



UNITY -CHAPTER3-

SOUL SEEK

목차

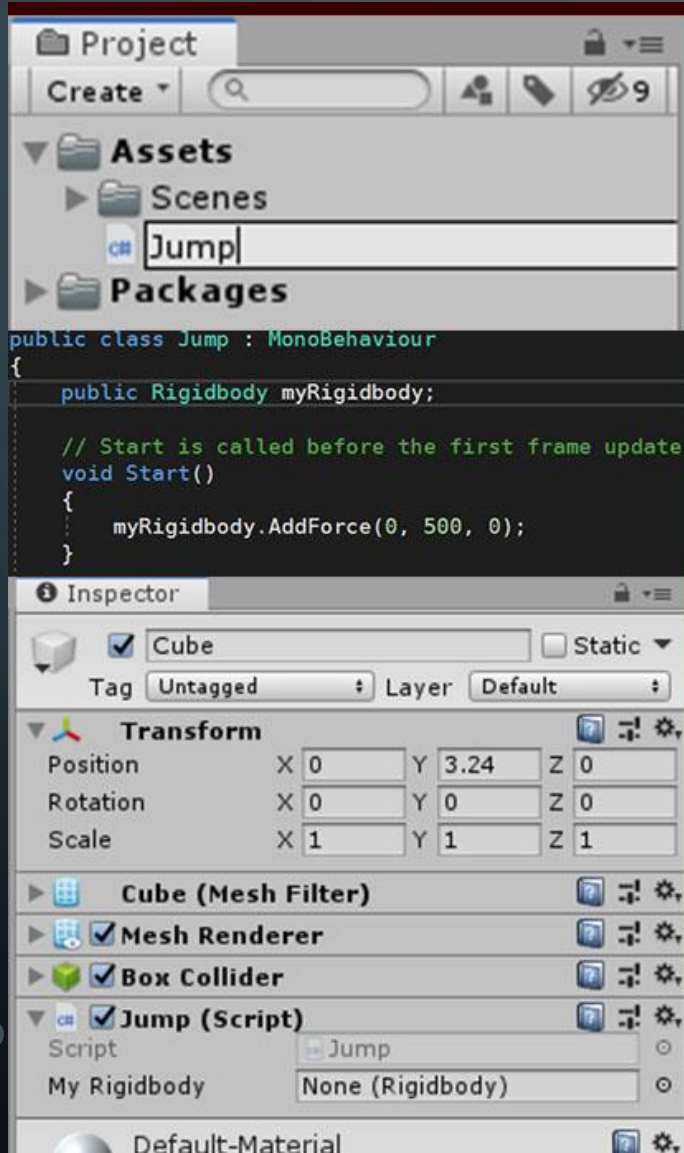
1. 컴포넌트를 변수로 사용하기
2. GameObject와 Prefab
3. PhysX – Collider, Rigidbody

The background is a dark blue gradient with a large, faint circular pattern in the center. The corners are decorated with white line art resembling circuit boards, with lines and small circles representing components.

컴포넌트를 변수로

1. 컴포넌트를 변수로 사용하기

물리가 적용되는 **Cube**를 준비해서 이 물리 작용을 하는 **Rigidbody, Collider**를 직접 적으로 변수로 사용하는 방법에 대해 알아보자.



1. Jump Component를 만들자.

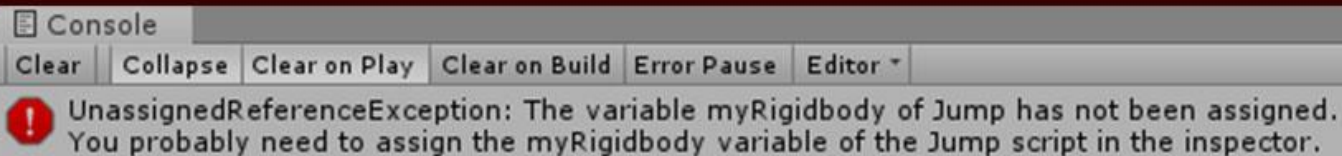
2. 만든 Component를 VisualStudio로 수정해 보자.

외부에서도 사용할 수 있도록 public으로 Rigidbody를 선언하자.

