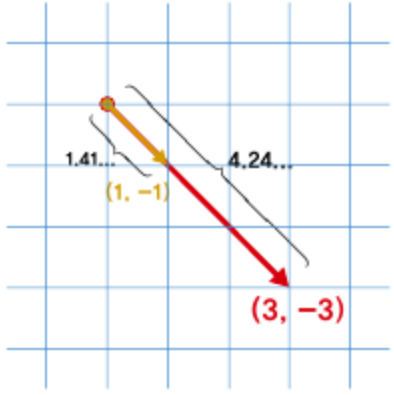
방향 벡터

벡터를 속도라고 해석하면 벡터의 화살표 방향은 이동하려는 방향, 화살표의 길이는 속력(이동거리)
벡터의 정규화(방향을 유지하면서 벡터의 크기를 1로 만드는 것)



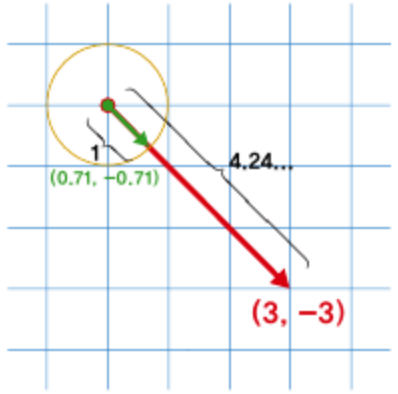
▶ 벡터 (3, −3)

(3, −3) = (방향) × (속력 또는 이동거리)



▶ (1, −1)의 3배에 해당하는 (3, −3)

(3, −3) = (1, −1) × 3



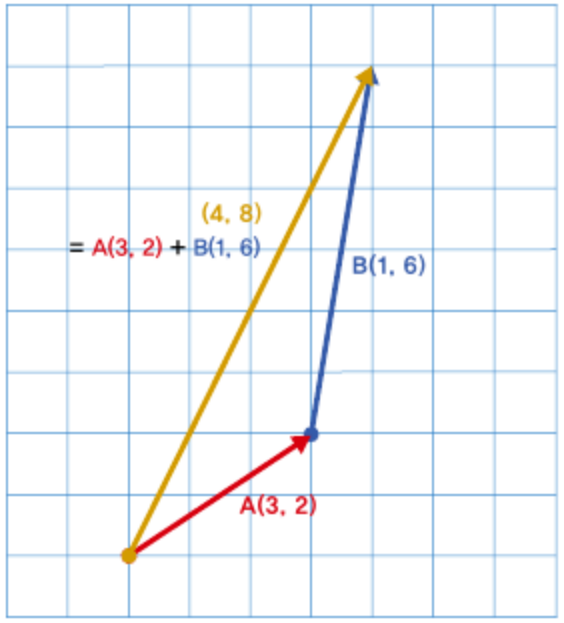
▶ 방향벡터 구하기

(3, −3) = (0.71, −0.71) × 4.24

벡터(Vector)

벡터의 덧셈

두 벡터 A와 B를 더하는 행위를 공간상에서 보면 A만큼 이동한 상태에서 B만큼 더 이동한다는 의미



▶ A + B는 A만큼 이동한 상태에서 B만큼 더 이동한 것

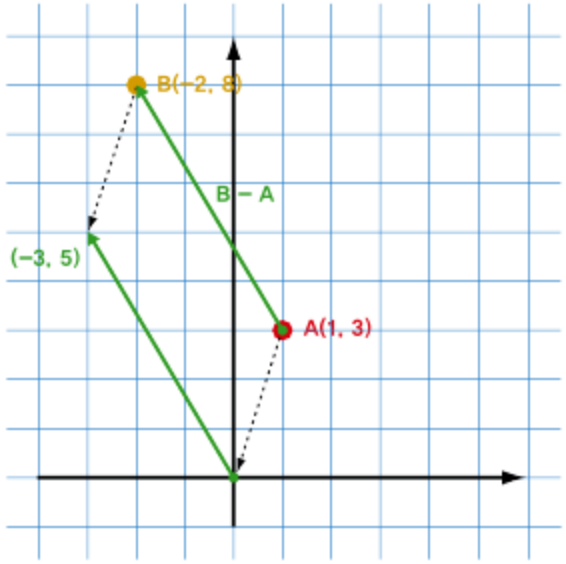
벡터(Vector)

