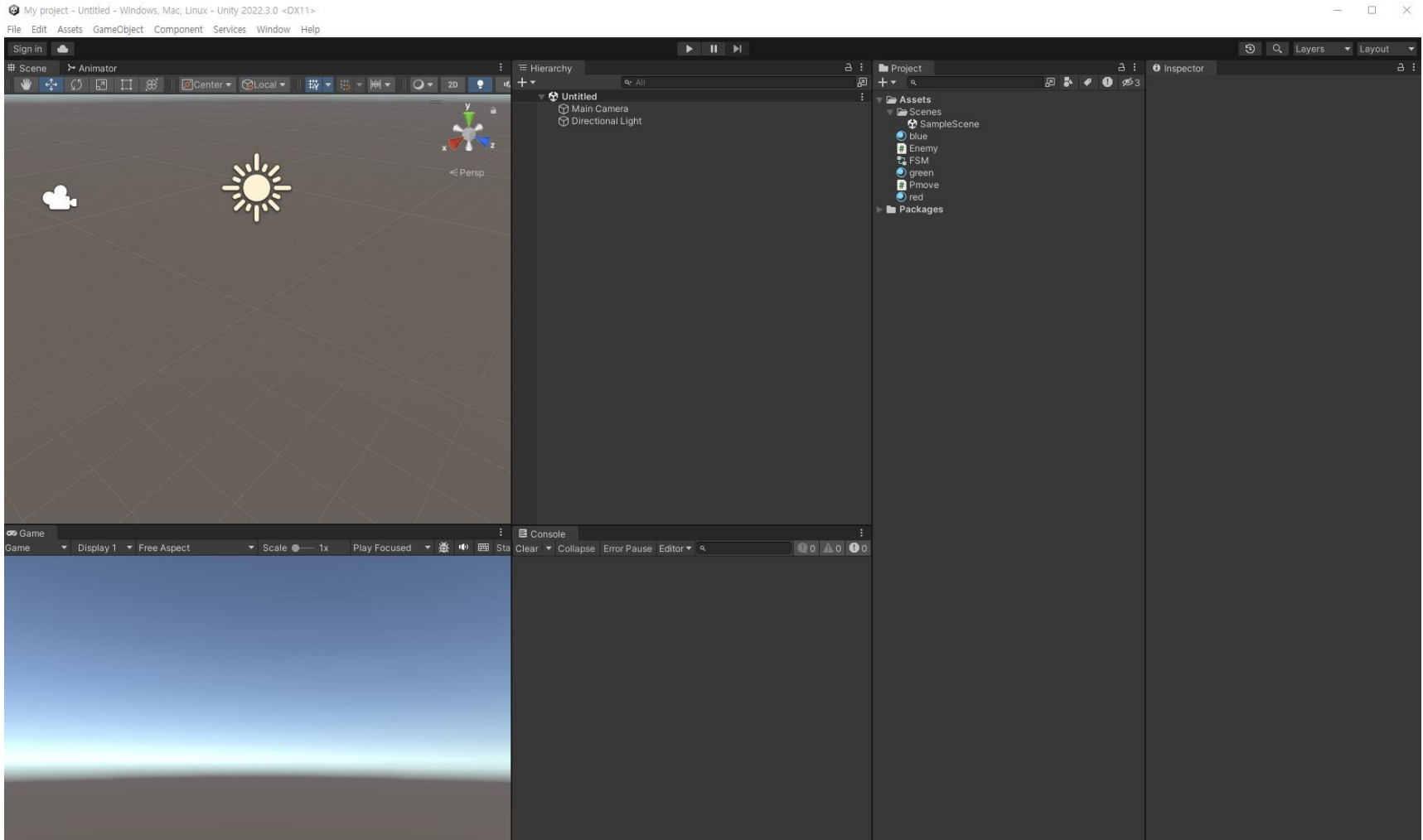


# 뷰(View) : 각 탭으로 분리된 윈도우



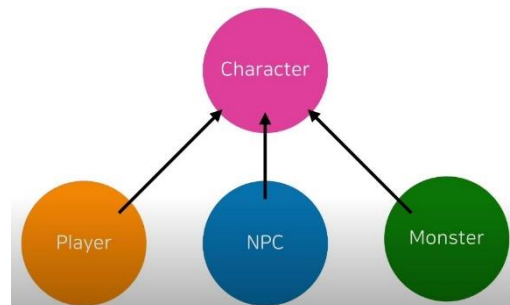
# 컴포넌트 기반의 개발

게임 오브젝트 (컴포넌트 들을 담는 그릇)



- 필요한 **컴포넌트**(기능 단위)들을 모아 **게임 오브젝트**(태그, 단순히 이름 역할)를 구성
- ✓ 컴포넌트 = 클래스

\* 상속이 항상 좋은 방법인 것은 아님



# 컴포넌트

## 컴포넌트의 구조

컴포넌트 필수 기능  
(MonoBehavior)

컴포넌트  
고유 기능

## MonoBehaviour

- 컴포넌트로서 게임 오브젝트에게 추가될 수 있다
- 유니티의 통제를 받는다
- 유니티 이벤트 메시지를 감지할 수 있게 된다



# Material

오브젝트에 텍스처를 적용하기 위한 매개체  
텍스처의 반복과 재질 등 표현

## Texture

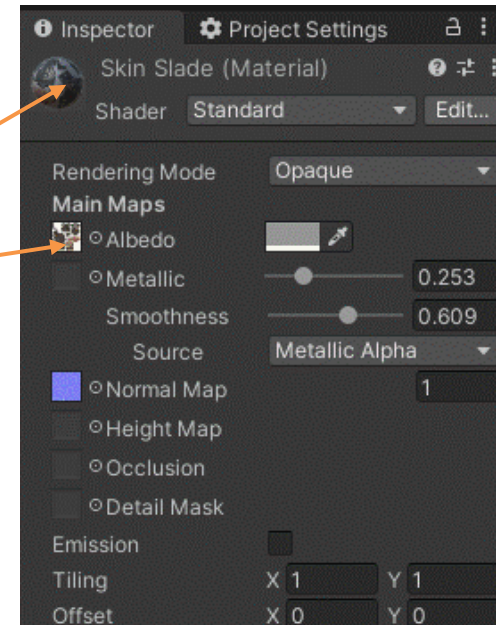
- 3D 모델의 표면에 그려지는 이미지 파일

## Shader

- Material에 적용한 텍스처를 렌더링할 때 재질감을 표현하는 방법
- CPU → GPU

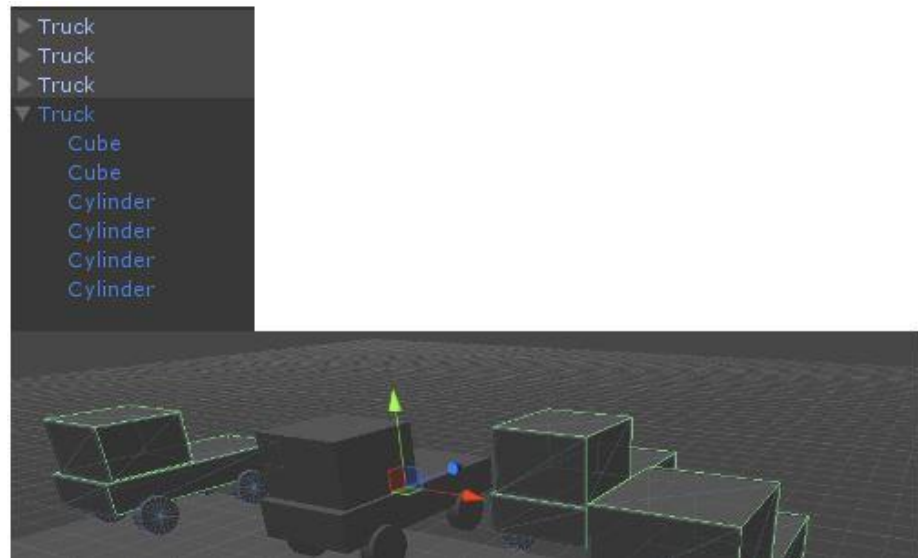
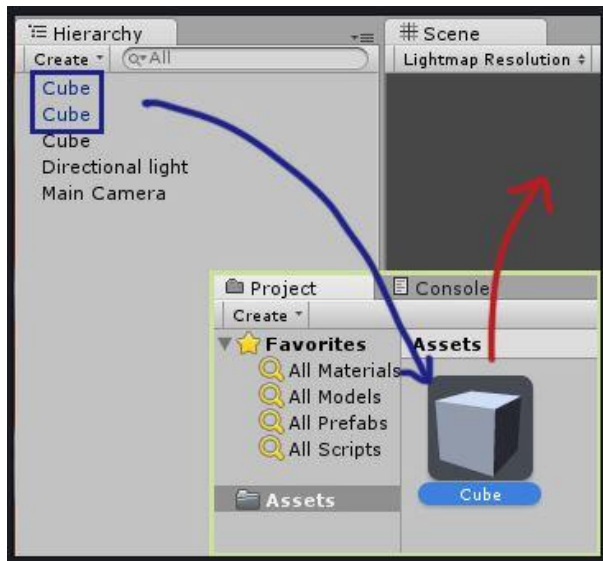
## Rendering

- 2차원 화상에 광원·위치·색상 등의 정보를 이용해 사실적인 3차원 화상을 만드는 과정.