3. Start Method에서 수정사항을 적용해보자. Vector값으로 X, Y, Z값을 각각 적용해서 설정해준다.

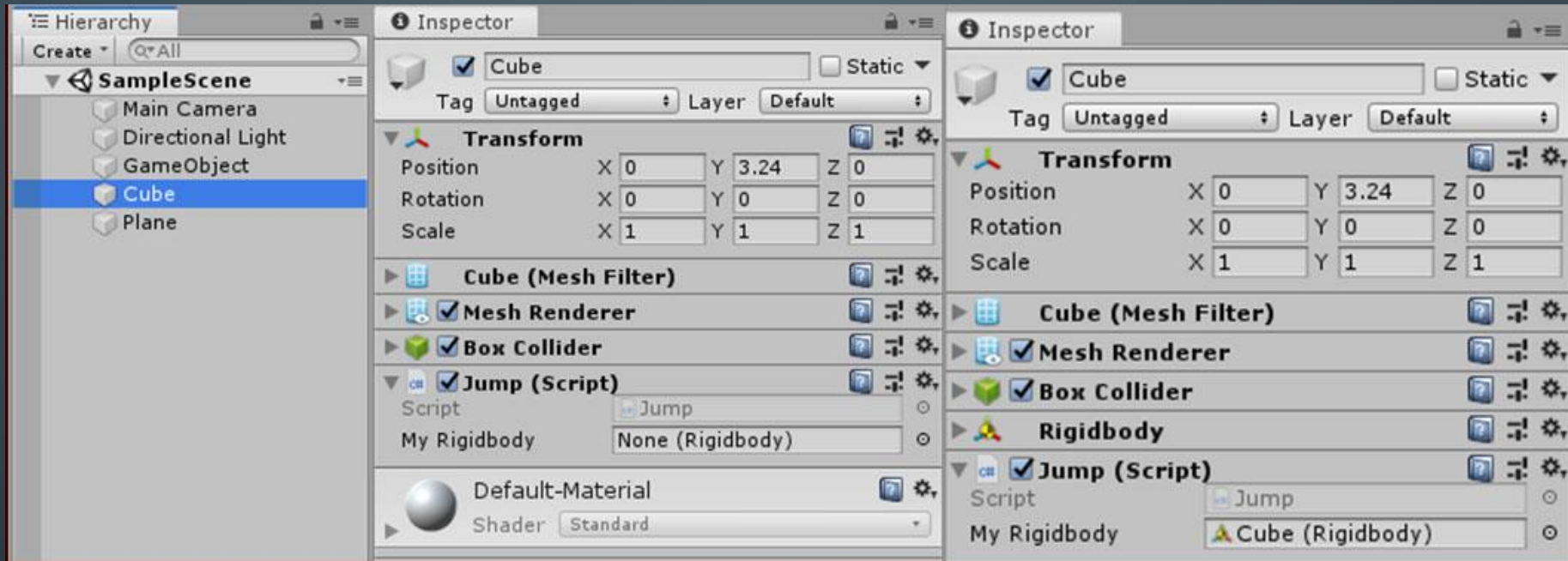
4. Cube Object에 AddComponent 또는 드래그 앤 드롭으로 가져다 놓는다.

5. 실행버튼으로 실행하면 아래와같은 할당하지 않은 객체라는 Exception이 발생한다.



1. 컴포넌트를 변수로 사용하기

Component는 기본적으로 할당을 해야 사용할 수 있다.



1. Component를 Inspector에서 확인해 보면 선언한 Rigidbody형식의 변수를 볼 수 있고 옆에 None(Rigidbody)를 볼 수 있다.

2. public 으로 해두었기 때문에 Inspector창에서 확인 가능하며 편집도 가능하기 때문에 Inspector창에서 Rigidbody가 Add 되어 있는 오브젝트를 링크 시켜주면 할당되는 것이다.

GAMEOBJECT & PREFAB

2. GAMEOBJECT & PREFAB

GameObject와 **Transform Component**에는 유용한 **Method**들이 있다.

activeInHierarchy : **Hierarchy**상에 활성화되어 있는지 여부를 파악한다.

```
gameObject.activeInHierarchy
```

```
bool GameObject.activeInHierarchy { get; }  
Defines whether the GameObject is active in the Scene.
```

SetActive : **Object**를 활성화 비활성화 한다.