벡터의 뺄셈

목적지를 B, 현재 위치를 A라고 할 때, B - A = 현재 위치에서 목적지까지의 방향과 거리
어떤 물체가 다른 물체를 추적할 때 어떤 방향으로 얼마만큼 가야 하는지 알 수 있음

벡터 A(1, 3)과 벡터 B(−2, 8)이 있다고 가정해봅시다. B에서 A를 빼면 같은 자리의 성분끼리 뺄셈이 됩니다.

$$B - A = (-2, 8) - (1, 3) = (-2 - 1, 8 - 3) = (-3, 5)$$

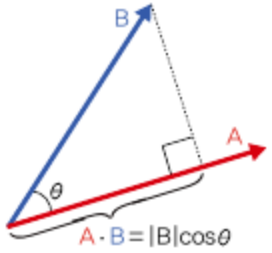


▶ 벡터의 뺄셈이 공간상에서 가지는 의미

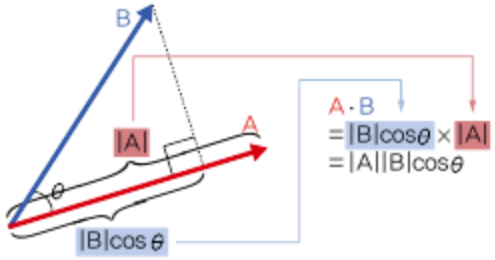
벡터(Vector)

벡터의 내적(dot product)

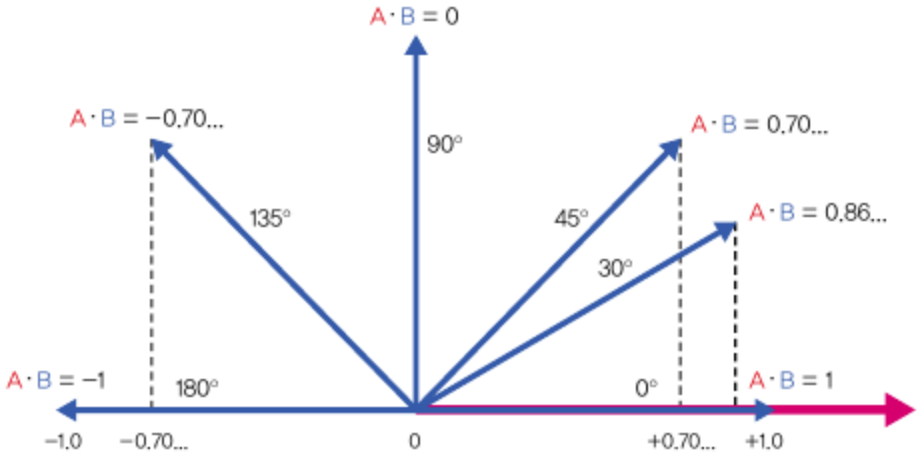
어떤 벡터 B를 다른 벡터 A로 투영.