## 프리팸(Prefab)

- 자주 사용하는 객체를 부품처럼 만들어 놓고 재사용할 수 있게 하는 것
- 원본 변경 시 모든 복사본이 자동으로 변경됨



# Rigidbody

## 강체 시뮬레이션 Rigid Body Simulation



# 게임 오브젝트의 이동 및 회전

## transform.[Translate/Rotate]

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

```
public class PlayerMove : MonoBehaviour {  
    public int speed = 5;  
    public int speedRot = 400;
```

```
    void Update () {
```

```
        transform.Translate(Input.GetAxis ("Horizontal") * Time.deltaTime * speed, 0, 0);
```

```
        transform.Translate(0, 0, Input.GetAxis ("Vertical") * Time.deltaTime * speed);
```

```
        transform.Rotate(0, Input.GetAxis ("Mouse X") * Time.deltaTime * speedRot, 0);
```

```
    }
```

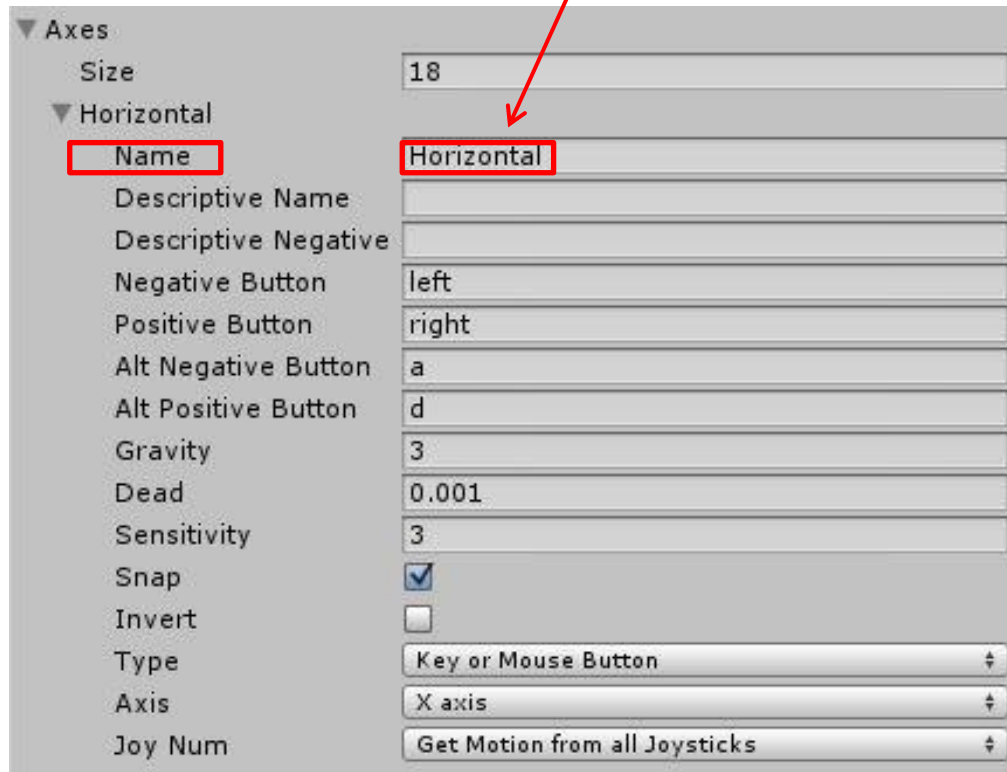
```
}
```



스크립트: C#, 자바스크립트

Input Class : **Input.GetAxis**("키조합 이름")

- Edit → Project Setting → Input 메뉴에 정의
- 예: `h = Input.GetAxis("Horizontal")` : 방향키(←, →) 에 따라 -1 ~ +1의 값





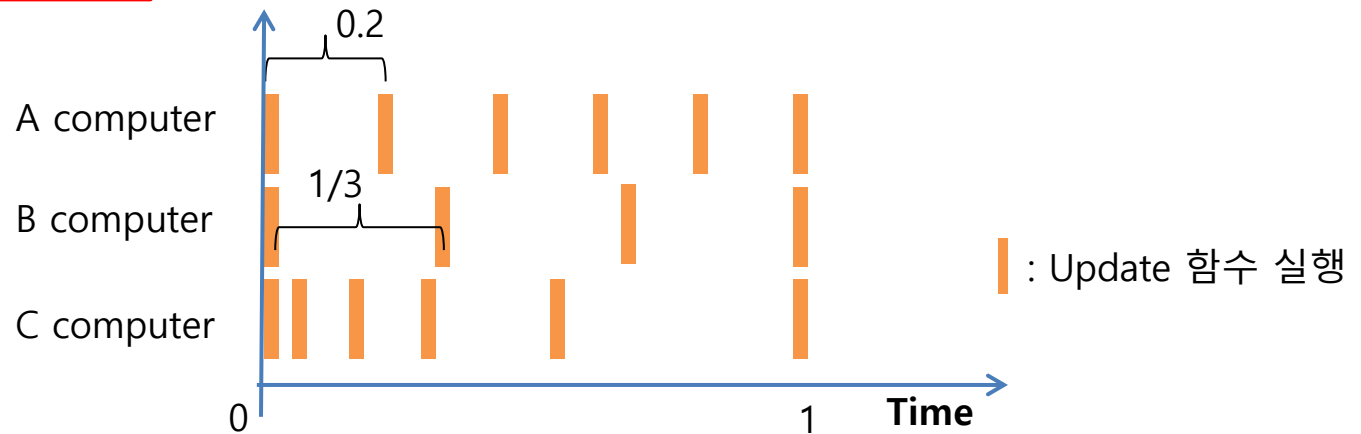
## Update 함수

- 매 프레임마다 실행되는 함수

## Start 함수

- 시작시 한번 실행되는 함수

**Time.deltaTime** : The time in seconds it took to complete the last frame (Read Only).



A Computer :  $5(\text{speed}) \times 5 \times 0.2(\text{Time.deltaTime}) = 5$

B Computer :  $5(\text{speed}) \times 3 \times 0.33(\text{Time.deltaTime}) = 5$

**Time.time** : This is the time in seconds since the start of the game.

**Time.timeScale** : The scale at which the time is passing.

-  $\text{Time.timeScale} = (\text{isPaused}) ? 0.0f : 1.0f;$

**Vector3 구조체**: Representation of 3D vectors and points.

- public Vector3(x: float, y: float, z: float)
- Vector3.forward = Vector3(0,0,1)

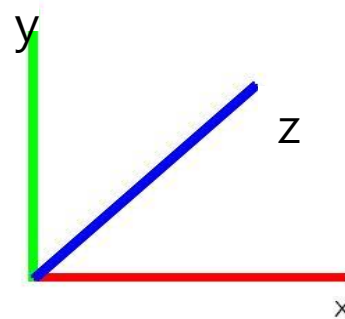
## Static Variables

<a href="#">back</a>	Shorthand for writing Vector3(0, 0, -1).
<a href="#">down</a>	Shorthand for writing Vector3(0, -1, 0).
<a href="#">forward</a>	Shorthand for writing Vector3(0, 0, 1).
<a href="#">left</a>	Shorthand for writing Vector3(-1, 0, 0).
<a href="#">one</a>	Shorthand for writing Vector3(1, 1, 1).
<a href="#">right</a>	Shorthand for writing Vector3(1, 0, 0).
<a href="#">up</a>	Shorthand for writing Vector3(0, 1, 0).
<a href="#">zero</a>	Shorthand for writing Vector3(0, 0, 0).

## Variables

<a href="#">magnitude</a>	Returns the length of this vector (Read Only).
<a href="#">normalized</a>	Returns this vector with a magnitude of 1 (Read Only).
<a href="#">sqrMagnitude</a>	Returns the squared length of this vector (Read Only).

