



초심자를 위한...?

Unity3D Overview

이 문서는 Unity 5.2.3f 버전을 토대로 작성되었습니다.

최종수정 15-12-30

Introduction

음....어쩌다보니까 서론까지 쓰게 되네요. 가볍게 쓸게요. 가볍게 읽어주세요.

꼭 읽어주세요. 서론을 읽어야 목차와 흐름이 이해가 될 겁니다... 부제목을 길게도 써놔서....

이 문서는 Unity Engine 을 처음 다루는 초심자들에게, Unity 에 대한 기본 개념을 심어주기 위해서 작성되었습니다

만..... 검수 받아보니 초심자에겐 어려울 수 있다고 하네요. 으앙. 이해가 안 된다 하는 부분은 유니티 자습서, API 문서와 연계해서 봐주세요.

이 문서는 총 5 회차로 나누어져 있습니다. 각 회차에서 다루는 내용은 아래와 같습니다.

1 회차 : 유니티 엔진의 기본 개념인 MonoBehaviour, GameObject, Component 에 대한 이해

2 회차 : Component 적용 예시와 Script 를 통한 참조

3 회차 : 유니티 특징 중 하나인 계층 구조(Hierarchy)에 대한 이해와 간단한 응용

4 회차 : Scripting(coding)을 위한 MonoBehaviour 심층 이해와 키보드 Input 처리

5 회차 : 유니티에서의 GUI (Graphical User Interface) 이해

1 회차부터 5 회차까지 꼭 따라가고 나면 **GameObject 와 Component 에 대한 관계, 스크립트를 통한 오브젝트 참조, Input 제어 방법, 물리 엔진의 기초 사용법, GUI 의 기본**에 대해서 감을 잡을 수 있을 겁니다.

또, 각 회차별로 예제와 스크립트를 달아두었습니다. 예제들을 하나씩 따라가며 만들어보면, 어느새 프로젝트 폴더엔 **"Player 가 밟으면 Scene 을 바꾸는 Portal", "키보드 입력을 받아 이동하는 Player", "클릭되면 Scene 이 바뀌는 스테이지 선택 버튼"** 이 생겨있을 겁니다!

이 예제들과 문서로부터 얻은 지식을 토대로 개발을 진행하면, 물리 엔진을 이용한 간단한 퍼즐게임을 혼자서도 충분히 제작할 수 있습니다.

그 지식들 써먹으라고 마구 굴릴겁니다. 실전이라고 생각하고 즐겁게 알려주세요. :)

심화 개념과 보충 개념은 각각 붉은색, 초록색 상자로 묶었습니다. 유니티에 어느 정도 익숙해져 있다면, 읽어보는 걸 추천드립니다. 무려 승직선배가 해주신 유니티 엔진 비밀이야기가..! 유니티를 한 번도 써 본적이 없는 초심자의 경우 잠시 pass 해주세요. 몰라도 개발에는 지장이 없을 뿐더러, 오히려 혼란을 느낄 수도 있습니다.

유니티 많이 사랑해주세요. 좋은 엔진이에요.

관련 자료 리서칭에 도움 주신 제노선배와 검수해주신 7 년 위 뽀선배 승직옹, 두 분께 감사합니다. 항상 고맙습니다. 양은정.

Contents > >

0회차 (12/31) Overview : Unity Tool

- Scene, Game, Project, Hierarchy, Inspector, console window, Play 버튼

1회차 (12/31) Component : GameObject와 MonoBehaviour, Inspector, 자주 쓰이는 Component들, Unity API Reference

- MonoBehaviour와 GameObject의 관계
- GameObject와 Component의 관계
- Component : Transform, Renderer, Collider, Rigidbody, Custom Component

2회차 (1/1) MonoBehaviour : Collision & Trigger

- Collision : Collider, Rigidbody, 그리고 Collision의 조건
- Collision & Rigidbody Example : Plane (3D)
- Trigger : Trigger의 개념
- MonoBehaviour : OnCollision, OnTrigger (3D and 2D)
- Trigger Example : Portal

3회차 (1/2) Hierarchy : Transform, Child, Parent, Finding Game Objects

- Hierarchy에 대한 이론 : Root, Child, Parent
- Script에서 어떻게 해당 GameObject의 Child에 접근할까? : Transform
- Hierarchy example : Tile의 Area 찾기

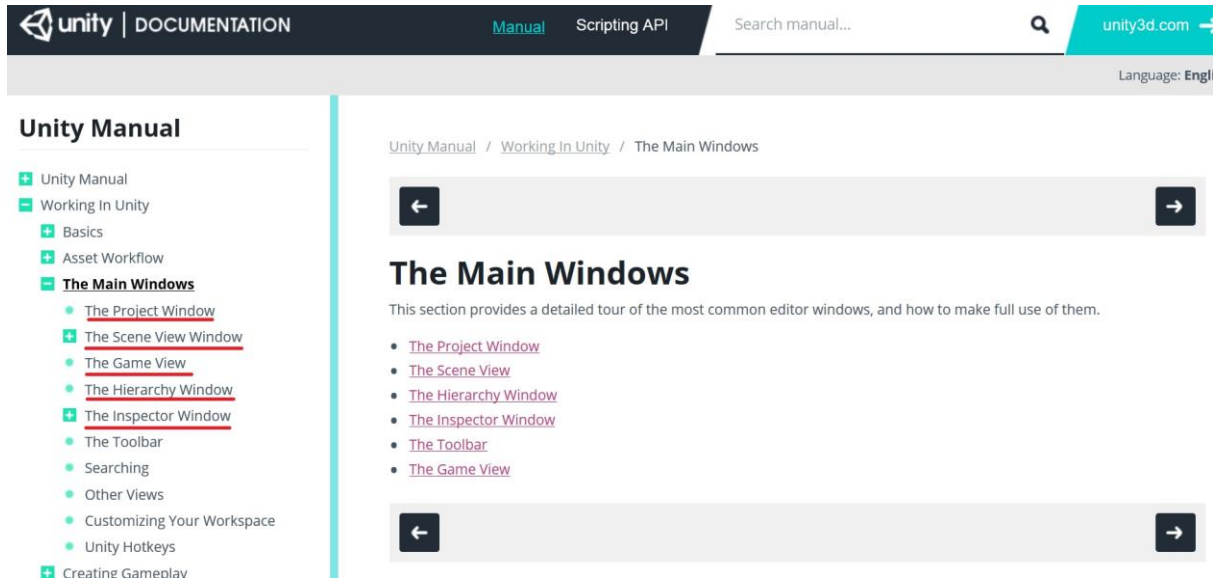
4회차 (1/3) MonoBehaviour : Start(), Update(), Handling Input Event with script

- MonoBehaviour : Start(), Update()
- Message : SendMessage()와 Message가 쓰이는 이유
- MonoBehaviour Update example : Player Movement

5회차 (1/4) GUI : World, Screen, Camera, Canvas, Transform, Rect Transform, OnClick() Event

- World와 Screen : 실제 게임 내부와 화면에 보이는 게임
- Canvas : GUI의 특수한 형태
- Transform, Pixel Resolution, Rect Transform,
- UI Example : Select Stage Button
- Event : OnClick()

0회차 (12/31) Overview : Unity Tool



<http://docs.unity3d.com/Manual/UsingTheEditor.html>

Unity Manual의 Working In Unity / The Main Windows 항목 (링크) 에서 자세한 설명을 볼 수 있어요. 영어판이 더 설명이 좋습니다.

- Scene : 우리가 만드는 실제 게임 내부(3D world space)에 대한 Window
- Game : Player(조작자)가 보는 화면에 대한 Window. 보통 Main Camera가 비추는 곳을 본다.
- Project : Unity Project Folder에 대한 Explorer. (Project Folder)/Asset 폴더를 보여준다.
- Hierarchy : 현재 Scene 내부에 존재하는 모든 GameObject를 계층별로 보여주는 Window.
- Inspector : 선택한 Object의 요소(Component), 또는 유니티 엔진의 속성(preference)을 관리하는 Window.
- Console : Log 또는 Error를 출력하는 Window.
- (Editor) Play : 버튼을 누르면 Editor내에서 게임이 Play된다. Game Window를 통해 확인 가능.

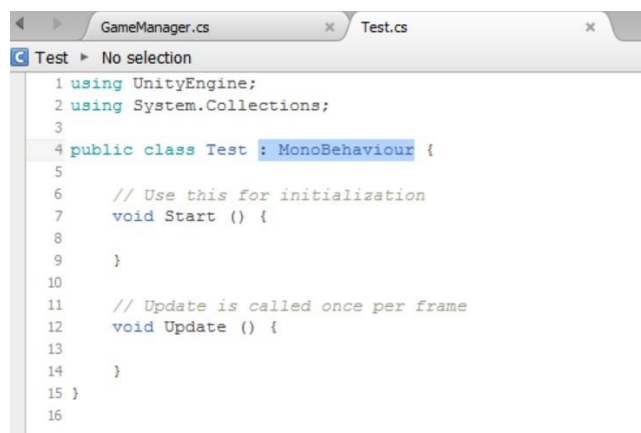
1회차 (12/31) Component : GameObject와 MonoBehaviour, Inspector, 자주 쓰이는 Component들, Unity API Reference

유니티 엔진을 구성하는 GameObject와 Component의 관계, 그리고 Component 중 스크립트인 MonoBehaviour를 간단히 이해해보자.

- MONOBEHAVIOUR

MonoBehaviour is the base class every script derives from. (<http://docs.unity3d.com/ScriptReference/MonoBehaviour.html>)

MonoBehaviour 클래스는 모든 스크립트가 기본적으로 상속받는 베이스 클래스이다. Unity 내부에서 Script를 만들면, 해당 스크립트의 클래스는 MonoBehaviour를 상속받은 채로 만들어진다. 이게 대체 뭐길래?



```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Test : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15 }
16
```

C#에서, (Class A) : (Class B) 란 A가 B를 상속받음을 의미한다.

우리가 말하는 "Game"이란, 매 frame마다 계산과 렌더링을 해서 연속적인 '것처럼' 보이게 한 것이다.

예를 들어서, 플레이어가 d키를 누르면 오른쪽으로 이동하는 게임을 만든다고 하자. d키를 누르는 동안 계속 옆으로 가는 것처럼 보이는 것은 사실 매 frame마다 d키가 눌렸는지 체크하여, d키가 눌렸다면 일정 속도만큼 모델의 좌표를 '바꾸는' 처리를 한 것이다. 실제로는 이산적인 계산의 결과물이지만 플레이를 하면 연속적으로 움직이는 것처럼 보이는데, 그 이유는 frame과 frame 사이의 간격이 매우 짧기 때문이다. 동영상의 재생되는 원리와 비슷하다.

그래서, 실시간으로 변화되어야 할 필요가 있는 오브젝트를 다루려면 프레임 단위로 실행되는 어떤 것이 필요하다. C에서 main 문 안에 while문을 만들어서 프레임 단위로 while문 안의 함수를 계속 실행했다면, 유니티에서는 그 역할을 MonoBehaviour의 Update()가 대신한다.