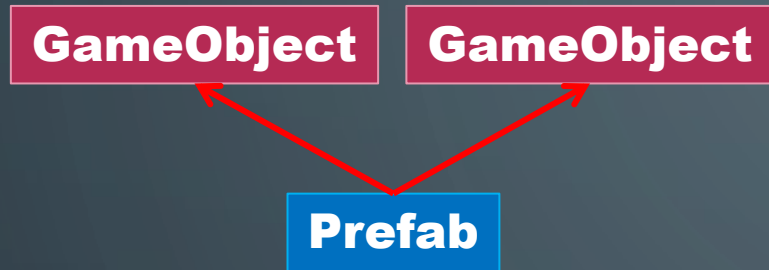
```
//활성화  
gameObject.SetActive(true);  
//비활성화  
gameObject.SetActive(false);
```

GetComponent : **Object**에 첨부된 **Type**이 있다면 해당 **Type**의 구성요소를 반환한다.

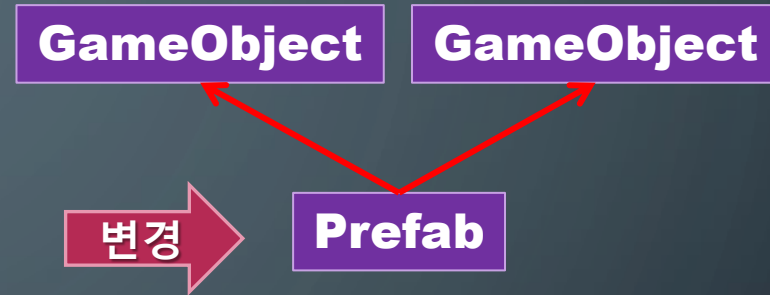
```
//GameObject가 가지고 있는 Component를 반환받아 할당받을 수 있다.  
Jump jump = gameObject.GetComponent<Jump>();
```

2. GAMEOBJECT & PREFAB

- **GameObject**의 생성과 관리를 수월하게 만드는 일종의 설계도면 같은 역할을 수행.
- **Hierarchy**에 생성한 **GameObject**를 **Project View**로 드래그하면 **Prefab**으로 생성.
- **Project View**에서 더블 클릭하거나 **Inspect View**에서 **Prefab** 모드로 전환할 수 있다.



똑같은 **GameObject**를
여럿 필요할 때 **Prefab**을
만들어 가져다 쓰면 된다.



Prefab으로 생성한 **Object**를 사용할 때
Prefab을 변경하면 배치되어있는 모든
Object에 적용된다.

Hierarchy에 생성된 **Object**들을 보면
Clone이라고 되어 있다.

Resource.Load()

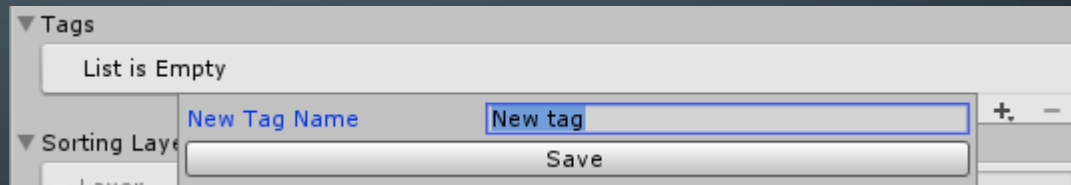
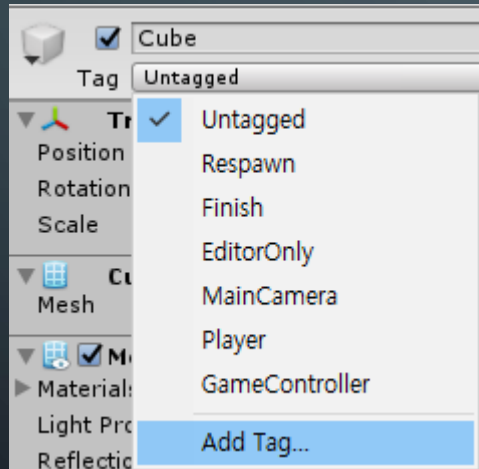
- **Asset** 폴더아래에 **Resources** 라는 이름의 폴더를 기본 경로로 하고 있다.
- 파일명을 부르듯이 경로명을 인자 값으로 생성할 수 있다.

```
Transform posTransform;  
GameObject prefab;  
void Start()  
{  
    GameObject obj = Instantiate(Resources.Load("") as GameObject, posTransform);  
    GameObject obj2 = Instantiate(prefab, posTransform);  
}
```


2. GAMEOBJECT & PREFAB

GameObject 검색

- **GameObject.Find("이름");**
 - 성능을 많이 잡아먹어서 **Update**에서 사용 금물
 - 비활성화 상태의 **Object** 검색을 못한다.
- **GameObject.FindWithTag("태그명");**
- **GameObject.FindGameObjectWithTag("태그명");**
 - 태그를 설정해두고 태그로 오브젝트를 찾는다.
 - 비활성화 상태의 **Object** 검색을 못한다.