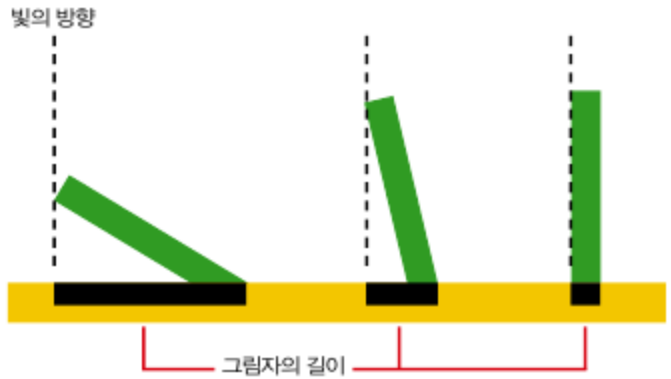
▶ A의 크기가 1인 경우 A · B를 구하는 공식



▶ A · B를 구하는 공식



▶ 여러 각도에 대한 대표적인 내적값



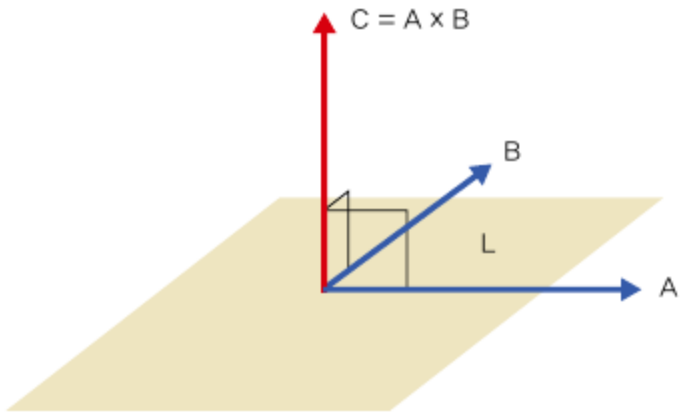
▶ 수직에 가깝게 막대기를 세울수록 그림자가 짧아진다

벡터(Vector)

벡터의 외적(cross product)

두 벡터 모두 수직으로 통과하는 벡터를 구하는 연산

두 벡터 가 평면에 포함되어 있으면 그 평면의 수직인 방향을 구할 수 있음(평면이 바라보는 방향을 구할 수 있음)
이 방향 벡터가 노말 벡터 또는 법선 벡터.

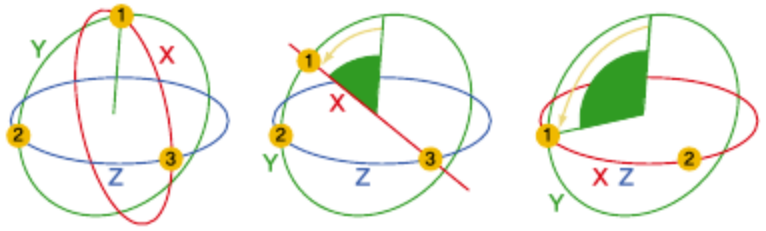


▶ A x B을 표현한 그림

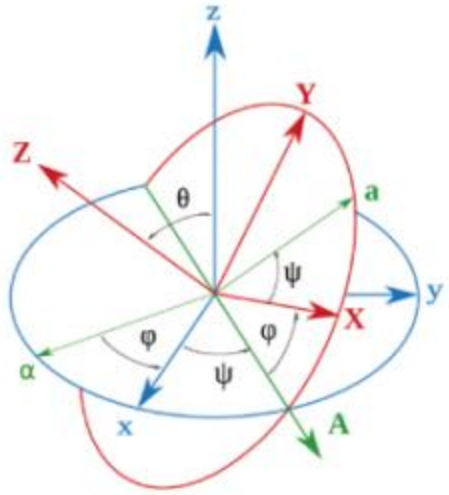
쿼터니언(Quaternion)

쿼터니언

- 회전을 나타내는 타입
- 짐벌락(Gimbal Lock)
- 오일러각



▸ 짐벌락의 예



▸ 오일러각의 표현

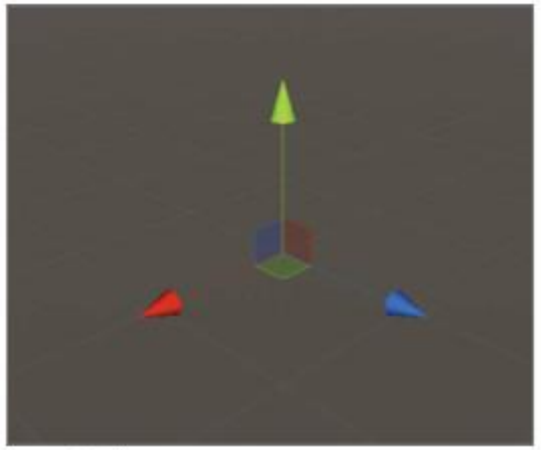
```
Quaternion a = Quaternion.Euler(30, 0, 0);
Quaternion b = Quaternion.Euler(0, 60, 0);

// a만큼 회전한 상태에서 b만큼 더 회전한 회전값을 표현
Quaternion rotation = a * b;
```