(Start()와 Update()에 대한 설명은 4회차에서 자세히 >>)

기본적으로 script를 만들면 MonoBehaviour를 붙여주는 이유이다. 만약 실시간으로 계산을 해야 할 필요가 없는 클래스(e.g. 데이터 저장용)는 MonoBehaviour의 Update를 쓸 필요가 없으므로, 클래스 이름 뒤의 ': MonoBehaviour'를 지워서 상속을 끊어도 된다.

- MONOBEHAVIOUR와 GAMEOBJECT, 그리고 COMPONENT

MonoBehaviour

Motion

MovieTexture

NavMesh

NavMeshAgent

NavMeshHit

NavMeshObstacle

NavMeshPath

NavMeshTriangulation

Network

NetworkMessageInfo

NetworkPlayer

NetworkView

NetworkViewID

Inherited members

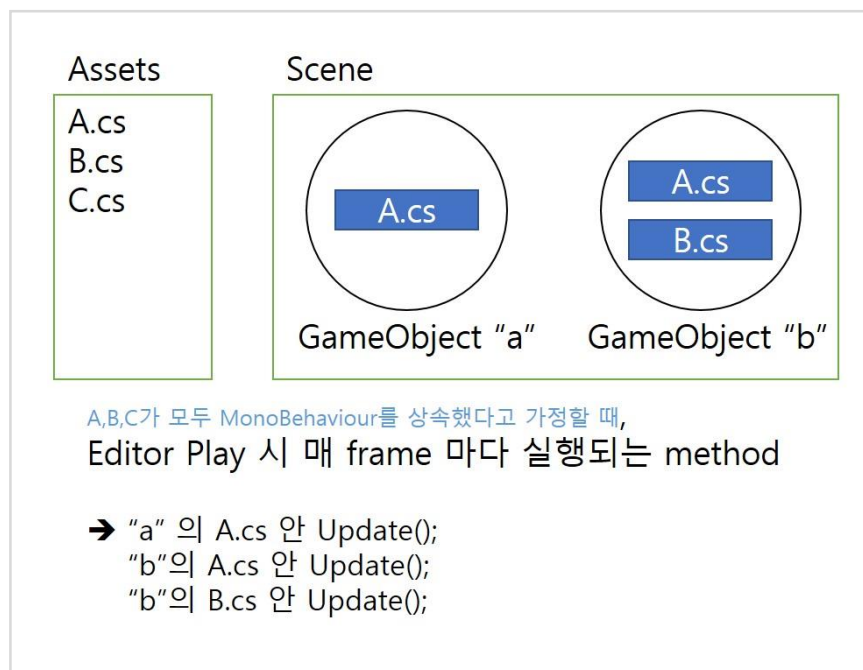
Variables

enabled	Enabled Behaviours are Updated, disabled Behaviours are not.
isActiveAndEnabled	Has the Behaviour had enabled called.
gameObject	The game object this component is attached to. A component is always attached to a game object.
tag	The tag of this game object.
transform	The Transform attached to this GameObject (null if there is none attached).
hideFlags	Should the object be hidden, saved with the scene or modifiable by the user?
name	The name of the object.

Variable - `gameObject` : The game object this component is attached to. A component is always attached to a game object.

유니티 엔진에서, Script는 그 자체로 Scene 내부에 존재할 수 없다. Scene 내부에 살아있는 것은 GameObject들이며, Script는 이 GameObject에 붙어있어야만 Scene 내부에 관여할 수 있다.

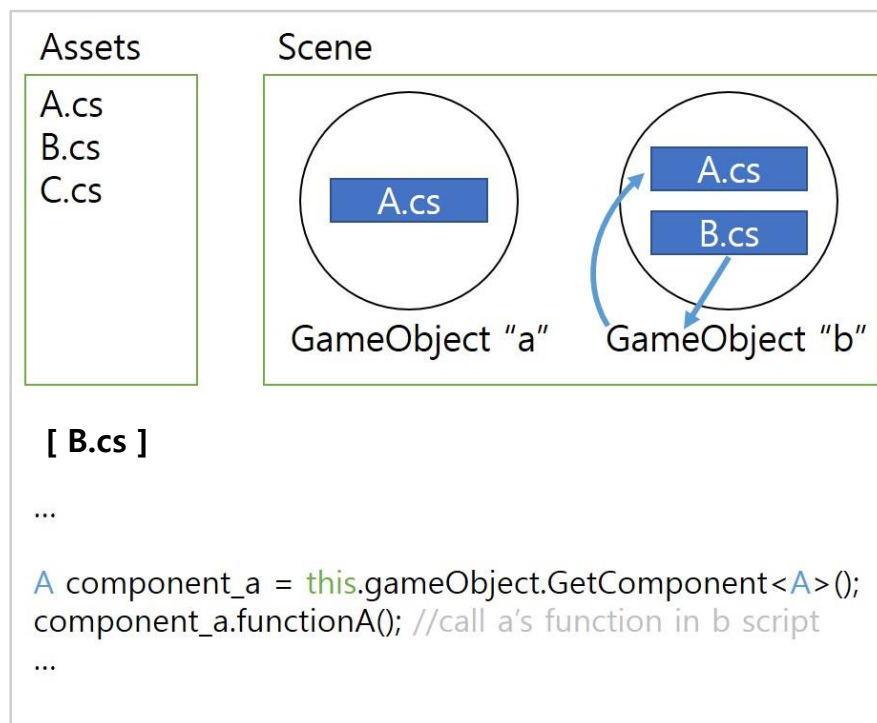
Script는 GameObject에 "Component"의 형태로 붙어있을 수 있다.



Assets에 A, B, C 스크립트를 만들었다고 하더라도 실제 Scene 내부 GameObject들에 Component로서 존재하는 스크립트는 A.cs, B.cs 뿐이므로 C 스크립트는 작동하지 않는다.

따라서 MonoBehaviour를 상속받은 모든 스크립트는 어떤 gameObject에 붙어있음을 전제로 한다. 즉, MonoBehaviour를 상속받은 script에서 gameObject 라는 멤버변수는 런타임(프로그램이 실행되는 동안)에서는 절대로 null이어서는 안 된다.

이를 이용하여 Script에서 GameObject를 참조하는 것도 가능하고, 동일한 Game Object에 붙어 있는 다른 Component에 접근하는 것 역시 가능하다. MonoBehaviour를 상속받은 모든 Script가 유일한 GameObject를 가지고 있음이 보장되기 때문에 역으로 gameObject로부터 Script를 찾을 수도 있는 것이다.



```
this.gameObject.GetComponent<A>();
```

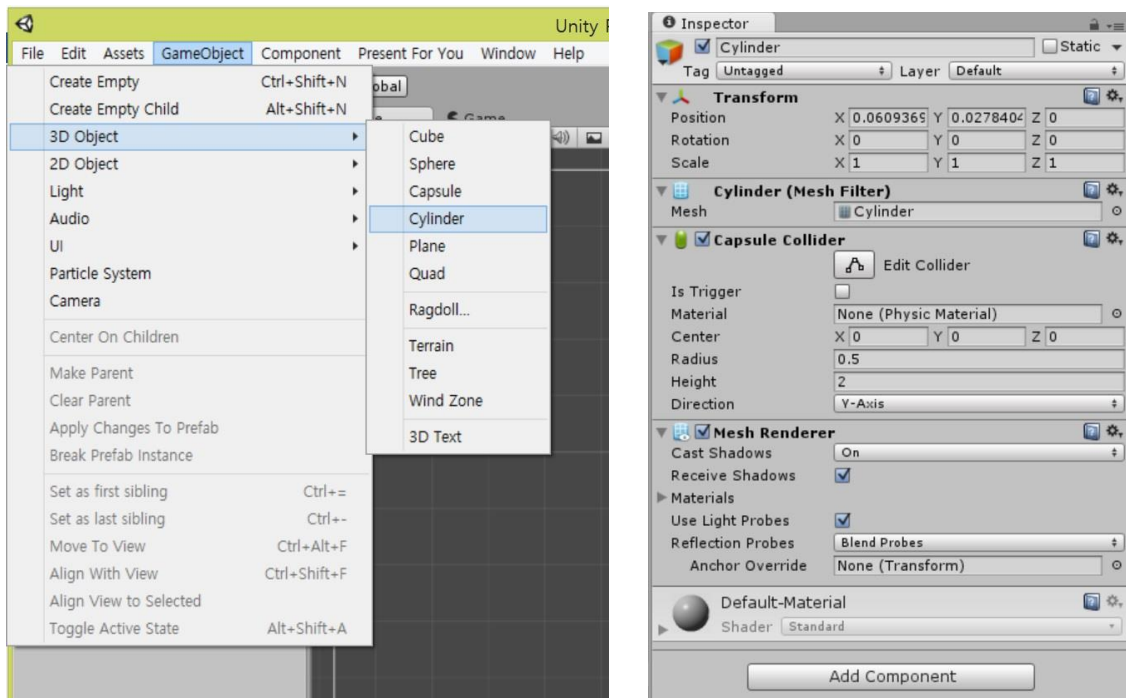
(이 클래스(B) 가 붙어있는 gameObject에 붙어있는 A라는 이름을 가진 Component(=Script=class A)를 가져온다.)

이런 것도 가능하다!

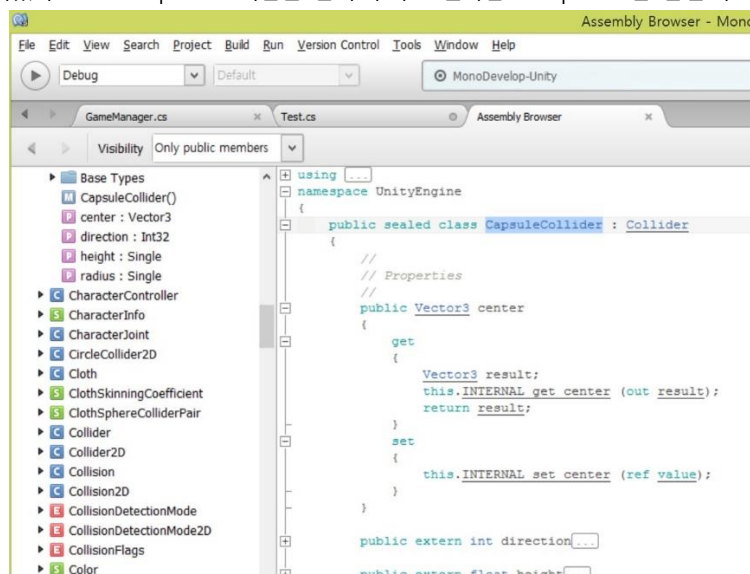
- GAMEOBJECT와 COMPONENT

GameObject란 Scene 내부에서 실제로 존재하는 개체를 말한다. GameObject 자체로는 Scene 내부에 존재하는 것밖에 기능이 없지만, "Component"를 GameObject에 추가함으로써 해당 Script가 작동하게 한다.