## 유니티좌표계 : 왼손좌표계



# 컴포넌트 불러오기

**GetComponent** <컴포넌트이름>();

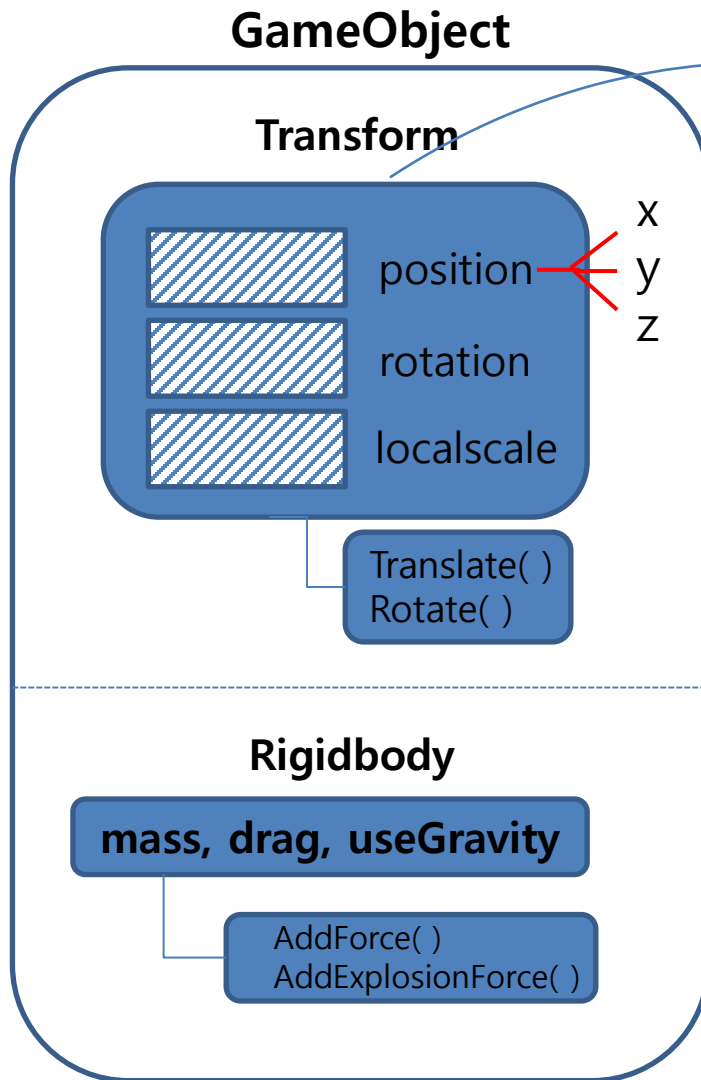
- GetComponentInChildren <컴포넌트이름>();
  - GetComponentInChildren <컴포넌트이름>();
- 

```
public class PlayerMove : MonoBehaviour {  
    void Start () {  
        Rigidbody rb = GetComponent<Rigidbody>();  
        rb.mass = 100;  
        //GetComponent<Rigidbody>().mass = 100  
    }  
}
```

# this.gameObject. **GetComponent<Rigidbody>()**

# Transform 컴포넌트

TransformCompo



```
Transform player;  
public Transform gun;  
public Transform enemy;  
public GameObject robot;  
Vector3 newPosition;  
float x=3f, y=0.5f, z=0f;
```

```
void Start()
```

```
{  
    player = GetComponent<Transform>();  
    player.position = new Vector3(x, y, z);  
}
```

```
void Update()
```

```
{  
    player.Translate( 0, 0, Input.GetAxis("Vertical") * 0.1f);  
    player.Rotate( 0, Input.GetAxis("Horizontal"), 0);  
  
    gun.position = player.position;  
    robot.transform.position = new Vector3(player.position.x +1,  
                                           player.position.y, player.position.z);
```

```
    newPosition.x = 2.0f;  
    newPosition.y = 1.0f;  
    newPosition.z = 3.0f;  
    enemy.position = newPosition;
```

```
    //enemy.position.x = 0.0f; // Error!
```

```
}
```

## Character Controller를 이용한 이동

```
[RequireComponent(typeof(CharacterController))]
```

```
public class CCMove : MonoBehaviour {
    CharacterController CC;
    float z, x;
    float gravity = 0.3f;
    float yAxis = 0f;
    int speed = 5;
    Vector3 sumVector, xVector, zVector, yVector;

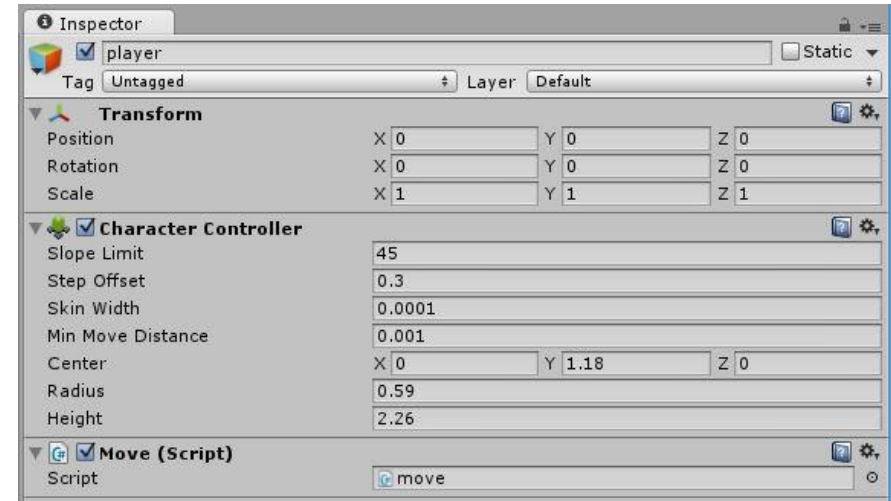
    void Start()
    {
        CC = GetComponent<CharacterController>();
    }

    void Update()
    {
        z = Input.GetAxis("Vertical");
        x = Input.GetAxis("Horizontal");

        zVector = transform.forward * speed * Time.deltaTime * z;
        xVector = transform.right * speed * Time.deltaTime * x;

        if (CC.isGrounded == true)
        {
            yAxis = 0f;
            if (Input.GetButton("Jump"))
                yAxis = 0.1f;
        }
        else {
            yAxis -= gravity * Time.deltaTime;
        }
        yVector = new Vector3(0, yAxis, 0);

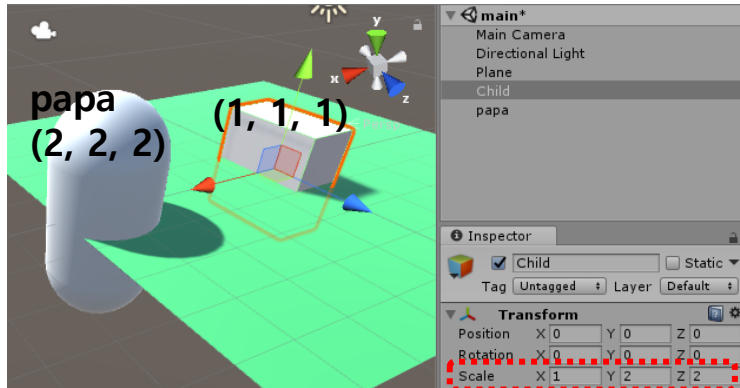
        sumVector = xVector + zVector + yVector;
        transform.Rotate(0, Input.GetAxis("Mouse X"), 0);
        CC.Move(sumVector);
    }
}
```



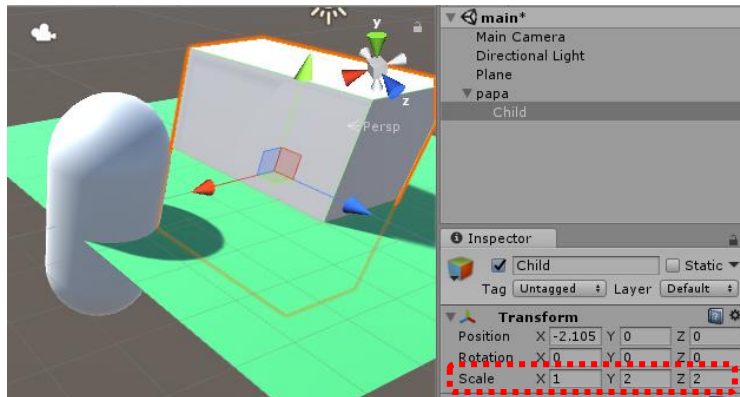
Input.GetAxis Vs. [Input.GetButton](#)

해당 키가 눌러져있는지에 대한 bool값을 반환

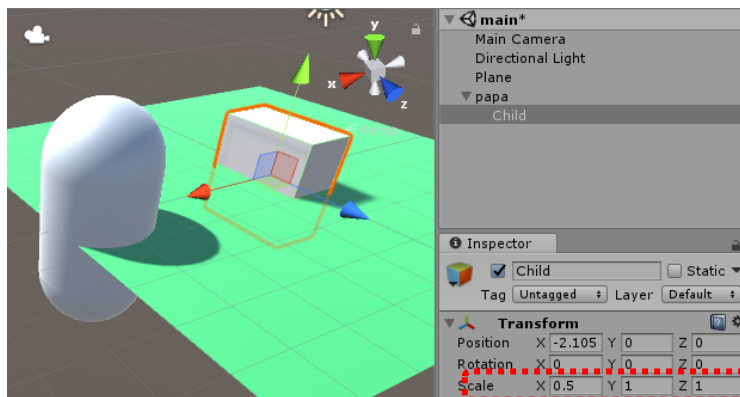
\* `transform.localScale` 변수는 부모의 크기(Scale)에 대한 비를 저장하고 있으며 부모-자식 관계가 설정되지 않은 경우에는 자신의 크기를 저장



```
public Transform papa;
void Start()
{
    //transform.parent = papa;
    transform.localScale = new Vector3(1, 2, 2);
}
```