공간과 움직임

공간과 움직임

유니티 공간

- 전역 공간
- 오브젝트 공간
- 지역공간

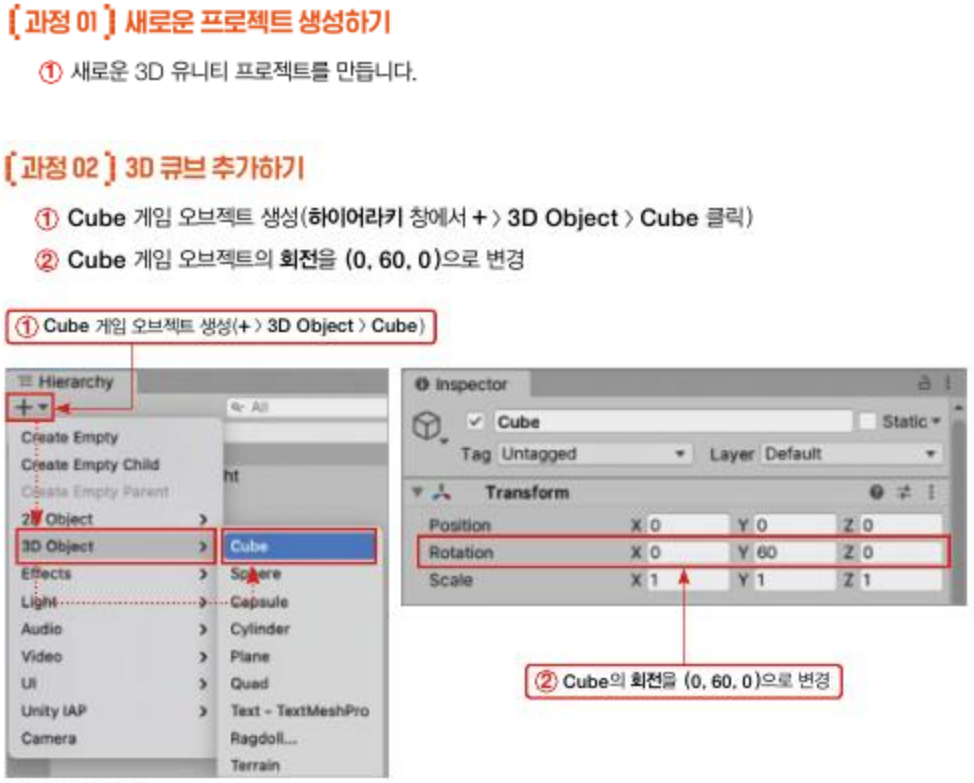


▶ 3D 좌표계

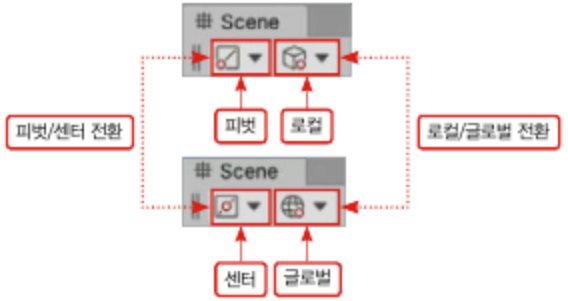
공간과 움직임

전역 공간

월드의 중심이라는 절대 기준이 존재하는 공간이며 월드 공간이라 부르기도 함.
전역 공간에서 방향을 정하고 좌표를 계산하는 기준을 전역 좌표계라고 한다.