유니티에서 기본적으로 제공하는 GameObject들이 몇 종류 있다. 이는 편의를 위해서 빈 GameObject에 특정 역할을 하는 Component들을 미리 붙여놓아 세팅해둔 것이다..



e.g. 3D Object > **Cylinder** : 기본적으로 Transform, Mesh filter, Capsule Collider, Mesh Renderer 네 가지의 Component가 붙어있다. Add Component 버튼을 눌러 추가로 원하는 Component를 붙일 수 있다.



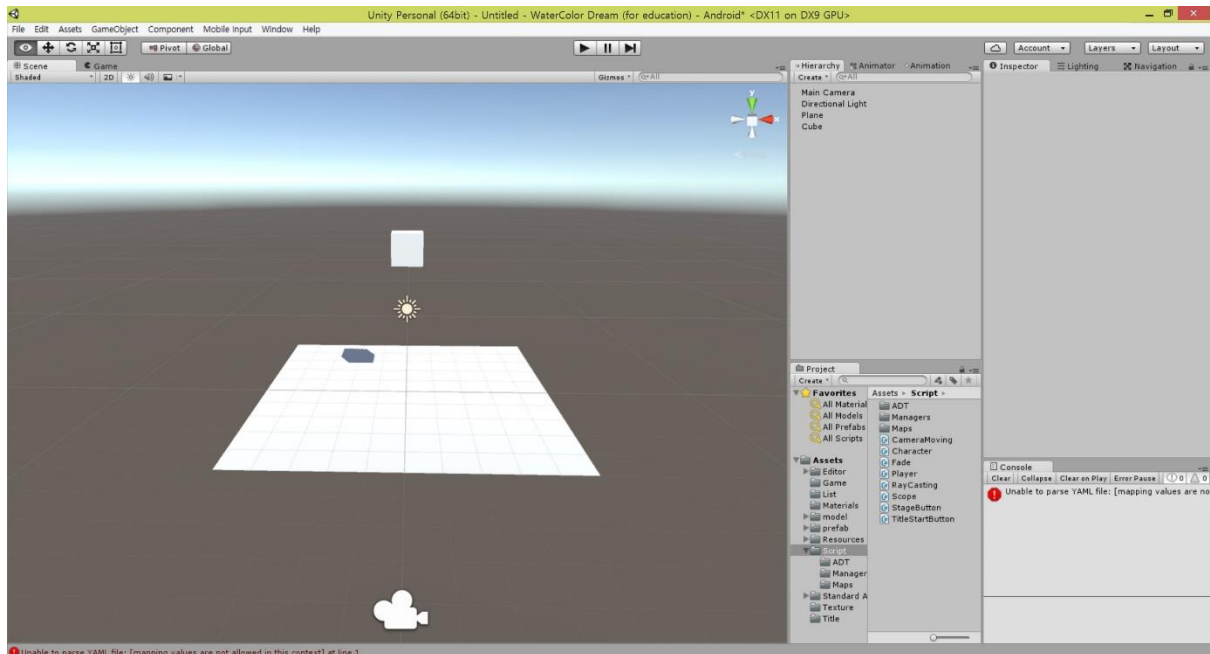
각각의 Component에 해당하는 Script가 유니티엔진 내부에 존재한다. (▲ e.g. Capsule Collider class)

자주 쓰이는 몇 가지 Component를 소개한다. 각 Component가 가진 변수 및 메서드는 Unity API Reference를 참고! Component 이름에 링크 걸어뒀으니 필요할 때 보세요. ~_~

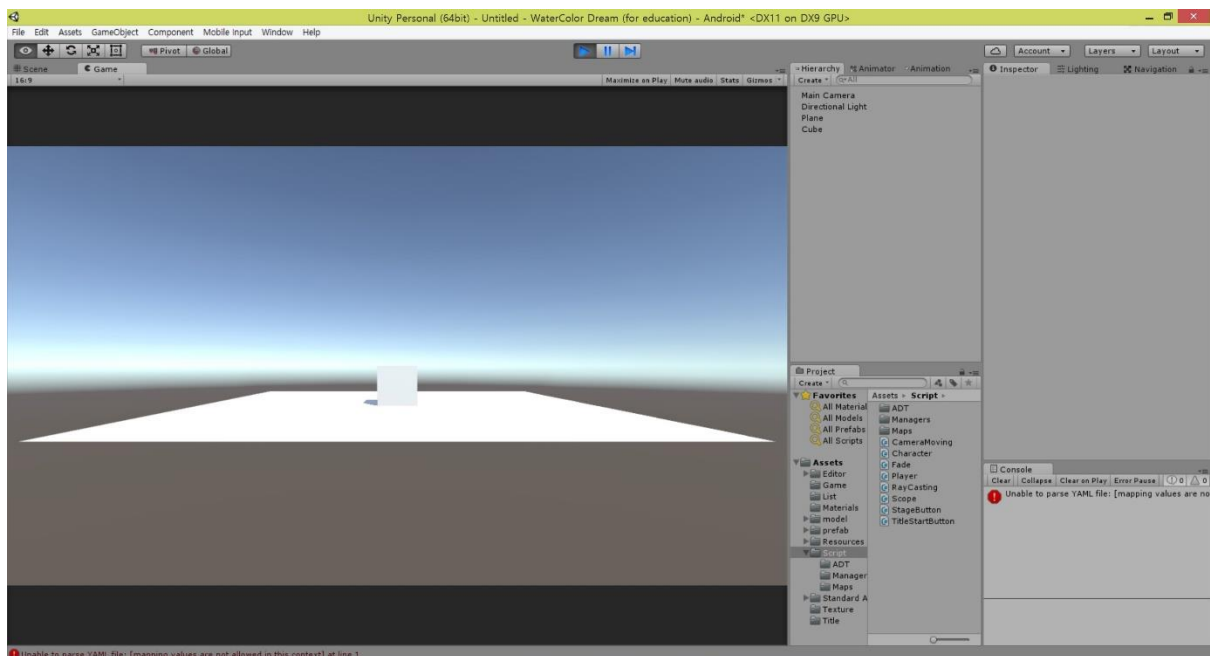
이름	ScreenShot	설명
Transform		해당 GameObject의 Position, Rotation, Scale 값을 float 단위로 조절한다.
Renderer	 <p>Sprite Renderer (2D)</p>  <p>Mesh Renderer(3D)</p> <p>Filter와 Renderer 가 같이 있어야 작동한다.</p>	<p>Target Sprite 또는 Target Mesh를 화면에 “Render” 한다.</p> <p>이 Component가 없으면 화면상에서 형체가 보이지 않는다.</p>
Collider	 <p>Collider 2D</p>  <p>Collider (3D)</p>	<p>충돌 범위를 지정하는 Component.</p> <p>Rendering된 모습과 관계 없이 실제로 충돌하는 범위를 설정할 수 있다.</p> <p>기본적으로 3D이며, 2D의 경우에는 Collider2D 컴포넌트가 따로 있다.</p>
RigidBody	 <p>Rigidbody 2D</p>  <p>Rigidbody (3D)</p>	<p>물리연산을 해주는 Component.</p> <p>이 Component가 붙은 GameObject에는 물리가 적용된다.</p> <p>중력, 마찰, 질량 등의 파라미터를 가진다.</p> <p>Rigidbody는 Collider와 대응되고, Rigidbody2D는 Collider2D와 대응된다.</p>

1 회차 과제)

스샷 1 처럼 Plane (position = (0,0,0)), Cube (position = (0,5,0)) 게임오브젝트를 하나씩 만들고 Editor Play 버튼을 눌렀을 때 스샷 2 처럼 Cube 가 중력을 받아 Plane 으로 떨어지게 만들어본다!



(스샷 1 : Play 전)



(스샷 2 : Play 중)

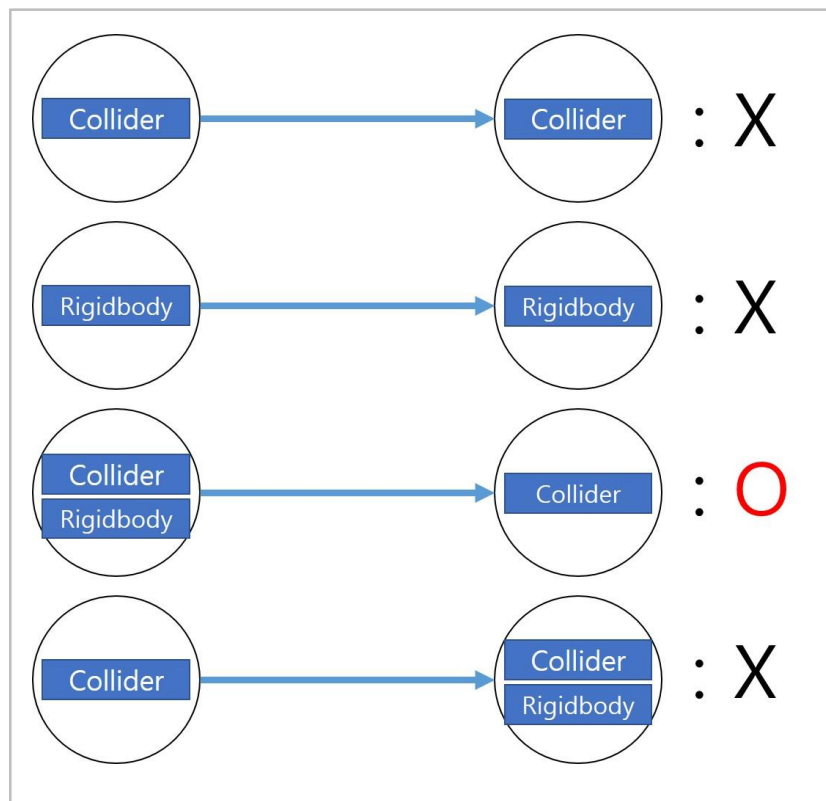
2회차 (1/1) Monobehaviour : Collision & Trigger

"Collision&Trigger" 예시를 통해 1회차에서 설명했던 Component를 직접 사용해보자. 바로 확인하기 좋은 물리 엔진을 이용하며, MonoBehaviour에서 어떻게 반응하는지도 알 수 있다.

- COLLISION : COLLIDER, RIGIDBODY

충돌을 만들기 위해선 Collider, Rigidbody가 필요하다. Collider는 충돌을 인식하는 '범위'를 지정하며, Rigidbody는 Collider로부터 충돌을 만들어낸다.

Monobehaviour에서 충돌을 만들어내기 위해서, 충돌하는 주체(때리는 쪽)와 객체(맞는 쪽)에 모두 Collider가 있어야 하고 주체에 Rigidbody가 있어야 한다.



Rigidbody를 가진 GameObject가 정지 상태이면 충돌이 감지되지 않는다.