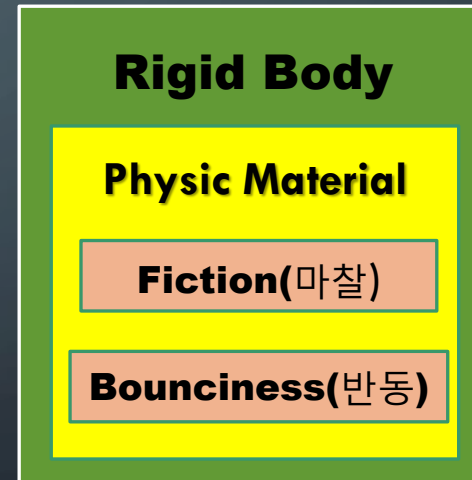


- **GameObject.FindChild("자식명");**
- **GameObject.GetChild(자식 번호);**
 - 자신의 자식에 있는 오브젝트를 얻고 싶을 때
 - 비활성화 상태의 오브젝트를 얻고 싶을 때

PHYSX - COLLIDER, RIGIDBODY

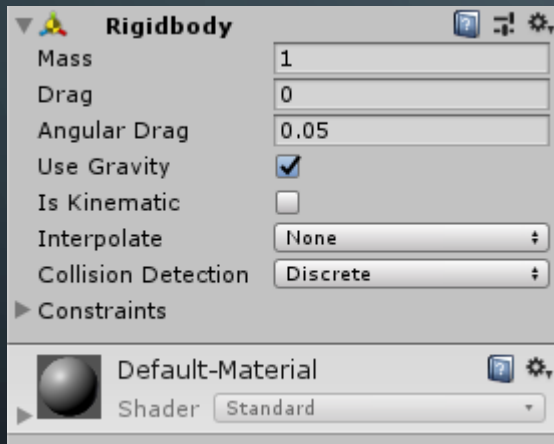
3. PHYSX - COLLIDER, RIGIDBODY

- **NVIDIA**의 **PhysX**물리엔진을 탑재하고 있다.
- **Rigid Body**로 물리기반은 움직임을 체크한다.
- **Collider**로 물체 간의 충돌을 체크한다.
- **Physic Material**로 충돌체크를 해서 반동을 준다.
- 물리적인 계산을 하기때문에 발사체의 경우 로우 폴리곤이나 평면으로 하거나 **POOL**시스템을 구축하여 재활용하는 형식을 한다.
- **FixedUpdate()**를 이용하여 계산하자.



3. PHYSX - COLLIDER, RIGIDBODY

- **Unity** 물리엔진에서 기본물리법칙을 적용 받는 **Object**에게 적용시키기 위한 **Component**
- **갈릴레오의 자유낙하 법칙**을 그대로 적용하였다.
- 실제와는 다르게 저항 값에 의해 낙하속도가 달라진다.
- 모델링에 적용할 시 모델링의 크기에 유의해야한다. (**Unity**에선 **1**의 값을 **1m**로 계산)
- **FBX** 임포트시 **Scale Factor**로 크기를 조정하자. (**Transform**에서의 스케일도 물리계산에 포함, 비효율)



- **Mass** : 질량을 의미 한다. 질량에 따라 낙하속도가 결정되지 않는다.
- **Drag** : 저항력을 의미 한다. 이 수치에 의해 움직이는 속도가 결정된다.
- **Angular Drag** : 회전 저항력을 의미한다. Drag 수치에 영향을 받는다.
- **Use Gravity** : 중력을 사용할지를 설정한다.
- **Is Kinematic** : 물리효과를 적용 받지 않는다.
- **Interpolate** : FixedUpdate()로 계산 시 시간간격때문에 움직임이 끊어져 보일 때 보간을 이용해 보정해준다. Interpolate – 이전 프레임의 Transform을 기준으로 보정해 준다. Extrapolate – 다음 프레임의 값을 추정해서 보정해 준다.
- **Collision Detection** : 너무 빠르게 움직이는 물체는 프레임 단위에서 놓칠 수 있는데 좀 더 세밀하게 체크하라고 설정해 줄 수 있는 옵션이다. Discrete → Continuous → Continuous Dynamic 의 순서로 정밀도가 올라간다.
- **Constraints** : 체크한 축의 회전과 이동을 금지시킨다.