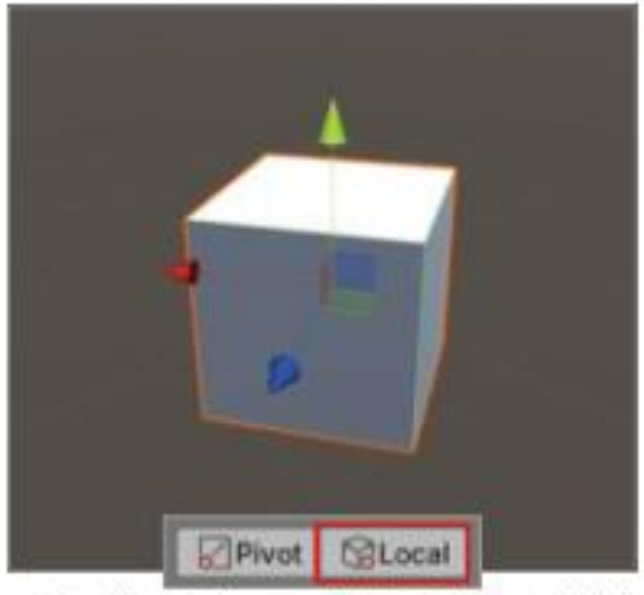
▶ 3D 큐브 추가

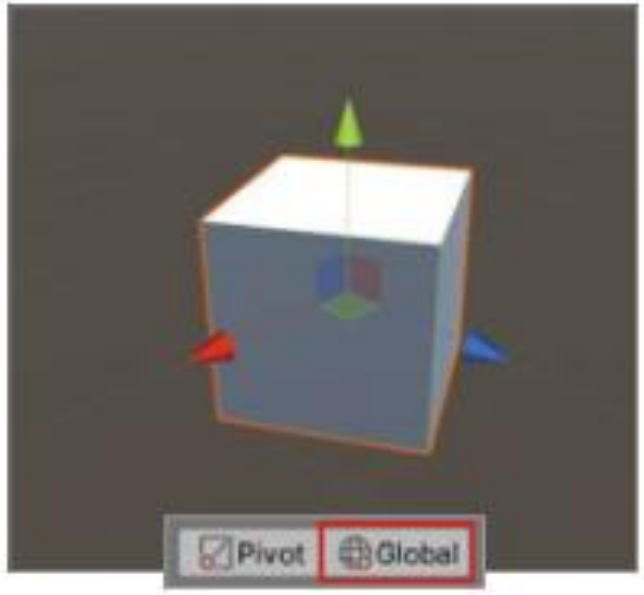


▶ 툴 핸들 토글 버튼

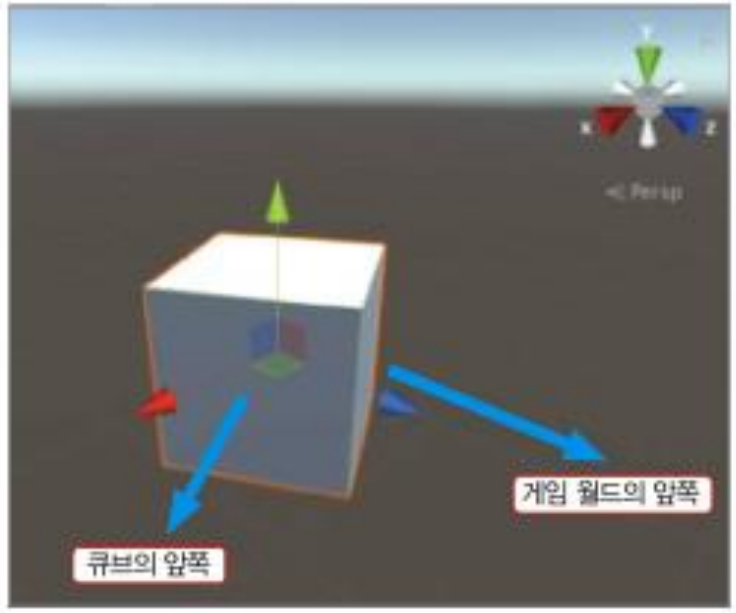
▶ 전역 공간 모드로 전환

공간과 움직임





▶ 오브젝트 공간 모드와 전역 공간 모드의 차이



▶ 전역 공간과 오브젝트 공간에서의 앞쪽

공간과 움직임

오브젝트 공간

오브젝트 자신의 방향을 배치 기준으로 사용

[과정 01] 오브젝트 공간 모드로 전환

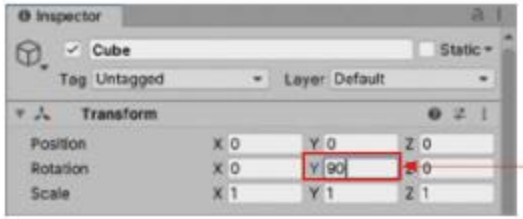
① Local/Global 전환 버튼 클릭 → Local 선택

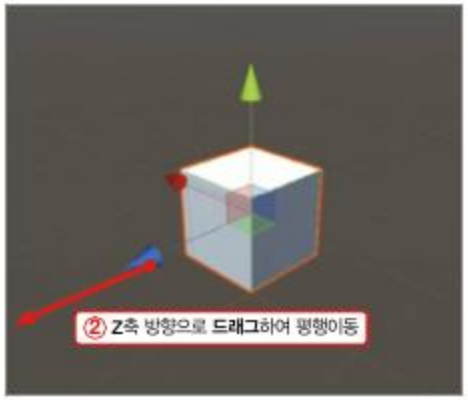
▶ 오브젝트 공간 모드로 전환

[과정 02] 큐브를 회전시키고 Z축으로 평행이동하기

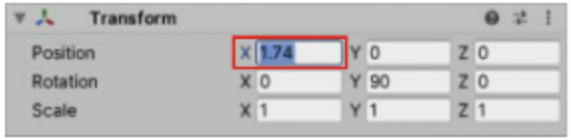
① Cube 게임 오브젝트의 **Y축** 회전을 **90**으로 변경