- COLLISION & RIGIDBODY EXAMPLE : PLANE (3D)

1회차 과제가 Collision의 좋은 예시이다!

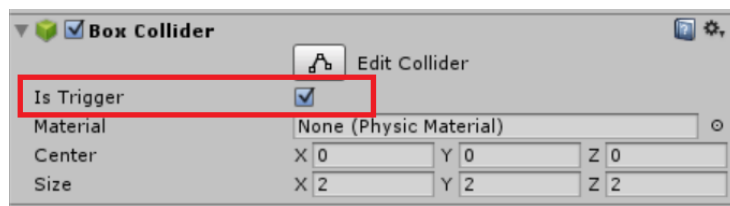
각각 Collider를 가지고 있는 Cube와 Plane 중, 움직이는 GameObject인 Cube에 Rigidbody를 붙이면 Cube는 Plane위에 안착한다. Cube와 Plane 두 군데에 모두 Rigidbody Component를 추가해도 충돌이 일어난다. (Unity5에선 Plane에 Rigidbody를 붙이려면 Rigidbody Component를 추가 후 Inspector에서 is Kinematic에 체크를 해야 한다.) 하지만, Plane에만 Rigidbody를 붙여놓고 Play 후 Cube GameObject를 Scene window에서 움직이면 Plane을 통과해버린다. 충돌이 일어나지 않는다.

- TRIGGER

Trigger란 “계기, 총의 방아쇠” 라는 뜻이다. 여기서는 ‘특정 범위 안에 들어오거나 특정 범위를 빠져나왔을 때’ 등, 범위에 관련된 계기’라는 뜻으로 이해하자. 뒤에 나올 2회차 예제를 보고 감을 잡자.

Collision이 두 물체의 충돌로 인한 물리적 처리(튕겨나간다거나)를 한다면, Trigger는 튕겨나가는 등의 후처리가 없다는 점이 Collision과 다르다.

Collider를 통해 Trigger를 유발하는 zone인 Trigger zone을 설정할 수 있다. Collider는 ‘범위’를 지정하는 Component이므로, Inspector에서 is Trigger에 체크하면 Collider가 가진 범위가 고스란히 Trigger의 범위가 된다. Trigger 역시 Collision처럼 충돌 주체에 Rigidbody가 있어야 감지된다.



- MONOBEHAVIOUR : ONCOLLISION, ONTRIGGER

MonoBehaviour에서는 이러한 Collision과 Trigger에 대해서 Script로 관리할 수 있게 다음과 같은 Message를 지원한다.

OnCollisionEnter(2D), OnTriggerEnter(2D) ,OnCollisionStay(2D) , OnTriggerEnterStay(2D),
OnCollisionExit(2D) , OnTriggerEnterExit(2D)

Message는 MonoBehaviour Class에서 정의된 Method가 아니므로 Override 할 필요 없이 그냥 정의하면 된다. (개발을 위해선 이 정도까지만 알아도 문제가 없다. 메시지에 대한 자세한 설명을 원한다면 4회차 참고)

```
// A grenade
// - instantiates a explosion prefab when hitting a surface
// - then destroys itself

using UnityEngine;
using System.Collections;

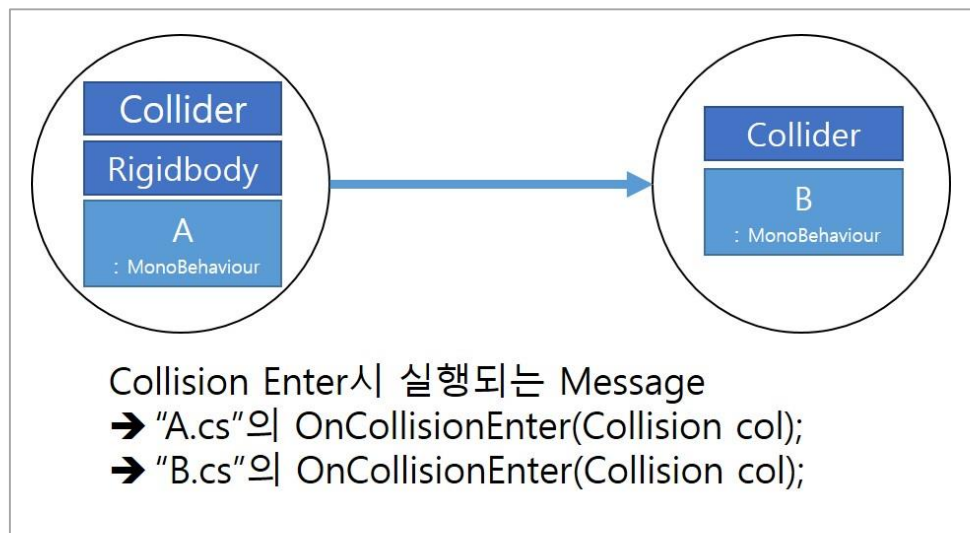
public class ExampleClass : MonoBehaviour {

    public Transform explosionPrefab;

    void OnCollisionEnter(Collision collision) {
        ContactPoint contact = collision.contacts[0];
        Quaternion rot = Quaternion.FromToRotation(Vector3.up, contact.normal);
        Vector3 pos = contact.point;
        Instantiate(explosionPrefab, pos, rot);
        Destroy(gameObject);
    }
}
```

OnCollisionEnter 예제. <http://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html>

OnCollision 계열 메시지는 파라미터로 Collision 타입을 받고, OnTrigger 계열 메시지는 파라미터로 Collider 타입을 받는다. Collision은 충돌 지점의 좌표 등 Trigger 처리에선 안 쓰이는 몇 개의 변수와 메서드를 더 가지고 있다는 차이가 있다.



두 GameObject 사이에 Collision이 발생하면, 두 GameObject 내부의 OnCollision 메시지가 모두 실행된다.

OnCollision(OnTrigger) 메시지는 호출될 때 상대방에 대한 정보를 파라미터로 가지고 있으며, 이를 이용하여 함수 내에서 충돌한 상대방에 대해 접근할 수 있다. 예를 들면, GameObject A와 B가 충돌했을 경우, 호출되는 A의 OnCollision 메시지에 파라미터로 들어가는 Collision엔 B의 Collider 정보가 포함되어 있고, B의 OnCollision 메시지에 파라미터로 들어가는 Collision엔 A의 Collider 정보가 들어간다. 2회차 예제의 예시 코드를 확인하자.

기본적으로 Rigidbody와 Collider가 3차원 지원이므로 OnCollision, OnTrigger 역시 3차원 관련 메세지이다. 만약 2D 충돌 또는 2D 트리거를 스크립트로 다룬다면 메시지 이름으로 OnCollision(state)2D와 OnTrigger(state)2D를 써야 하며 파라미터로 각각 Collision2D와 Collider2D를 받는다

- TRIGGER EXAMPLE : PORTAL

“Player” 라는 tag를 가진 GameObject가 해당 오브젝트의 Trigger를 작동하면 Scene을 바꾸는 함수를 실행하게끔 하는 Portal class를 짜보자. 고려할 점은 다음과 같다.

- 1) 플레이어는 Tile 안으로 들어올 수 있어야 하므로 Collision이 아닌 Trigger를 사용한다.
- 2) OnTrigger Message를 사용해야 하므로 이 클래스는 MonoBehaviour를 상속받아야 한다.
- 3) Tag가 “Player”인 GameObject가 트리거를 작동시켜야만 함수가 실행되어야 한다.

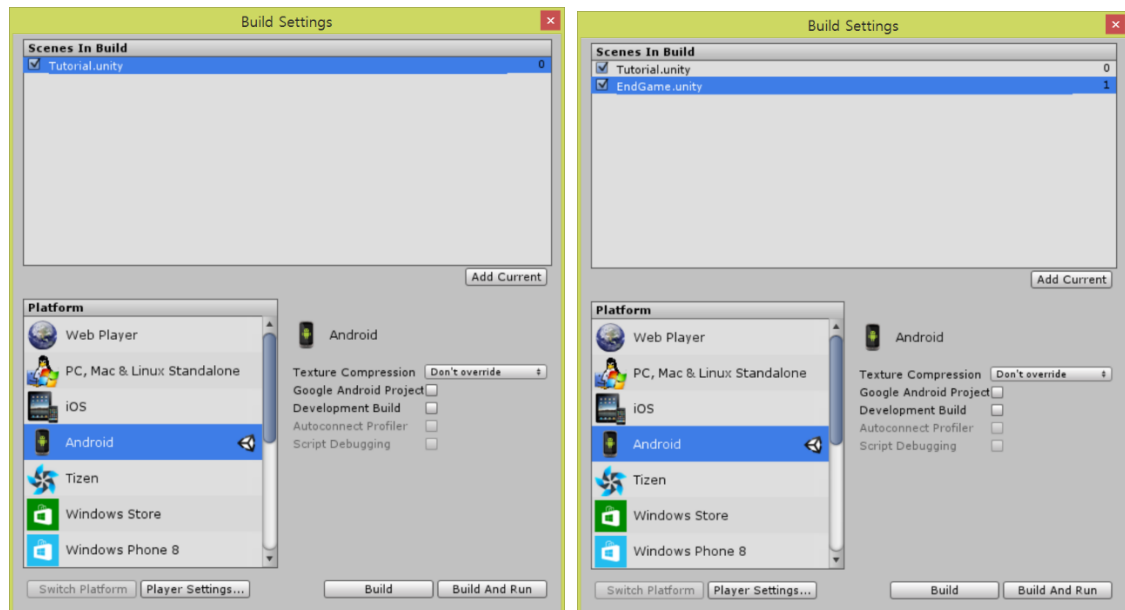
예시 코드는 다음과 같다.

```
C Portal ▶ OntriggerEnter (Collider col)
1 using UnityEngine;
2 using System.Collections;
3
4 public class Portal : MonoBehaviour {
5
6     void OnTriggerEnter(Collider col){
7
8         //만약 Trigger를 일으킨 collider가 붙어있는 GameObject의 Tag가 "Player"라면
9         if (col.tag == "Player") {
10
11             //EndGame 이라는 이름을 가진 scene을 Load한다.
12             Application.LoadLevel("EndGame");
13
14             Debug.Log ("Portal Trigger Enter :: Scene Change!");
15         }
16     }
17 }
18 }
19 }
```

비어있는 Scene에 Cube를 두 개 만들고, 떨어질 큐브에 Rigidbody Component를 추가한 뒤 떨어지는 큐브가 부딪칠 큐브엔 Portal Component를 추가해보자. Rigidbody Component가 붙은 큐브는 Inspector에서 “Player” tag를 달아주자. Portal Component가 붙은 큐브의 Collider 설정을 Trigger로 바꾸고, Collider size를 x, y, z 각각 2로 설정하자.

그리고 플레이를 눌러 두 큐브를 충돌시켜보자.

만약 "EndGame"이라는 이름의 Scene을 만들었음에도 12번째 줄에서 Error가 났다면, 아래와 같이 File > Build Settings 에서 EndGame 이라는 Scene을 새로 추가하고 Play를 다시 눌러보자.