3. PHYSX - COLLIDER, RIGIDBODY

Rigidbody Code 활용

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

```
public class Bullet : MonoBehaviour  
{
```

```
    //총알의 파괴력
```

```
    public float damage = 20.0f;
```

```
    //총알 발사 속도
```

```
    public float speed = 1000.0f;
```

```
    Rigidbody rigid;
```

```
    void Awake()  
{
```

```
        rigid = GetComponent<Rigidbody>();  
    }
```

```
    void Start()  
{
```

```
        rigid.AddForce(transform.forward * speed);  
    }
```

Vector로 힘을 가하는 함수.

```
void AddForce(Vector3 force);
```

- 월드 좌표의 기준이므로 발사하는 주체가 회전을 한다면 잘못된 발사가 된다.

```
void AddRelativeForce(Vector3 force);
```

- 발사 주체의 좌표축을 기준으로 하려면 이 함수를 써야한다.

```
rigid.AddRelativeForce(Vector.forward * speed);
```

3. PHYSX - COLLIDER, RIGIDBODY

Collider

- 충돌을 감지하는 센서역할을 한다.
- 기본형태와 특수형태가 있다.
- **is Trigger** 프로퍼티를 통해 충돌감지에 대한 처리를 설정 할 수 있다.
- **Tag**를 이용한 충돌 이벤트 처리를 활용 할 수 있다.
- 충돌 감지 조건이 충족해야 이벤트가 발생한다.

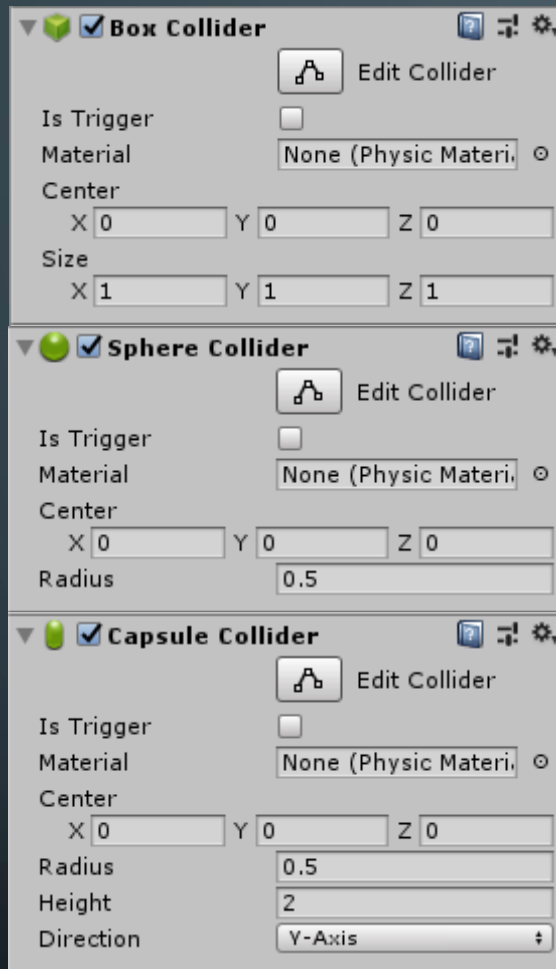
충돌 이벤트

- **is Trigger**가 체크되어 있지 않을 때
void OnCollisionEnter : 두 물체가 충돌이 일어나기 시작 했을 때 발생한다.
void OnCollisionStay : 두 물체 간의 충돌이 지속될 때 발생한다.
void OnCollisionExit : 두 물체가 다시 떨어졌을 때 발생한다.
- **is Trigger**가 체크되어 있을 때
void OnTriggerEnter : 두 물체가 충돌이 일어나기 시작 했을 때 발생한다.
void OnTriggerStay : 두 물체 간의 충돌이 지속될 때 발생한다.
void OnTriggerExit : 두 물체가 다시 떨어졌을 때 발생한다.