② **씬** 창에서 **Cube**에 표시된 **평행이동 도구**의 **Z축** 화살표를 누르고 **Z축** 방향으로 **드래그**





▶ 3D 큐브를 회전시키고 Z축으로 평행이동



▶ X 값이 변경됨

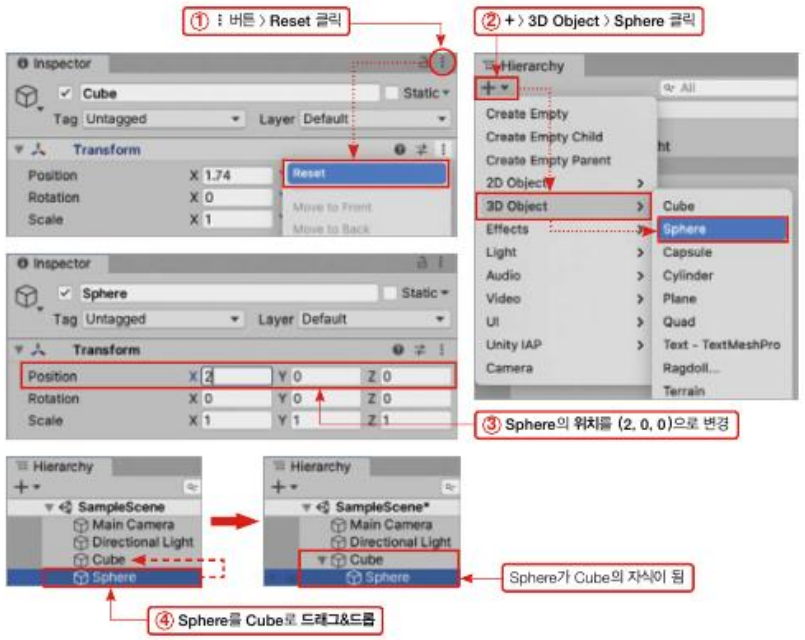
공간과 움직임

지역 공간

부모 오브젝트 존재하지 않으면 지역 좌표계와 전역 좌표계가 일치
부모 오브젝트를 기준으로 한 지역 좌표계

[과정 01] 큐브의 자식 만들기

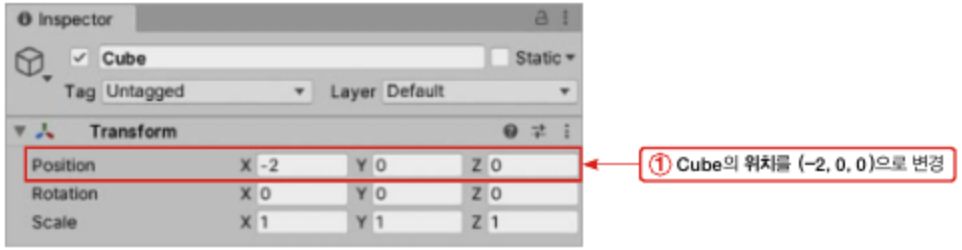
① **인스펙터** 창에서 **Cube** 게임 오브젝트의 **Transform** 컴포넌트의 ⋮ 버튼 > **Reset** 클릭
② **Sphere** 게임 오브젝트 생성(**하이어라키** 창에서 + > **3D Object** > **Sphere** 클릭)
③ **Sphere** 게임 오브젝트의 **위치**를 **(2, 0, 0)**으로 변경
④ **하이어라키** 창에서 **Sphere**를 **Cube**로 **드래그&드롭**