Trigger가 작동해서 Scene이 이동함을 확인할 수 있다.

2회차 과제)

OnCollision 계열 message를 사용해서 아무거나 하나 만들어보자!

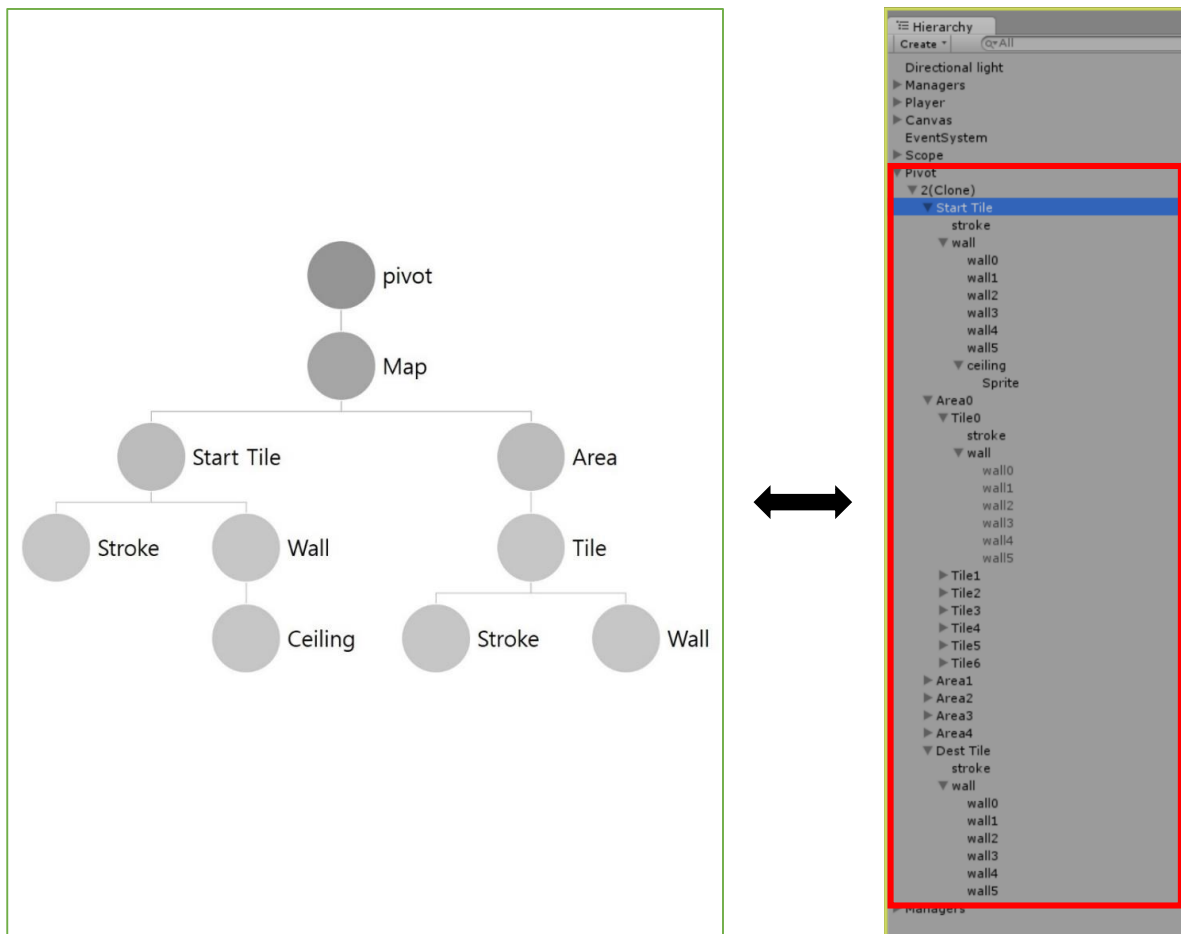
그리고, Play를 돌려봐서 잘 되나 확인해보자!

3회차 (1/2) Hierarchy : Transform, Child, Parent, Finding Game Objects

Scene에 GameObject가 어떻게 배치되는지를 알아보고, 계층 구조를 접근하기 위해 Transform를 사용해본다.

- HIERARCHY ; 계층 구조

Hierarchy window는 현재 Scene에 존재하는 모든 GameObject를 가지고 있다. 특징은, "계층 구조"로 GameObject들을 나타낸다는 것이다.



ViewTree (좌)와 Hierarchy Window (우) (WaterColor Dream, in-game)

유니티의 Hierarchy는 Tree 자료구조와 유사하다. Tree는 node로 구성되어 있으며 child node나 parent node를 가질 수 있듯이, Hierarchy는 Game Object, 정확히는 transform으로 구성되어 있으며 Child transform이나 parent transform을 가질 수 있다. Tree의 최상위 node를 root라고 하듯, 현재 자신이 위치한 계층의 최상위 transform을 root라고 한다.

- SCRIPT에서 어떻게 해당 GAMEOBJECT의 CHILD에 접근할까? : TRANSFORM CLASS

1회차 (Component)에서, Transform은 해당 GameObject의 위치, 오일러각도, 크기를 관리하는 Component라는 것을 알았다. 그렇다면 Hierarchy에서 말하는 Transform과 Transform Component는 다른 것일까?

Texture3D
Time
Touch
TouchScreenKeyboard
TrailRenderer
Transform
Tree
TreeInstance
TreePrototype
UICharInfo
UILineInfo

Transform

class in UnityEngine / Inherits from: [Component](#)

[SWITCH TO MANUAL](#)

Description

Position, rotation and scale of an object.

Every object in a scene has a Transform. It's used to store and manipulate the position, rotation and scale of the object. Every Transform can have a parent, which allows you to apply position, rotation and scale hierarchically. This is the hierarchy seen in the Hierarchy pane. They also support enumerators so you can loop through children using:

Transform API 문서를 읽어보자. <http://docs.unity3d.com/ScriptReference/Transform.html>

Transform은 오브젝트의 Position, Rotation, Scale이다. Scene은 3차원 공간이므로, Scene에 존재하는 모든 Object는 (당연히) transform을 가지고 있다. MonoBehaviour와 GameObject 사이의 관계처럼, GameObject와 Transform 역시 1:1 대응이 되어있는 셈이고, 그래서 Unity Engine에서는 이 유일성을 이용해서 Hierarchy를 표현한 것으로 보인다. 각각의 Transform을 GameObject으로 치환할 수 있기 때문이다.

1차시에서, Component는 Script가 GameObject에 붙어있는 방법이므로 각각의 Component에 해당하는 Script가 존재한다고 했다. (6p) 즉, Transform 이라는 Script는 기본적으로 해당 GameObject의 Position, Rotation, Scale 값을 가지고 있는 동시에 (이 값들은 Inspector에 노출된다), Hierarchy에 관련된 것들도 포함하고 있다는 것이다(이 값들은 Inspector에 노출되지 않는다). Hierarchy의 Transform과 Inspector의 Transform은 같은 Class이다.

Transform Class를 이용하면 Script 내에서 해당 GameObject의 child나 parent, grandparent, root에 쉽게 접근할 수 있다.

Microphone
MonoBehaviour
Motion
MovieTexture
NavMesh
NavMeshAgent
NavMeshHit
NavMeshObstacle
NavMeshPath
NavMeshTriangulation
Network
NetworkMessageInfo
NetworkPlayer

Inherited members

Variables

enabled	Enabled Behaviours are Updated, disabled Behaviours are not.
isActiveAndEnabled	Has the Behaviour had enabled called.
gameObject	The game object this component is attached to. A component is always attached to a game object.
tag	The tag of this game object.
transform	The Transform attached to this GameObject (null if there is none attached).
hideFlags	Should the object be hidden, saved with the scene or modifiable by the user?
name	The name of the object.

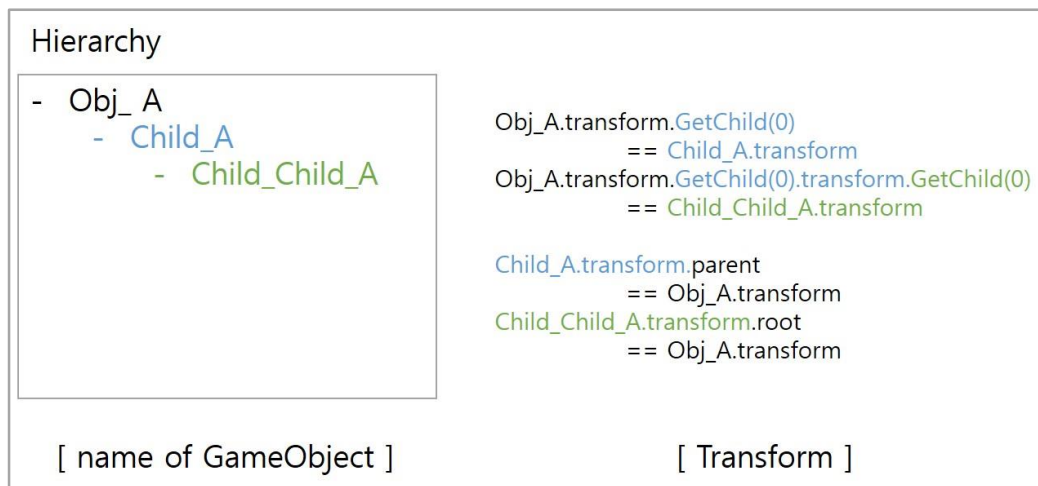
MonoBehaviour API 문서를 읽어보자. <http://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

다시 MonoBehaviour로 돌아오자. MonoBehaviour는 반드시 GameObject를 가지고 있고, GameObject는 반드시 Transform을 가지고 있다.

MonoBehaviour 클래스는 'Behaviour' 클래스를 상속받았다. 그렇기 때문에 Behaviour가 가지고 있던 멤버 변수인 gameObject, tag, transform, name 등등을 고스란히 멤버 변수로 가지고 있다.

이 때 transform 변수는 Transform type으로, 해당 Script가 Component로 붙어있는 GameObject의 Transform을 말한다. 즉, A.cs 라는 MonoBehaviour를 상속받은 스크립트가 obj_A 라는 GameObject의 Component로 붙어있으면, A.cs 안에서 부를 수 있는 transform 이라는 변수는 obj_A의 Transform을 가지고 있는 것이다.

이를 이용하여 현재 자기 자신의 GameObject가 가진 transform으로부터 상위 계층의 GameObject나 하위계층의 GameObject에 접근한다. Transform API 문서를 보면 Transform 역시 멤버 변수로 gameObject를 가지고 있으므로, 찾고자 하는 GameObject의 transform에 접근한 후 gameObject 변수를 사용하면 원하는 GameObject에 쉽게 접근이 가능하다.



Transform GetChild (int index) : 입력받은 index의 Child에 해당하는 Transform을 반환한다.

Child가 여러 개 존재할 수 있기 때문에 메서드로 Child를 찾는다.