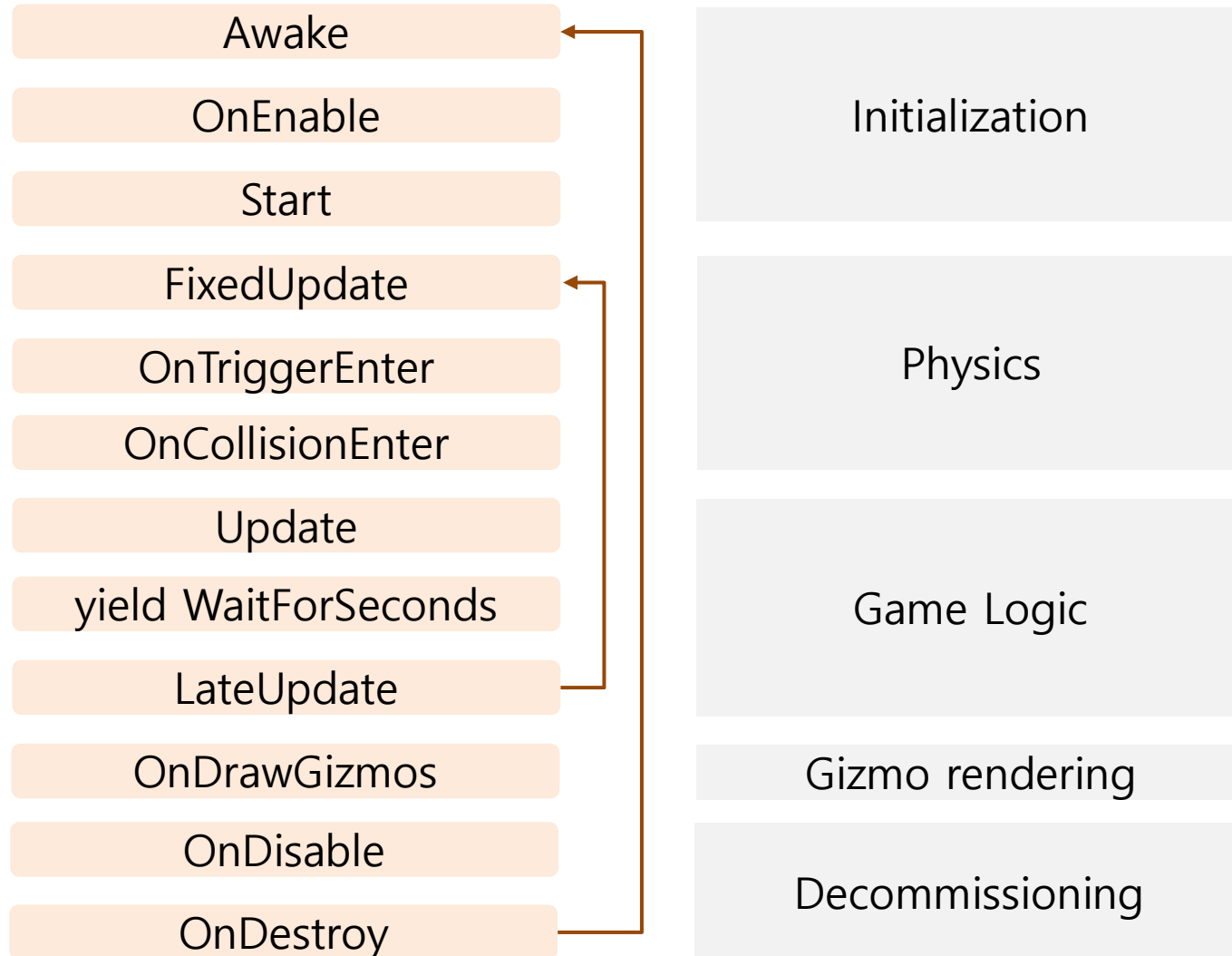


```
public Transform papa;
void Start()
{
    transform.parent = papa;
    transform.localScale = new Vector3(1, 2, 2);
}
```



```
public Transform papa;
void Start()
{
    transform.localScale = new Vector3(1, 2, 2);
    transform.parent = papa;
}
```

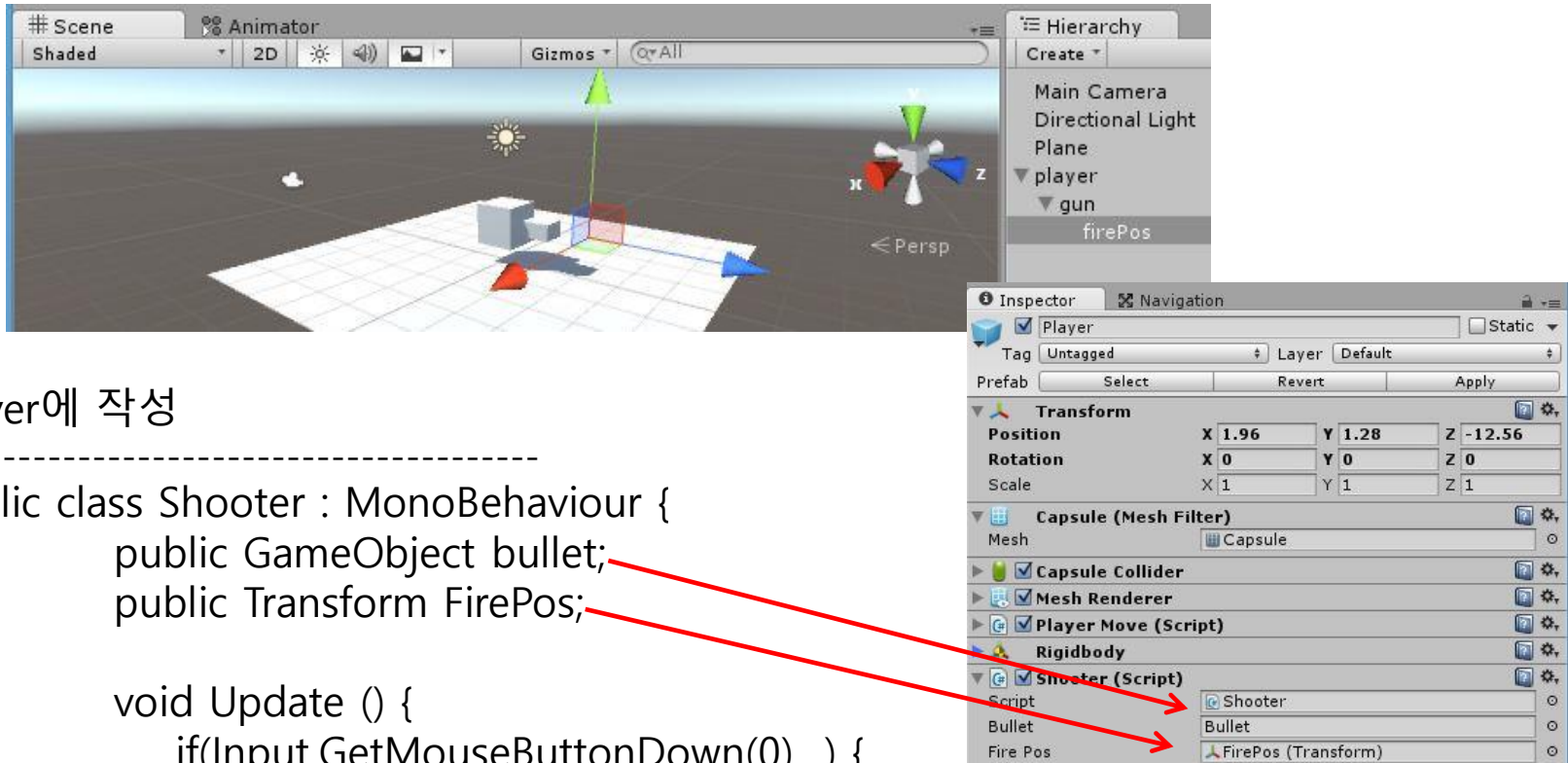
# 이벤트 함수의 실행 순서





# FPS 슈팅1 (Projectile)

Player 자식으로 Gun을 만들고 그 아래에 총알 발사위치를 나타낼 빈 게임오브젝트 생성



Player에 작성

```
public class Shooter : MonoBehaviour {  
    public GameObject bullet;  
    public Transform FirePos;  
  
    void Update () {  
        if(Input.GetMouseButtonDown(0) ) {  
            Instantiate (bullet, FirePos.position, FirePos.rotation);  
        }  
    }  
}
```

# FPS 슈팅2(Projectile)

Main Camera를 Player 자식으로 하여 머리 위치에 세팅

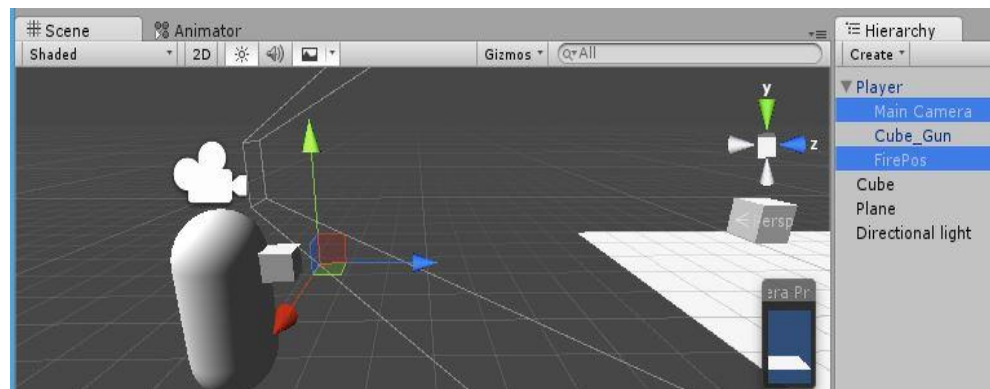
Bullet Prefab 만들기

Bullet Prefab 에 작성

```
public class Shoot : MonoBehaviour {  
    public int Speed;  
    float destroyTime = 2.0f;
```

```
    void Start () {  
        GetComponent<Rigidbody>().AddForce(transform.forward * Speed);  
        Destroy (gameObject, destroyTime);  
    }
```

```
}
```