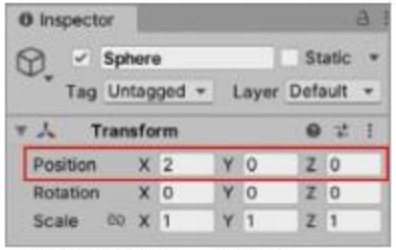
▶ 큐브의 자식 오브젝트 만들기

[과정 02] 큐브 위치 옮기기

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



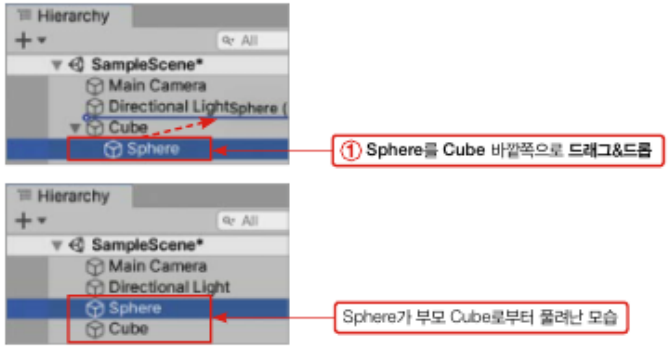
▶ 큐브 위치 옮기기

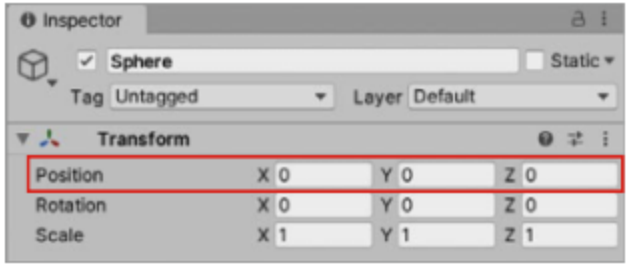


▶ 변경되지 않은 Sphere의 위치

[과정 03] 구를 부모 Cube로부터 풀어주기

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





▶ Sphere의 전역 위치

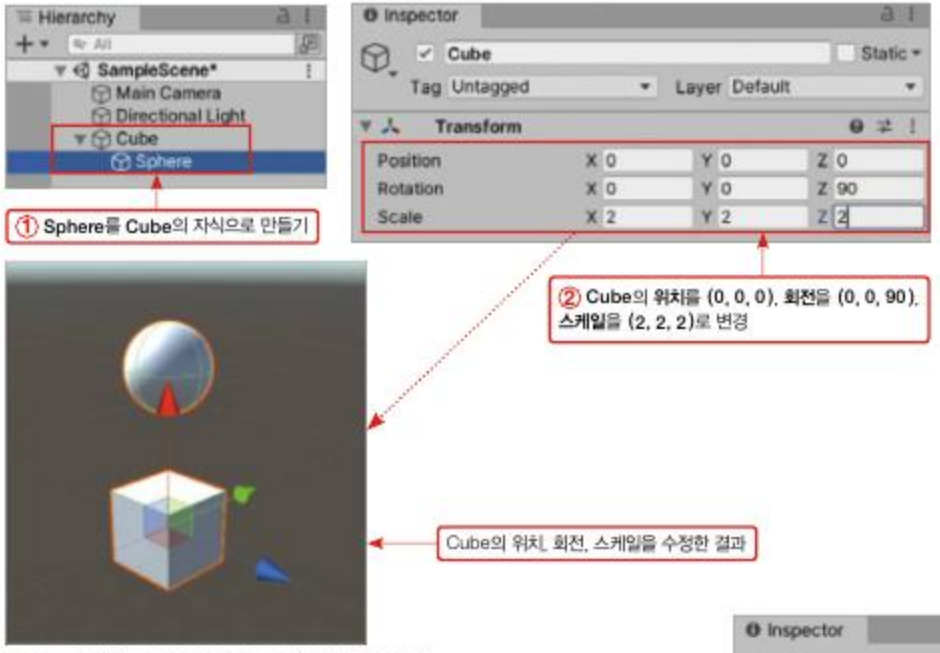
▶ Sphere를 자신의 부모 Cube로부터 풀어주기

공간과 움직임

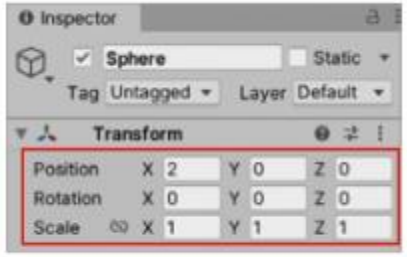
[과정 04] 부모에 의한 자식의 회전과 스케일 변경 확인하기

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



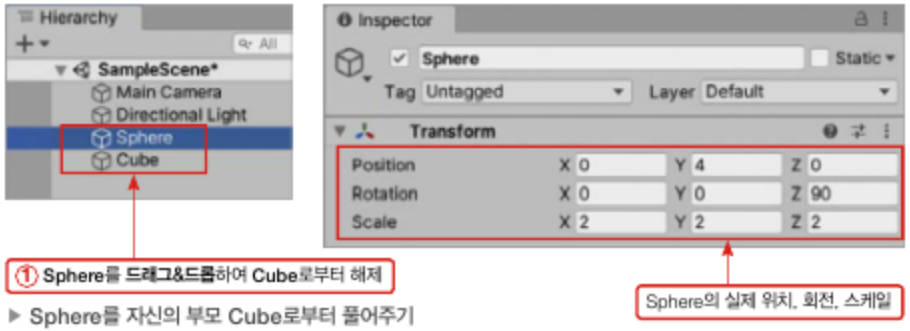
▶ 부모에 의한 자식의 회전과 스케일 변경 확인하기



▶ 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

공간과 움직임

오브젝트의 이동과 회전

```
using UnityEngine;

public class Move : MonoBehaviour
{
    public Transform childTranform;

    void Start()
    {
        transform.position = new Vector3(0, -1, 0);
        childTranform.localPosition = new Vector3(0, 2, 0);
        transform.rotation = Quaternion.Euler(new Vector3(0, 0, 30));
        childTranform.localRotation = Quaternion.Euler(new Vector3(0, 60, 0));
    }

    void Update()
    {
        float xInput = Input.GetAxis("Horizontal");
        float zInput = Input.GetAxis("Vertical");
```

공간과 움직임

```
if (zInput > 0)
{
    // 초당 1만큼 이동
    // 지역 공간
    transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime);
    //transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime, Space.Self);
    // 전역 공간
    //transform.Translate(new Vector3(0, 1, 0) * Time.deltaTime, Space.World);
}
else if (zInput < 0)
{
    transform.Translate(new Vector3(0, -1, 0) * Time.deltaTime);
}

if (xInput > 0)
{ // 초당 180도 회전
    transform.Rotate(new Vector3(0, 0, 180) * Time.deltaTime);

    childTranform.Rotate(new Vector3(0, 180, 0) * Time.deltaTime);
}
```

공간과 움직임

```
        else if (xInput < 0)
        {
            transform.Rotate(new Vector3(0, 0, -180) * Time.deltaTime);

            childTranform.Rotate(new Vector3(0, -180, 0) * Time.deltaTime);
        }
    }
}
```

공간과 움직임

오브젝트의 이동과 회전

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)`

Transform 타입의 방향

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