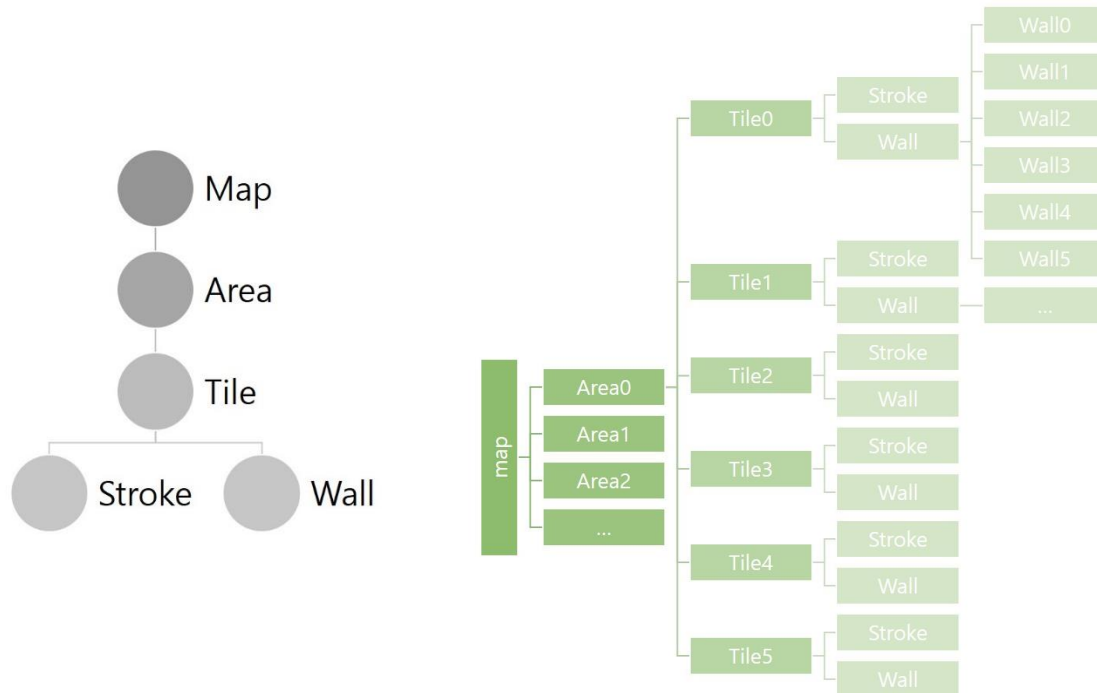
Transform parent (Variable) : 해당 Transform의 Parent Transform을 반환한다.

Transform root (Variable) : 해당 Transform의 Root Transform을 반환한다.

더 많은 메서드와 변수는 Transform API 참고. <http://docs.unity3d.com/ScriptReference/Transform.html>

Transform 클래스를 이용하여, Script 상에서 (Runtime에서) 새로 GameObject를 생성한 후, Hierarchy 상에 등록하거나 제외하는 것도 가능하다. SetParent, SetChild, SetSibling, DetachChild 등의 메서드가 Transform Class 내에 존재한다. API 참고.

- HIERARCHY EXAMPLE : TILE의 AREA 찾기



좌 : ViewTree (Logic) / 우 : Hierarchy Window

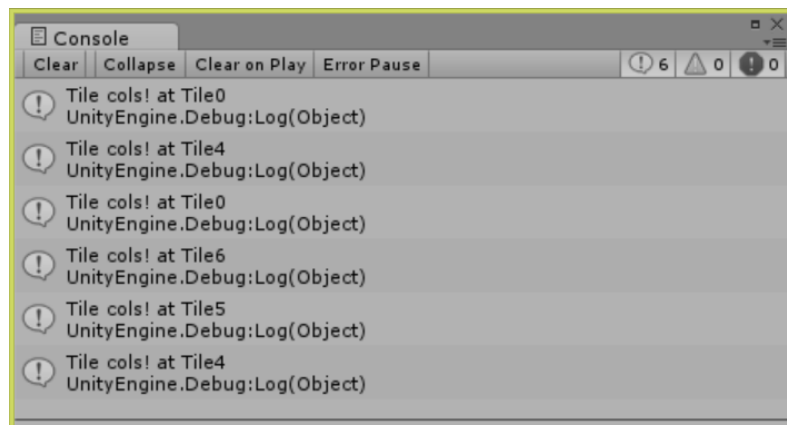
어떤 게임의 맵이 위와 같은 계층 구조를 가진다고 하자. 하나의 Map은 1개 이상의 Area(Area0~)로 이루어져있고, 하나의 Area는 6개의 Tile(Tile0~Tile5)로 이루어져있다.

Player가 충돌한 Tile이 어디인지 알고 싶어서, Tile 클래스에 다음과 같은 OnCollisionEnter() 메시지를 추가했다.

```

8 void OnCollisionEnter(Collision col){
9
10     Debug.Log ("Tile cols! at "+ this.gameObject.name);
11
12 }
```

그랬더니 Console에 다음과 같은 Log가 찍혔다.



Tile0 이라는 이름을 가진 GameObject가 유일하지 않기 때문에 Tile 이름만 가지고는 Player가 어떤 Tile에 충돌했는지 정확히 알기 어렵다. 그래서, Log를 조금 수정하여 Tile의 이름과 이 Tile이 속한 Area의 이름까지 출력하려고 한다. 어떻게 해야 할까?

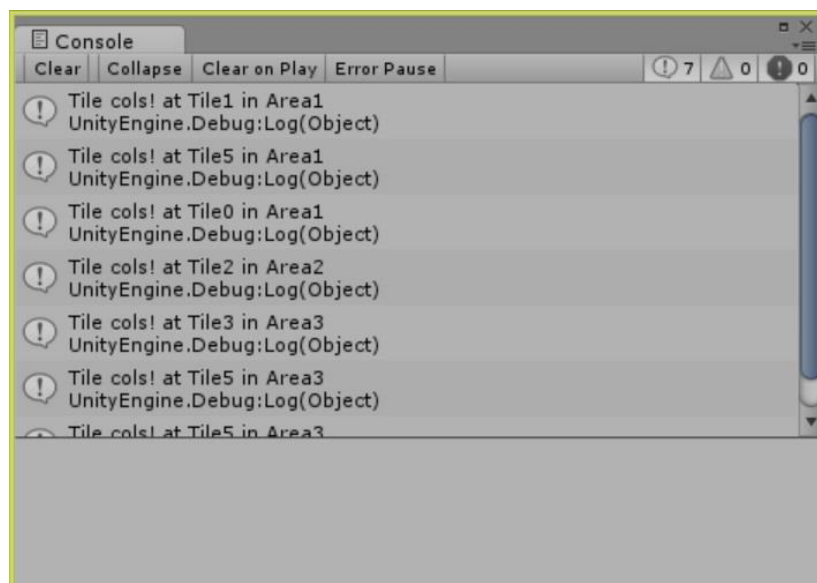
고려할 점은 다음과 같다.

- 1) OnCollision Message를 사용하려면 MonoBehaviour를 상속받아야 한다.
- 2) MonoBehaviour를 상속받은 클래스는 유일한 gameObject와 transform 멤버변수를 갖고 있다.

예시 코드는 다음과 같다

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Tile : MonoBehaviour {
5
6     public Color rgb;
7
8     void OnCollisionEnter(Collision col){
9
10         Debug.Log ("Tile cols! at " + this.gameObject.name + " in " + transform.parent.name);
11
12     }
13 }
14
```

고친 코드로 로그를 찍은 결과는 다음과 같다.



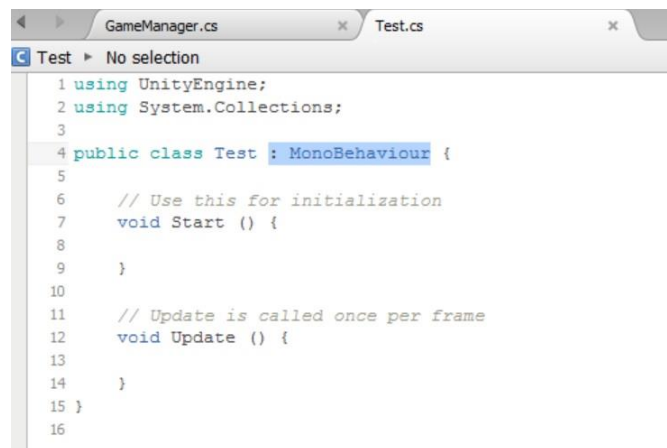
3 회차 과제) 딱히 없음 오늘 내용 이해만 잘 해주세요~!

4회차 (1/3) MonoBehaviour : Message, Start(), Update(), Handling Input Event with script

본격적으로 MonoBehaviour를 사용하기 위해서 중요 함수들을 알아보고, 유니티의 함수 호출 원리를 이해한다.

- MONOBEHAVIOUR : START(), UPDATE()

1회차에서, 매 Frame마다 호출되는 메시지가 MonoBehaviour의 Update(); 라고 했다. 3회차에서, Message는 Class에서 정의된 Method가 아니므로 Override 없이 사용 가능하다고도 했다. 이번엔 MonoBehaviour의 Message중 일부인 Start()와 Update()에 대해서 조금 더 알아보자.