충돌 감지 조건

- 충돌을 일으키는 양쪽 게임 오브젝트 모두 **Collider Component**가 추가되어 있어야한다.
- 두 **GameObject**중에 움직이는 쪽에는 반드시 **Rigidbody**가 있어야 한다.
- **is Trigger**가 체크되어 있다면 멈추거나 바운드되는 물리효과는 일어나지 않고 이벤트 감지만 한다.

3. PHYSX - COLLIDER, RIGIDBODY



Box Collider

- **Is Trigger** : 충돌이 발생했을 때 충돌이벤트를 발생시킨다. **Rigid body**가 없어도 충돌이벤트를 발생 시킬 수 있기때문에 여러가지로 활용된다.
- **Material : Physic Material**이 필요 할 경우 설정한다.
- **Center** : 중심점의 위치를 설정한다.
- **Size** : 박스의 크기를 설정한다.

Sphere Collider

- **Radius** : 구의 반지름을 지정한다. 반지름 반경으로 설정한다.

Capsule Collider

- **Radius** : 기둥 위아래 반구의 반지름을 설정한다.
- **Height** : 기둥의 높이를 설정한다.
- **Direction** : 기둥이 생성되는 축방향을 설정한다.

충돌 체크에 대한 검증을 각자 가지고 있는 재원을 기반으로 하기 때문에 재원이 적을 수록 그 처리 속도와 부담이 적어진다.

Sphere -> Capsule -> Box 순서이다. 그러므로 **Sphere**로 대부분 처리하는 것이 효율적이다.

3. PHYSX - COLLIDER, RIGIDBODY

Collider Code 활용

Tag나 Layer를 설정해서 충돌 감지에 활용 할 수 있다.