Input.GetMouseButton(0) : 마우스 왼쪽버튼 클릭 시 계속 발생  
Input.GetMouseButtonDown(1): 마우스 오른쪽버튼 클릭 시 한번 발생  
Input.GetMouseButtonUp(2): 마우스 가운데버튼 놓았을 때 한번 발생

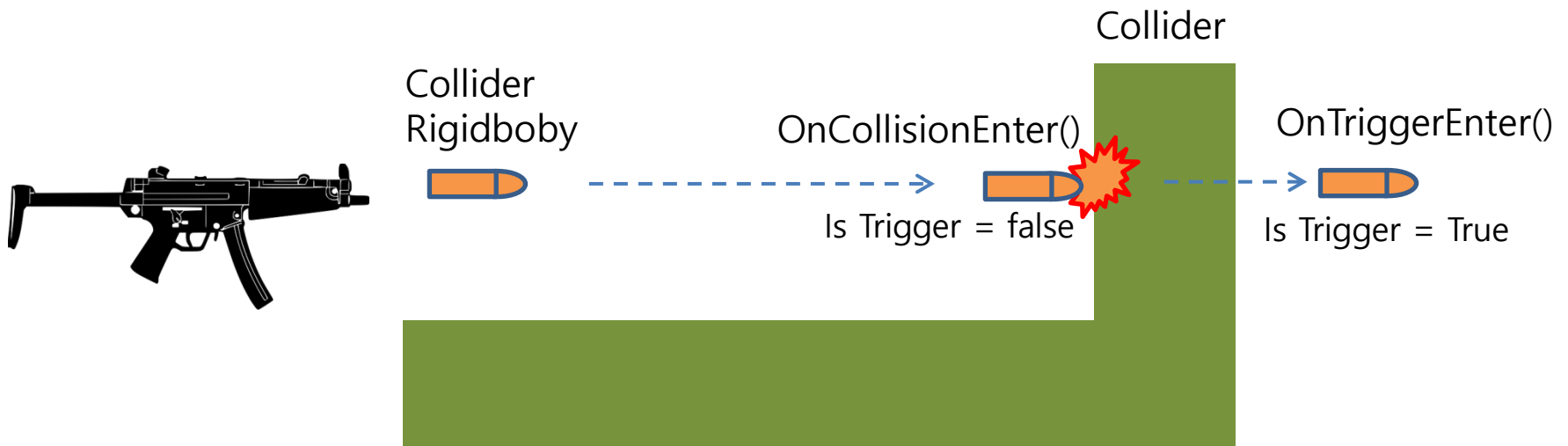
# Collider

Rigidbody 컴포넌트: 물리시뮬레이션

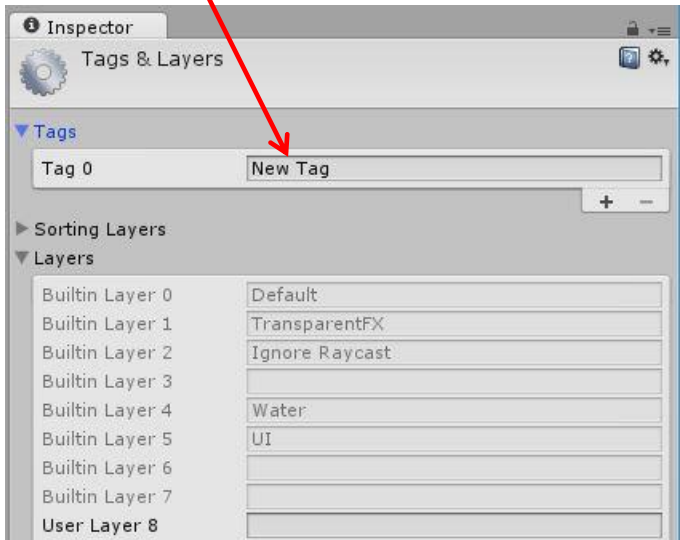
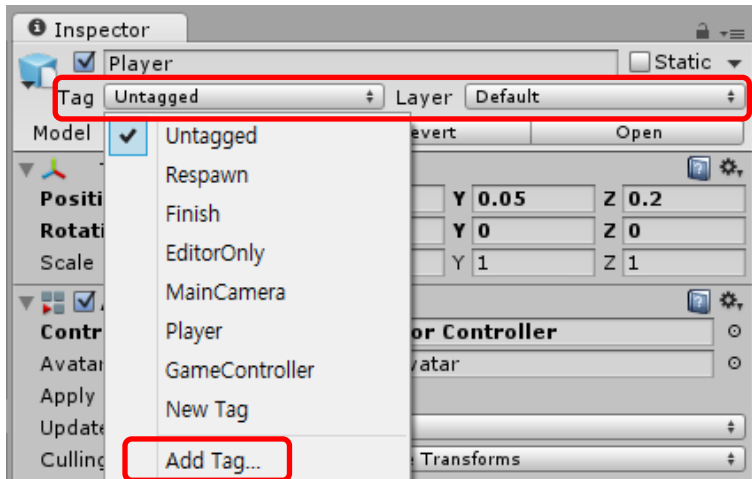
- mass, drag, Use Gravity, Is Kinematic 등의 속성

Collider 컴포넌트: 충돌감지

- [Box, Sphere, Capsule, Mesh, Wheel, terrain] Collider



# Tag & Layer



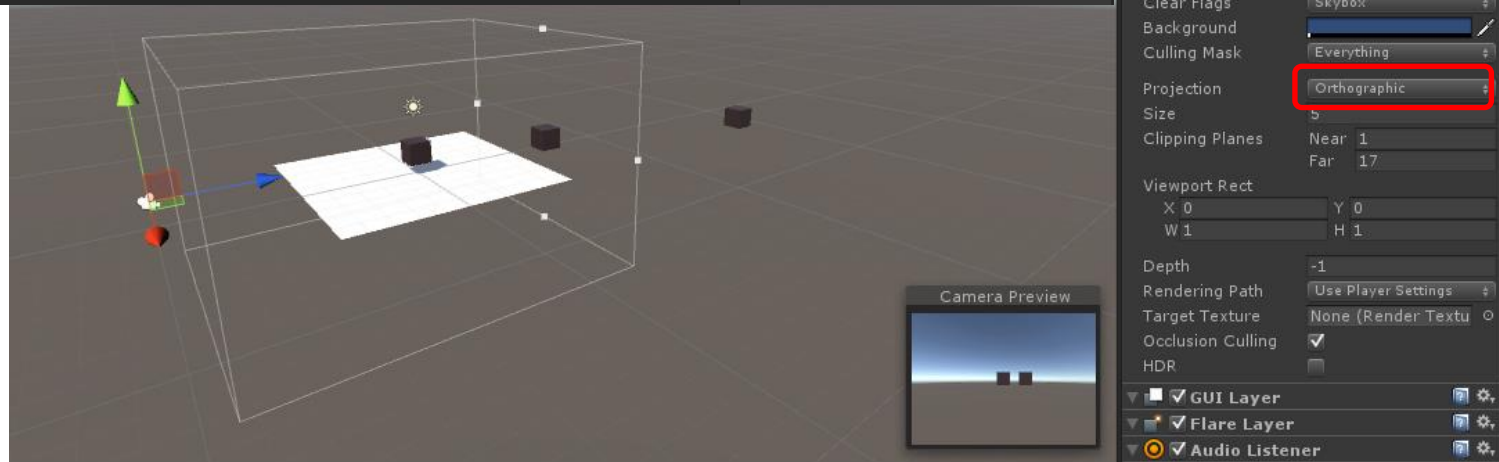
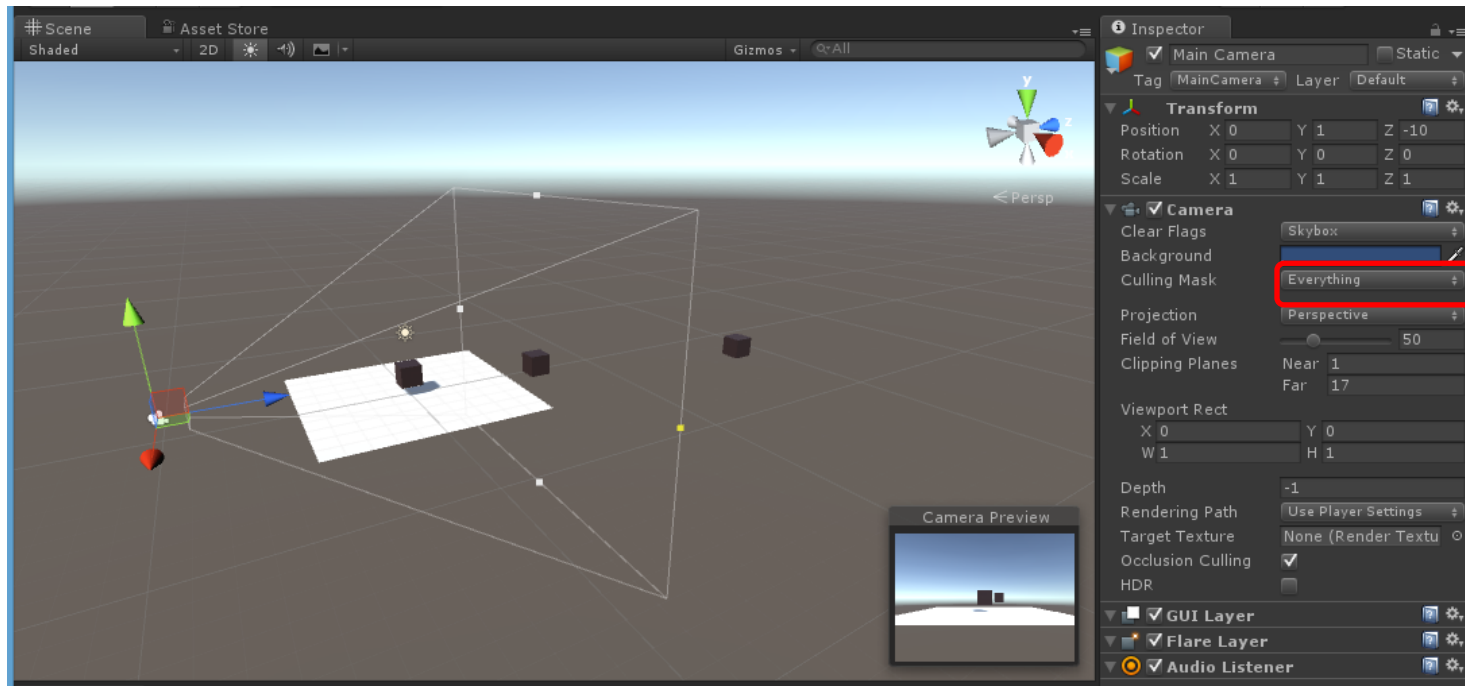
## 관통하지 않는 경우

```
void OnCollisionEnter(Collision coll) {  
    if(coll.gameObject.tag == "BULLET") {  
        Destroy(gameObject);  
    }  
}
```