```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Test : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11     // Update is called once per frame
12     void Update () {
13
14     }
15 }
16
```

Script를 만들면 기본적으로 제공되는 form.
MonoBehaviour 상속, void Start(); void Update();

Start()는 해당 Script가 붙은 GameObject instance가 생성될 때 딱 한 번 실행된다. 주로 초기화를 할 때 쓰인다. Update()는, 해당 gameObject가 Enable 상태일 때 매 프레임마다 호출된다. 실시간으로 입력을 처리하거나 혹은 실시간으로 값을 체크해야 하는 경우 Update문을 사용한다.

Awake()라는 메시지가 있다. Awake()는 Start()와 비슷하게 딱 한 번 실행되는데, Awake()는 Scene이 Load될 때 실행되고 Start는 Instance가 생성될 때 실행된다는 차이가 있다. 모바일 어플리케이션에서 홈 화면으로 나가졌다거나 하는 이유로 어플리케이션이 잠시 Pause되었다가 재개될 때에도 Awake()가 호출된다.

LateUpdate()는 Update와 비슷하게 계속 호출이 되지만, Update와는 달리 모든 인스턴트의 Update가 호출된 후에 호출된다는 차이가 있다. Awake와 LateUpdate는 키워드만 알아두고, 개발도중 필요해지면 더 깊이 공부하는 것을 추천한다.

Unity 자습서 참고. <http://docs.unity3d.com/kr/current/Manual/ExecutionOrder.html>

- MESSAGE?

우리가 Script를 짜고 GameObject에 Component로 등록만 해두면, Start()나 Update(), OnCollision() 등의 메서드를 따로 외부에서 호출하지 않아도 특정 조건이 되면 자동으로 메서드가 호출된다. 아니, 정확히는 기본적으로 void Start(); 나 void Update(); 등의 메서드는 접근제한자가 public이 아니기 때문에 클래스 외부에선 호출할 수도 없다!

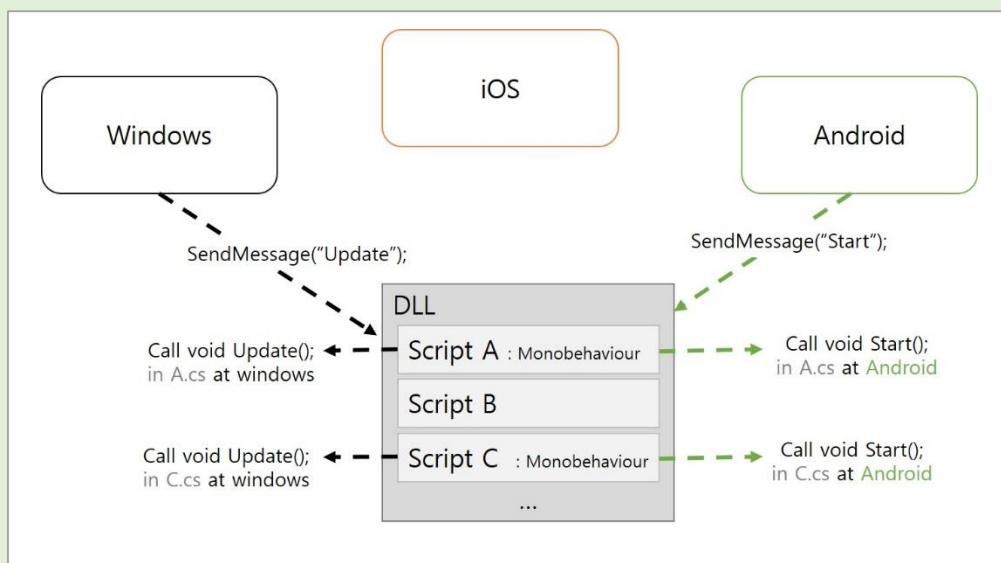
그런데 Editor 플레이를 누르거나 실제로 Build를 하면 이런 함수들이 호출되어 잘 돌아감을 볼 수 있다. 어떻게? “**접근 제한자와 상관없이 이름만 알면 그 함수가 호출될 수 있게**” 해 주는 메서드가 있기 때문이다. 바로 SendMessage(string methodName) 인데, 이 **SendMessage 메서드에 의해 꾸준히 호출되는 특정 메서드의 이름을 정해둔 게 바로 MonoBehaviour가 제공하는 Message**들이다.

그렇다면 대체 왜 SendMessage로 메서드를 호출해야 할까?

유니티의 강점인 “크로스플랫폼 지원”이 어떻게 이루어지는지에 대해 알면 이를 이해할 수 있다.

PC, Mac, Android, iOS 등등의 플랫폼은 각자 다양한 언어로 짜여져 있으며, 언어가 다르면 프로그램을 실행할 수가 없다. C로 .exe build 파일을 만들어도 Java 기반인 Android에서는 이를 실행시킬 수 없는 것과 같다.

그래서 유니티엔진은 이를 해결하기 위해 한 가지 비책을 썼는데, 그게 바로 “Script를 DLL로 빌드하는 것”이다. DLL(Dynamic Linking Library)로 코드가 저장되면 언어에 관계없이 해당 라이브러리 안의 함수를 실행시킬 수가 있다. SendMessage로 MonoBehaviour 내의 기본 메서드(e.g. Update)를 부르는 역할을 유니티 스크립트 자체가 아닌 아닌 각 플랫폼에게 맡김으로써, 언어와 상관 없이 DLL 안의 메서드를 실행시킬 수 있게 되어 플랫폼 간 장벽을 없앴 것이다..



실제로 DLL에 접근해서 함수를 호출하는 역할은 각 플랫폼에게 맡긴다.

메서드의 이름만 알면 SendMessage(String name)로 해당 메서드를 실행할 수 있기 때문에 가능하다.

Assembly 내에서 문자열과 이름이 일치하는 메서드를 찾는 것을 Reflection이라고 한다. 궁금하면 검색 ㄱ

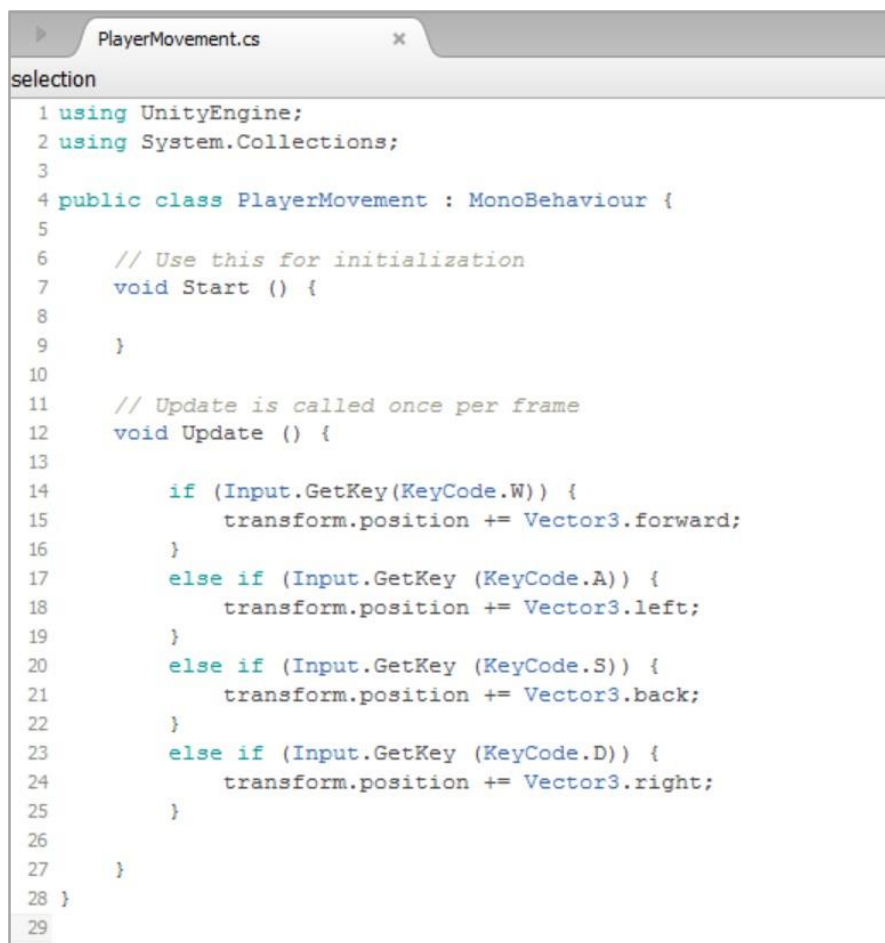
- MONOBEHAVIOUR UPDATE EXAMPLE : PLAYER MOVEMENT

Update()는 매 프레임마다 호출되기 때문에, 주로 실시간으로 값을 체크해야 하는 메서드가 등록된다. W,A,S,D 키보드 입력을 받아 Player를 움직이게 해보자.

고려해야 할 점은 다음과 같다.

- 1) 키보드 입력이 들어오는지를 실시간으로 (매 frame마다) 체크해야 한다.
- 2) 만약 키보드 입력이 들어왔다면, Player의 position을 바꾸어주자.
- 3) W+S, A+D 와 같은 동시 입력은 고려하지 않고 if~ else if문으로 코드를 짜 보자.

예시 코드는 다음과 같다.



```
1 using UnityEngine;
2 using System.Collections;
3
4 public class PlayerMovement : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11     // Update is called once per frame
12     void Update () {
13
14         if (Input.GetKey(KeyCode.W)) {
15             transform.position += Vector3.forward;
16         }
17         else if (Input.GetKey(KeyCode.A)) {
18             transform.position += Vector3.left;
19         }
20         else if (Input.GetKey(KeyCode.S)) {
21             transform.position += Vector3.back;
22         }
23         else if (Input.GetKey(KeyCode.D)) {
24             transform.position += Vector3.right;
25         }
26
27     }
28 }
29
```

Cube GameObject를 하나 만들고 그 Cube에 PlayerMovement Component를 붙이자. 혹시 Hierarchy에 Event System 이라는 GameObject가 없으면 빈 GameObject를 만들고 "Event System" 이라는 Component를 붙이자. Event System이 있어야 Input을 감지할 수 있다.

Editor Play를 누르고 w a s d 를 누르면 Cube가 각 방향에 맞게 이동하는 것을 볼 수 있다.

4회차 과제) PlayerMovement Script를 수정하여, Space input이 들어오는 순간 (GetKeyDown)
Vector3.Up 방향으로 jump되었다가 중력방향으로 다시 떨어져 내려오는 Player를 구현해보자!

5회차 (1/4) Graphical User Interface : World, Screen, Camera, Canvas, Transform, Rect Transform, OnClick()

게임의 중요 요소인 GUI 개발법을 익히고, 이를 위해 World, Screen 좌표계의 차이점을 공부한다.

- User Interface

UI(User Interface)란, User 가 편하게 사용할 수 있게끔 버튼, 텍스트 창 등의 Interface 를 제공해주는 것이다. UI 는 미니맵과 같이 게임에 대한 일부 information 을 제공해주기도 한다. UI 의 종류로는 CUI(Console UI), GUI (Graphical UI) 등이 있으며, 상용게임에서는 주로 GUI 가 쓰인다.



게임 스크린샷(사이퍼즈, 위) 와 해당 스크린샷에서의 UI 들 (아래)

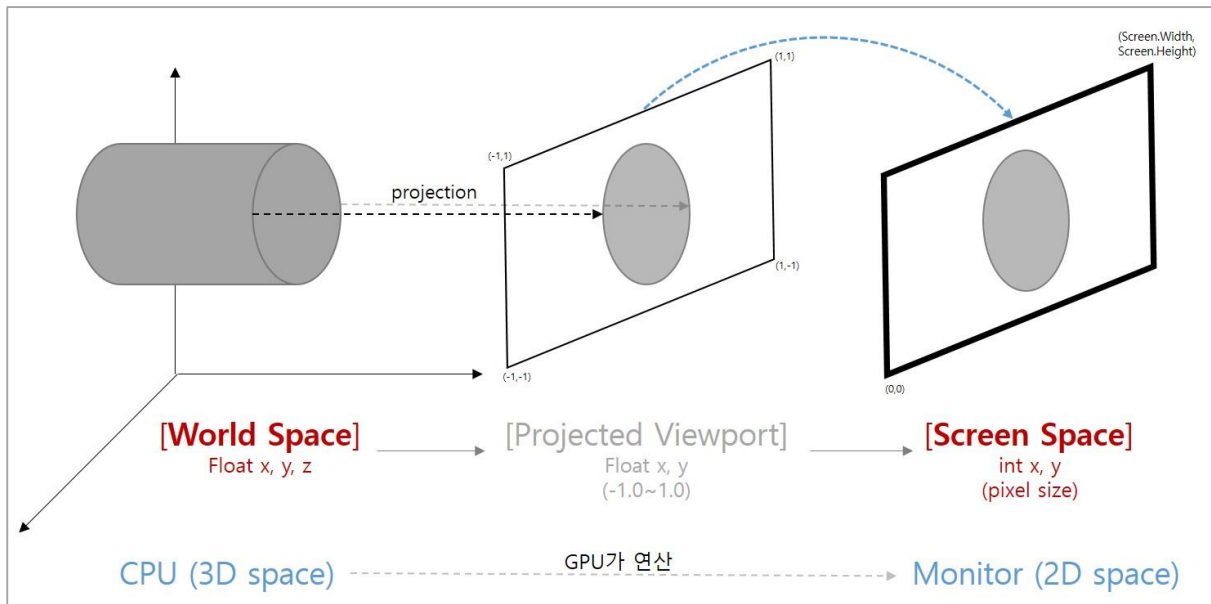
- 좌표계 : WORLD, SCREEN

유니티에서 쓰이는 좌표계는 크게 두 가지가 있는데, 각각 3차원 공간을 나타내는 World, 그리고 카메라에 의해 렌더링 된 평면인 Screen이다.

World Space는 컴퓨터 연산 상에서 3D 물체가 존재하는 공간이다. CPU가 연산하며, Vector3 (float x, float y, float z) 값을 Position으로 가진다.

그러나 우리가 게임을 할 때 눈에 보이지 않는 물체까지 컴퓨터가 연산할 필요는 없으므로 실제로 모니터에 출력되는 부분만을 따로 처리하는데, 이렇게 모니터에 맞게 변환된 World Space의 일부를 Screen Space라고 한다. 실제로 플레이어가 게임 도중 보는 화면이다.

Screen Space는 Camera Component가 비추고 있는 공간(Viewport)의 3차원 정보를 2차원 정보로 바꾼 것으로, GPU가 연산하며 int x, int y 값을 Position으로 가진다.



Unity에서 쓰이는 좌표계는 World와 Screen 두 가지가 있다. 3차원 정보를 2차원 정보로 변환하는 것을 Projection이라고 한다.
자세한 내용은 3학년 1학기 Computer Graphics 전공수업에서 배웁시다.

Screen Space와 World Space 변환 과정에는 'ViewPort'가 존재하며 이는 Camera Component가 제공한다. 그래서, 유니티에서는 Camera Component를 통해서 World Space와 Screen Space 값을 서로 변환할 수 있게 하는 메서드를 제공한다. API 참고. <http://docs.unity3d.com/ScriptReference/Camera.html>