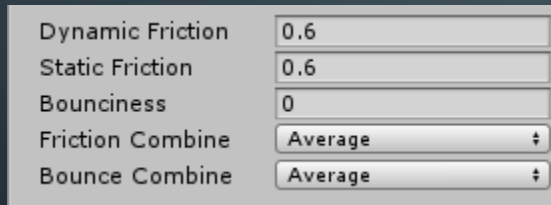
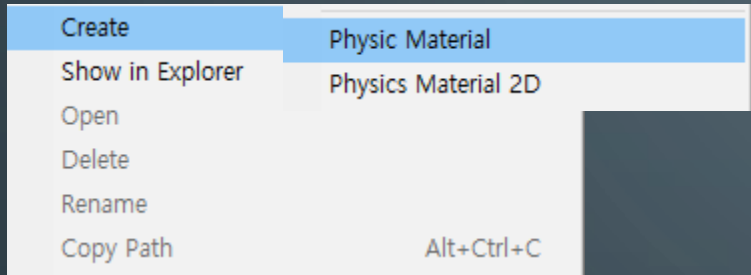
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RemoveBullet : MonoBehaviour
{
    //충돌이 시작할 때 발생하는 이벤트
    private void OnCollisionEnter(Collision coll)
    {
        //충돌한 게임오브젝트의 태그값 비교
        if(coll.collider.tag == "BULLET")
        {
            //충돌한 게임오브젝트 삭제
            Destroy(coll.gameObject);
        }
    }
}
```

3. PHYSX - COLLIDER, RIGIDBODY

Physic Material

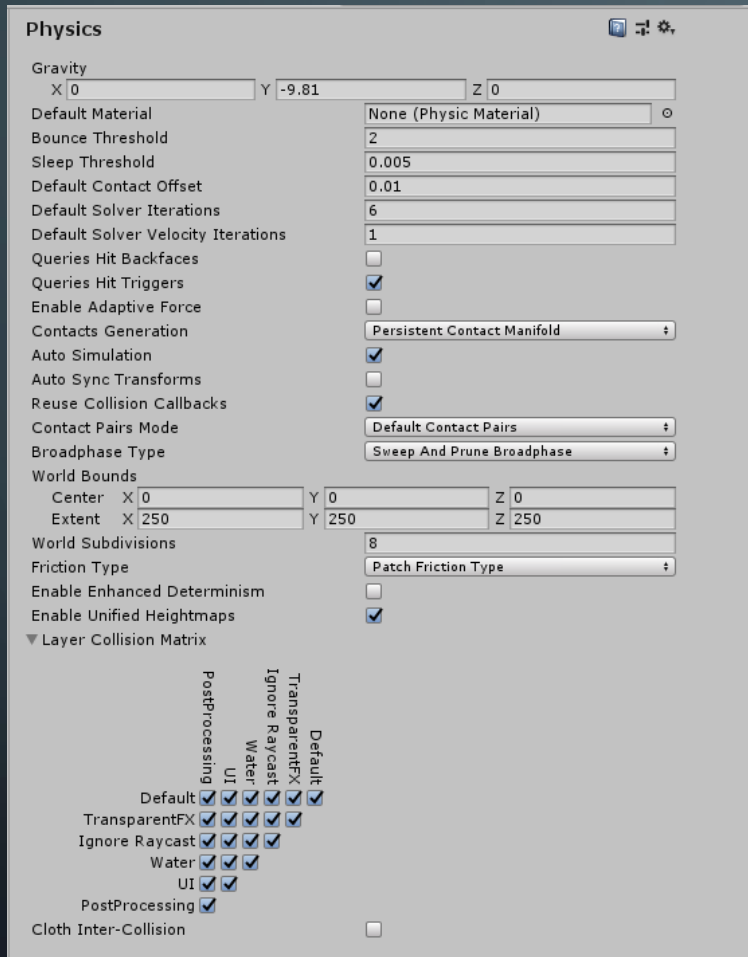
- 충돌체 표면의 마찰과 반동을 설정하는 기본 **Asset**이다.
- Rigidbody**와 충돌할 경우 설정된 물리재질에 따른 다양한 상호 작용을 볼 수 있다.
- Standard Assets**를 통해 기본적으로 셋팅 된 것을 제공하고 있다.



- Asset -> Create**에서 찾을 수 있다.
- Dynamic Friction** : 물체의 면에 작용하는 운동 마찰력, 물체가 접촉면에서 움직이고 있을 때 받는 마찰력의 크기를 지정한다.
- Static Friction** : 물체의 정지 마찰력, 물체가 멈춰 있을 때 외부의 힘으로 부터 안 움직이도록 버티는 힘을 의미.
- Bounciness** : 충돌이 일어날 때의 반동을 설정, **0**이면 반동이 발생하지 않으며, **1**이면 에너지의 손실 없이 무난히 반동 한다.
- Friction Combine** : 다른 물리 재질과 충돌할 경우 최종 마찰력을 어떻게 산출할지 정한다. **Average** - 평균값, **Multiply** - 곱, **Minimum** - 최소값, **Maximum** - 최대값.
- Bounce Combine** : 다른 물리 재질과 충돌할 경우 반동을 어떻게 설정할지 지정한다.

3. PHYSX - COLLIDER, RIGIDBODY

Physic Manager



- **Edit -> Project Setting -> Physics**를 선택해서 불러올 수 있다.
- **Gravity : Rigidbody의 Use Gravity**에 체크하면 **Gravity**에 설정된 값으로 중력이 작용한다. 기본값은 **Y**축으로 **-9.81**로 되어 있다. 즉, 중력 가속도 **9.8**이 설정되어 있는 것이다.
- **Default Material** : 두 물체가 충돌했을 때 반작용에 대한 속성을 설정한다. **None**으로 설정하지 않으면 각 **Rigidbody**에서 개별적으로 설정 할 수 있다.
- **Sleep Velocity, Sleep Angular Velocity**
: 이동가속도와 회전가속도가 일정 수치 이하로 떨어지면 자동으로 물리엔진의 영향에서 벗어나게 해 **CPU** 연산의 부하를 줄여주는데, 이를 **Rigidbody Sleeping**이라 한다.
- **Raycasts Hit Triggers** : 체크가 해제되면 **Raycast**와의 충돌감지를 하지 않는다.
- **Layer Collision Matrix : Build In Layer** 또는 사용자 정의 **Layer** 간의 충돌 감지여부를 체크 할 수 있다.

Rigidbody Sleeping으로 더 이상 동작하지 않고 휴면중인 **Rigidbody**를 깨우는 방법은 다음과 같다.

- **Rigidbody**가 있는 다른 오브젝트와의 충돌
- **Rigidbody**의 속성을 변경하거나 **AddForce**같은 함수로 힘이 가해질때.