## 관통하는 경우

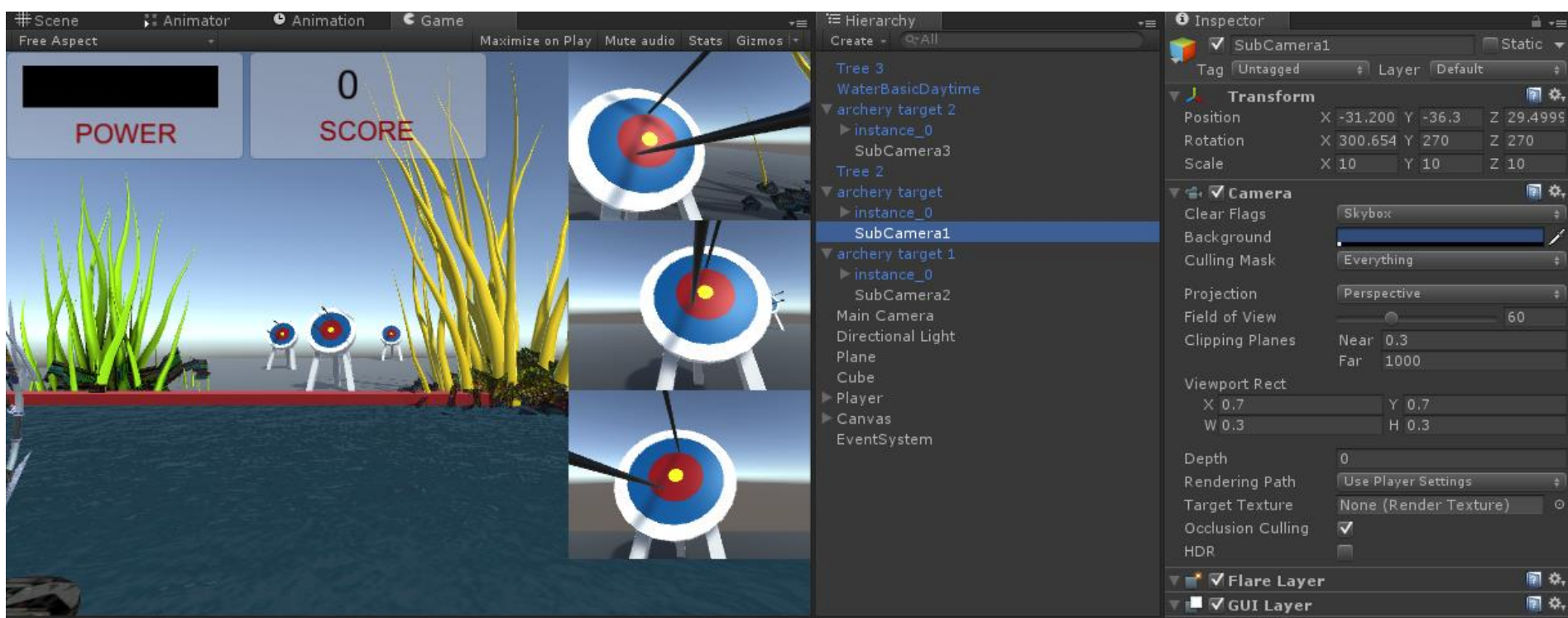
```
void OnTriggerEnter(Collider other) {  
    Destroy(other.gameObject);  
}
```

# Camera



# subCamera

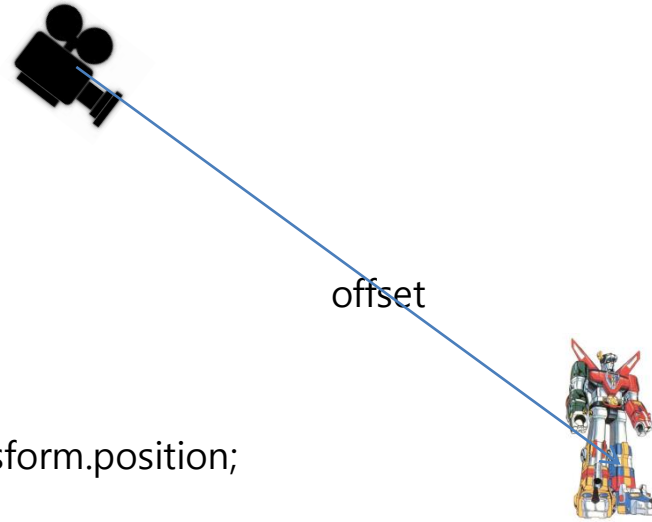
Arrow





# 카메라 추적1

```
public class CamMove : MonoBehaviour {  
    public GameObject player;  
    Vector3 offset;  
  
    void Start () {  
        offset = player.transform.position - transform.position;  
    }  
  
    void Update () {  
        transform.position = player.transform.position - offset;  
        transform.LookAt (player.transform);  
    }  
}
```



```
transform.position = new Vector3(transform.position.x, player.transform.position.y - offset.y, player.transform.position.z - offset.z);
```



# 카메라 추적2

```
public class CamMove : MonoBehaviour {
```

```
    int dist=5;
```

```
    int height=3;
```

```
    public GameObject player;
```

```
    Vector3 offset;
```

```
    void Update () {
```

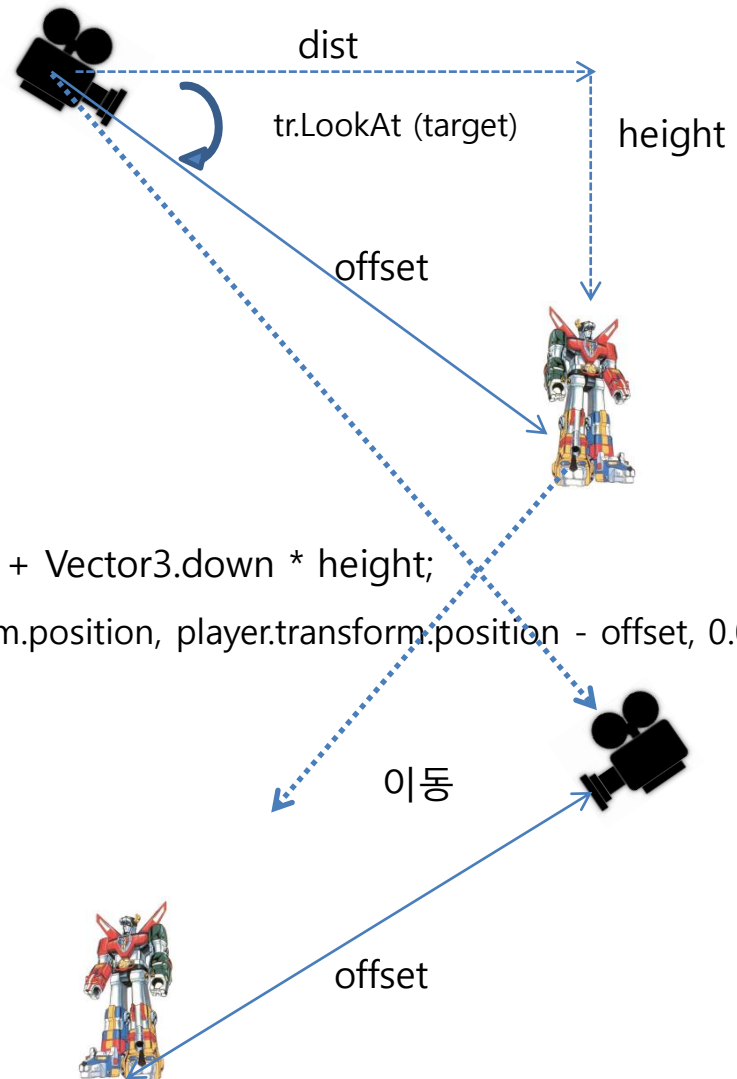
```
        offset = player.transform.forward * dist + Vector3.down * height;
```

```
        transform.position = Vector3.Lerp(transform.position, player.transform.position - offset, 0.05f);
```

```
        transform.LookAt (player.transform);
```

```
    }
```

```
}
```



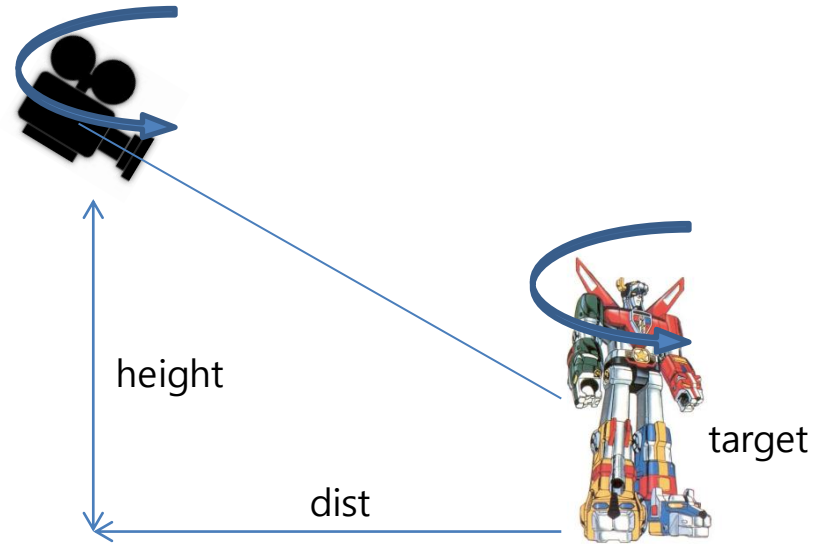
# 카메라 추적3

```
public class FollowCam : MonoBehaviour {
```

```
    public Transform target;  
    float dist = 5.0f;  
    float height = 3.0f;  
    float dampRotate = 0.5f;  
    Transform tr;
```

```
    void Start () {  
        tr = GetComponent<Transform> ();  
    }
```

```
    void LateUpdate () {  
        float currYAngle = Mathf.LerpAngle (tr.eulerAngles.y, target.eulerAngles.y, dampRotate* Time.deltaTime);  
        Quaternion rot = Quaternion.Euler (0, currYAngle, 0);  
  
        tr.position = target.position - (rot * Vector3.forward * dist) + (Vector3.up * height);  
        tr.LookAt (target);  
    }  
}
```




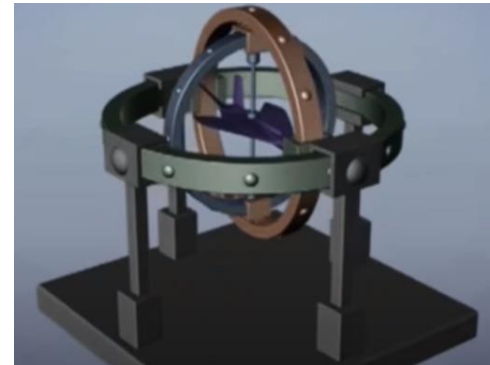
# 짐벌락

- Quaternion

두산백과

## 사원수

[ quaternion  , 四元數 ]



<https://www.youtube.com/watch?v=Mk-RPyRvuhw>

**요약** 복소수의 확장으로서 윌리엄 해밀턴(William Hamilton)이 생각해 낸 수로써, 일종의 벡터이다. 사원수의 사칙연산은 가능하나, 교환법칙은 성립하지 않는다.

일종의 벡터이다. 사원수  $\alpha$ 는  $i^2=j^2=k^2=-1$ 이 되는 세 수  $i, j, k$ 에 대하여  $\alpha=a+bi+cj+dk$  (단,  $a, b, c, d$ 는 실수)로 나타내고, 두 사원수  $\alpha, \alpha' (=a'+b'i+c'j+d'k)$ 의 합은  $\alpha+\alpha'=(a+a')+(b+b')i+(c+c')j+(d+d')k$ 로 정의하고, 두 사원수의 곱은 법칙  $ij=-ji=k, jk=-kj=i, ki=-ik=j$ 에 의하여  $\alpha\alpha'=aa'-bb'-cc'-dd'+(ab'+ba'+cd'-dc')i+(ac'+ca'+db'-bd')j+(ad'+da'+bc'-cb')k$ 로 정의한다.

# Vector3.Lerp Vs. Vector3.MoveTowards

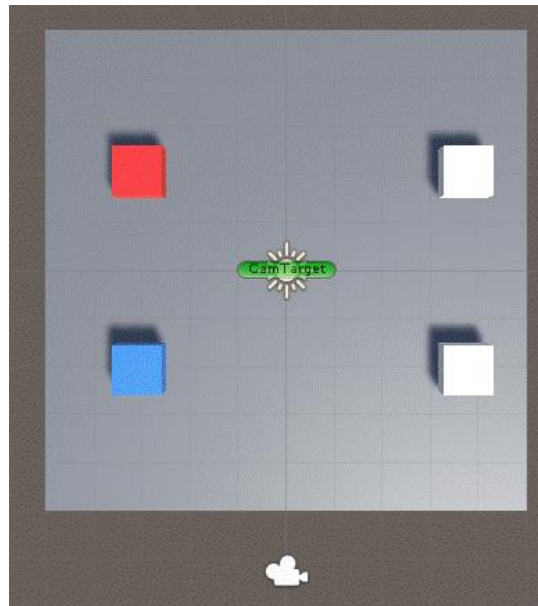
`transform.position = Vector3.Lerp (시작위치 , 도착위치, 0~1 사이의 보간값 );`

`transform.position = Vector3.MoveTowards (시작위치 , 도착위치, 호출당 이동할 거리);`



[https://www.youtube.com/watch?v=\\_QOvSLCXm7A](https://www.youtube.com/watch?v=_QOvSLCXm7A)

```
public class LerpExam : MonoBehaviour
{
    public Transform target;
    public float speed;
    void Update()
    {
        float step = speed * Time.deltaTime;
        transform.position = Vector3.Lerp(transform.position, target.position, step);
        // transform.position = Vector3.MoveTowards(transform.position, target.position, step);
    }
}
```



# DOTween



```
using DG.Tweening;
```

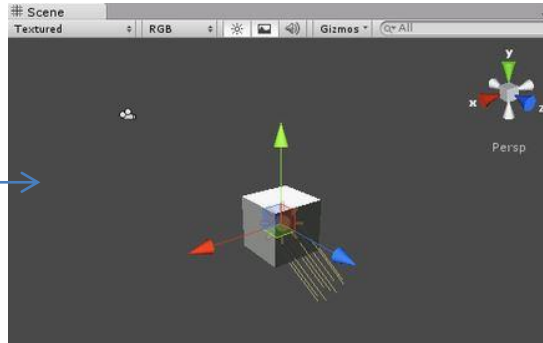
<https://www.youtube.com/watch?v=SZF1oZ-tqMs>

```
public class DOTween : MonoBehaviour
{
    void Start()
    {
        transform.DOMove(Vector3.up, 5);
        transform.DOScale(Vector3.one*3, 5);
        transform.DORotate(Vector3.up*90, 5);

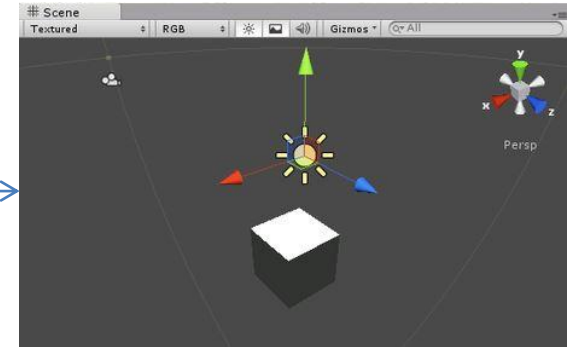
        Material mat = GetComponent<MeshRenderer>().material;
        mat.DOColor(Color.cyan, 5);
    }
}
```

# Light

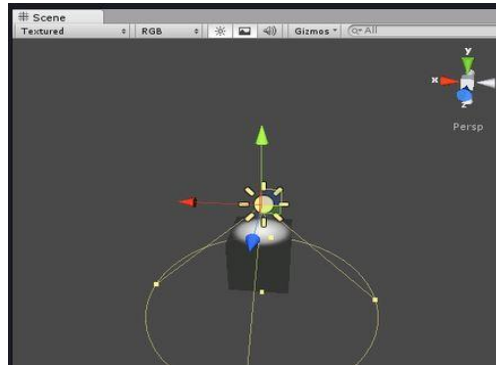
- Directional light →



- Point light →



- Spot light →



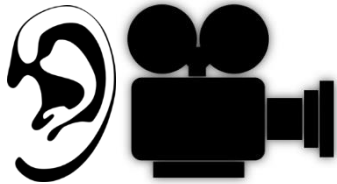
- Area Light : 라이트 맵핑에서 Bake 후 사용하는 간접조명
- Projector를 이용한 그림자: Decal 구현 시 유용, 그림자, 총탄흔적, 혈흔
- Plain Mesh를 이용한 그림자: 가장 부하 적음

라이트 맵핑: 3D 모델에 영향을 미치는 모든 조명 및 빛 반사, 그림자를 텍스처로 베이킹 해서 실행 시 화면과 믹싱

라이트프로브: 라이트 맵을 베이킹할때 라이트프로브에 광원데이터를 저장하여 실행시 근처를 지나는 객체에 색상과 보간하여 실시간 조명 효과



# Sound



Audio Listener  
- Main Camera에 위치



AudioSource



AudioSource



AudioClip

- wav, mp3, ogg 등

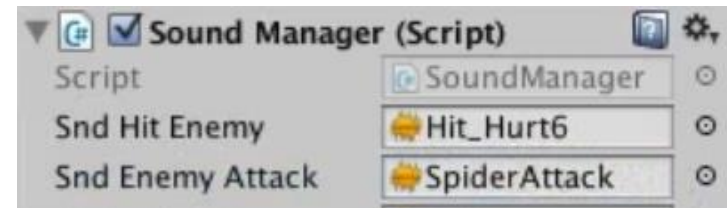
```
public class ExampleClass : MonoBehaviour {  
    AudioSource audio;  
    void Start() {  
        audio = GetComponent<AudioSource>();  
    }  
    void OnCollisionEnter() {  
        audio.Play();  
    }  
}
```

# AudioClip

```
public class SoundManager : MonoBehaviour {  
  
    public static SoundManager instance;  
    AudioSource myAudio;  
  
    public AudioClip sndHitEnemy;  
    public AudioClip sndEnemyAttack;  
  
    void Awake () {  
        if (instance == null)  
            instance = this;  
    }  
  
    void Start () {  
        myAudio = GetComponent<AudioSource> ();  
    }  
  
    public void PlayHitSound() {  
        myAudio.PlayOneShot (sndHitEnemy);  
    }  
  
    public void PlayEnemyAttack() {  
        myAudio.PlayOneShot (sndEnemyAttack);  
    }  
}
```

사운드 호출

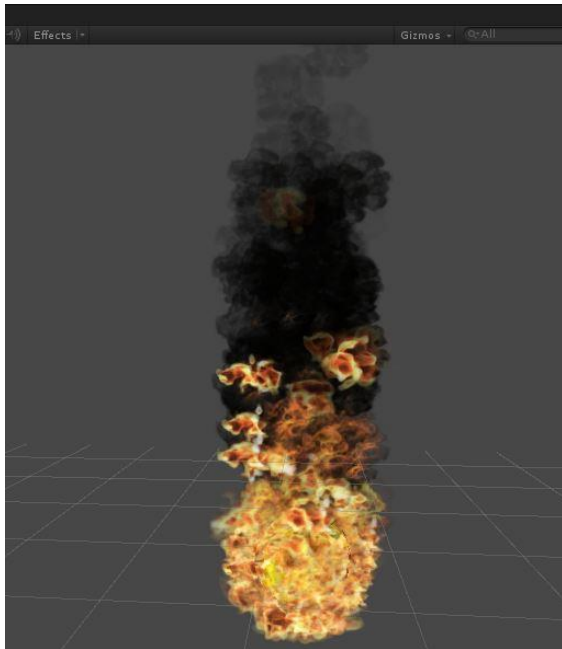
SoundManager.instance.PlayHitSound()



# Particle

- 화염, 안개, 연기, 폭발, 불 등의 효과

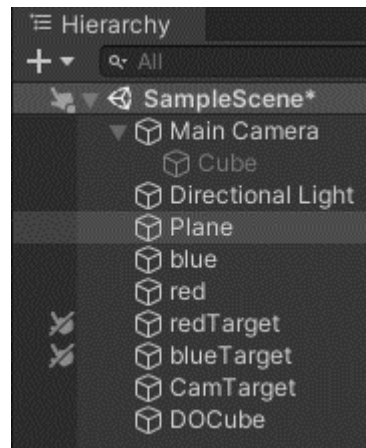
## 입자 시스템 Particle System



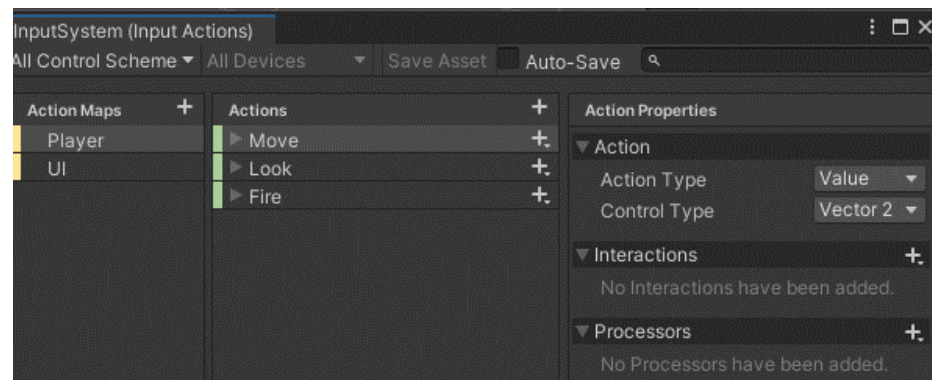
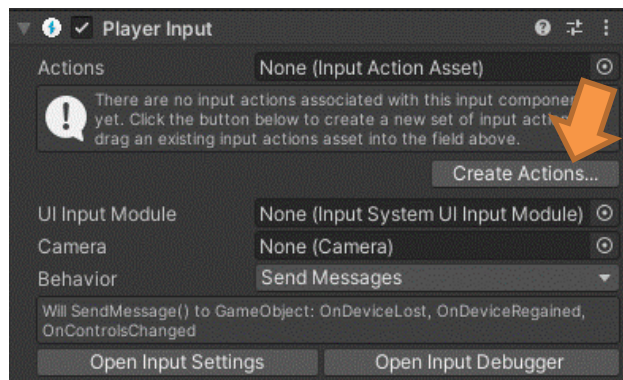
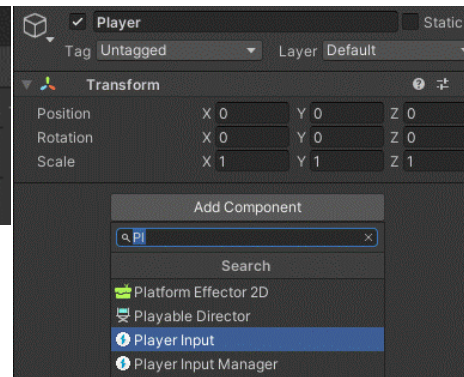
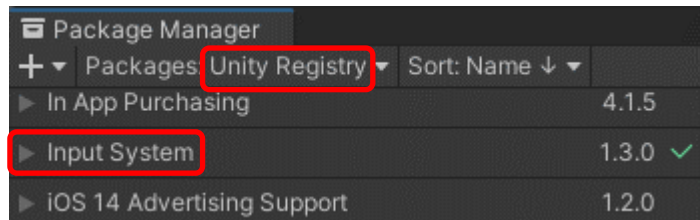
```
public class WallCtrl : MonoBehaviour {  
    public GameObject sparkEffect; //스파크 파티클 프리팹 연결 변수  
    // public Particlesystem sparkEffect; sparkEffect.Play(), .Stop()  
  
    void OnCollisionEnter ( Collision coll )  
    {  
        if (coll.collider.tag == "BULLET")  
        {  
            //스파크 파티클 동적으로 생성 후 변수에 할당  
            Object obj = Instantiate ( sparkEffect, coll.transform.position, Quaternion.identity );  
            //2초 후에 스파크 파티클 삭제  
            Destroy ( obj, 2.0f );  
  
            //충돌한 게임오브젝트 삭제  
            Destroy ( coll.gameObject );  
        }  
    }  
}
```

# Unity Editor Tip

- 뷰 확대/축소 : Mouse 원하는 창 + Shift + Space
- Create Empty : Ctrl + Shift + N (자식 : Alt + Shift + N)
- Vertex Snapping : 오브젝트 선택 + V
  - 지면(Terrain 등) : Ctrl + Shift
- Scene view와 Game view 일치 : 카메라 선택 후 Ctrl + Shift + F
- Inspector 창의 Normal과 debug 모드(private도 보임)
- [Range(0,10)]; public int n=0;
- 오브젝트 고정(선택 불가):



# New Input System





# New Input System

```
using UnityEngine.InputSystem;
```

```
public class Player : MonoBehaviour  
{
```

```
    Vector2 inputVec;
```

```
    Vector3 MoveVec;
```

```
    void OnMove(InputValue value)
```

```
{
```

```
        inputVec = value.Get<Vector2>();
```

```
        MoveVec = new Vector3(inputVec.x, 0, inputVec.y);
```

```
}
```

```
    private void Update()
```

```
{
```

```
        transform.Translate(MoveVec * Time.deltaTime*5);
```

```
}
```

```
}
```