```
Public Vector3 ScreenToWorldPoint ( Vector3 ScreenPos );  
Public Vector3 WorldToScreenPoint( Vector3 WorldPos );
```


- PIXEL RESOLUTION

Unity World Space는 기본적으로 3D 공간이지만 Sprite와 같은 2D GameObject도 제공한다. Width, height pixel값을 가진 Sprite를 World Space에 존재하게 하려면 해당 pixel 값을 적절한 Vector3 값으로 바꾸어야 하는데, 이 연산에 쓰이는 값이 바로 Pixel Resolution이다.

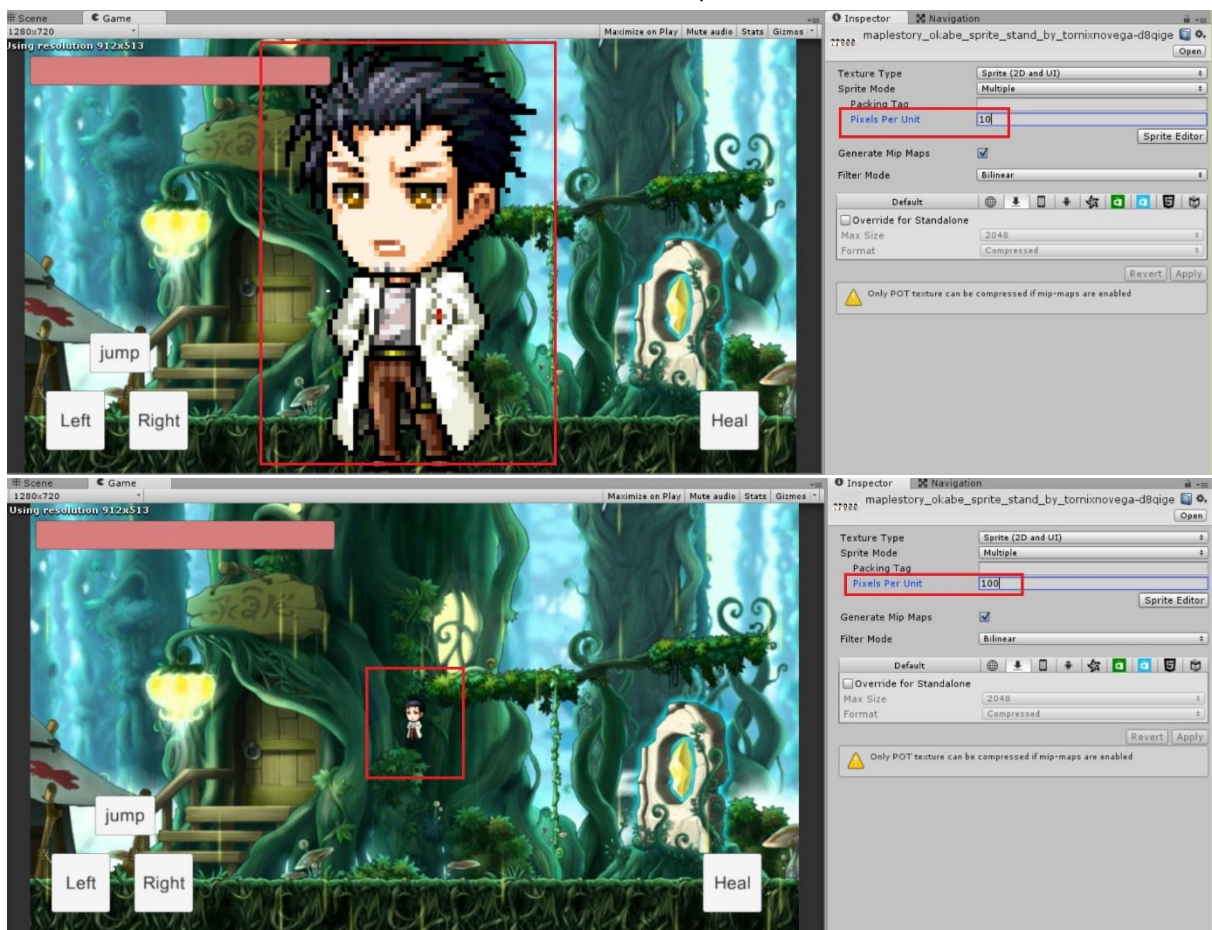
Sprite가 World Space에 들어갔을 때의 크기는 다음과 같이 계산된다.

$$\text{Sprite. X} = \text{Width(pixel)} / \text{Pixel Resolution}$$

$$\text{Sprite. Y} = \text{Height(pixel)} / \text{Pixel Resolution}$$

예를 들어, 크기가 1920x1080이고 Pixel Resolution이 100인 어떤 Sprite를 World에 띄우고 좌측 아래가 (0,0,0)이 되게 이동한다면, 우측 상단의 좌표는 (19.20, 10.80, 0) 이 된다.

Pixel Resolution 값이 작을수록 World 내에서 보여지는 Sprite의 크기 역시 커진다.



Pixel Resolution이 10일 때(위)와 Pixel Resolution이 100일 때(아래) Sprite의 크기 차이.

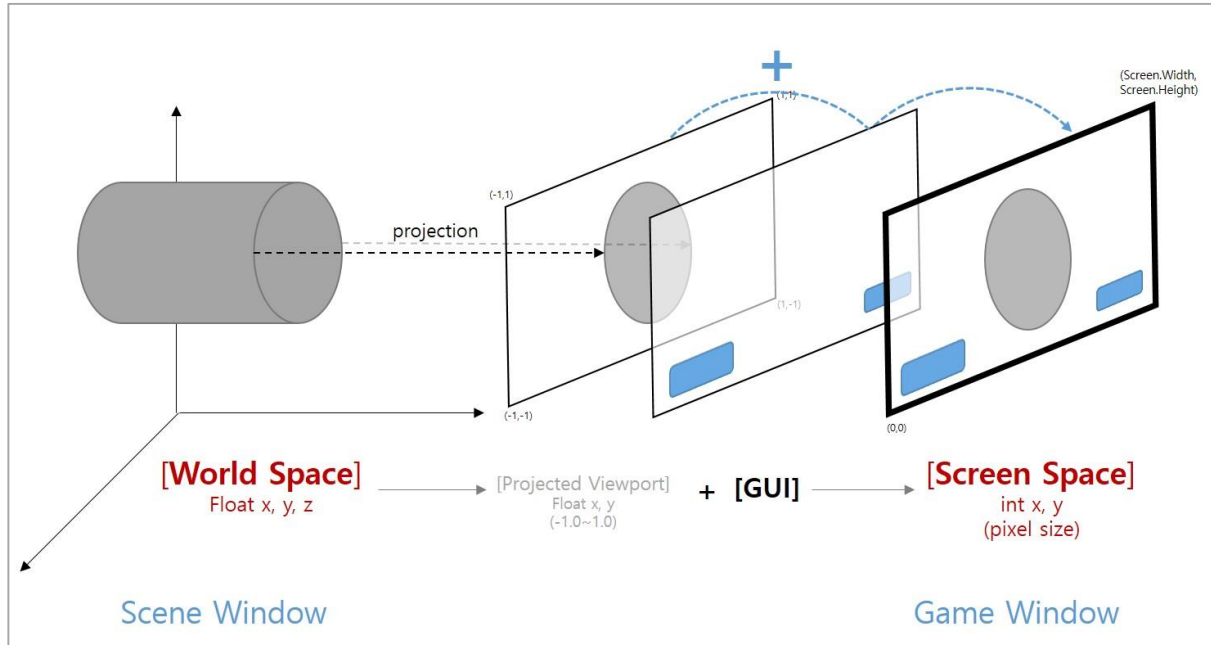
Pixel Resolution은 Image Import Setting에서 바꿀 수 있다. Asset에서 image 파일을 선택했을 경우 Inspector에 Import Setting이 뜬다.

Pixel Resolution은 World 내에서 Sprite를 규격에 맞게 배치할 때 유용하게 쓰이는데, Camera의 ScreenToWorld 메서드를 사용하여 화면에 보이는 Sprite의 World 값을 계산했을 경우 오차가 생길 수 있기 때문이다. (실제로 은근히 생긴다)

- CANVAS : WORLD SPACE에 존재하는 SCREEN SPACE

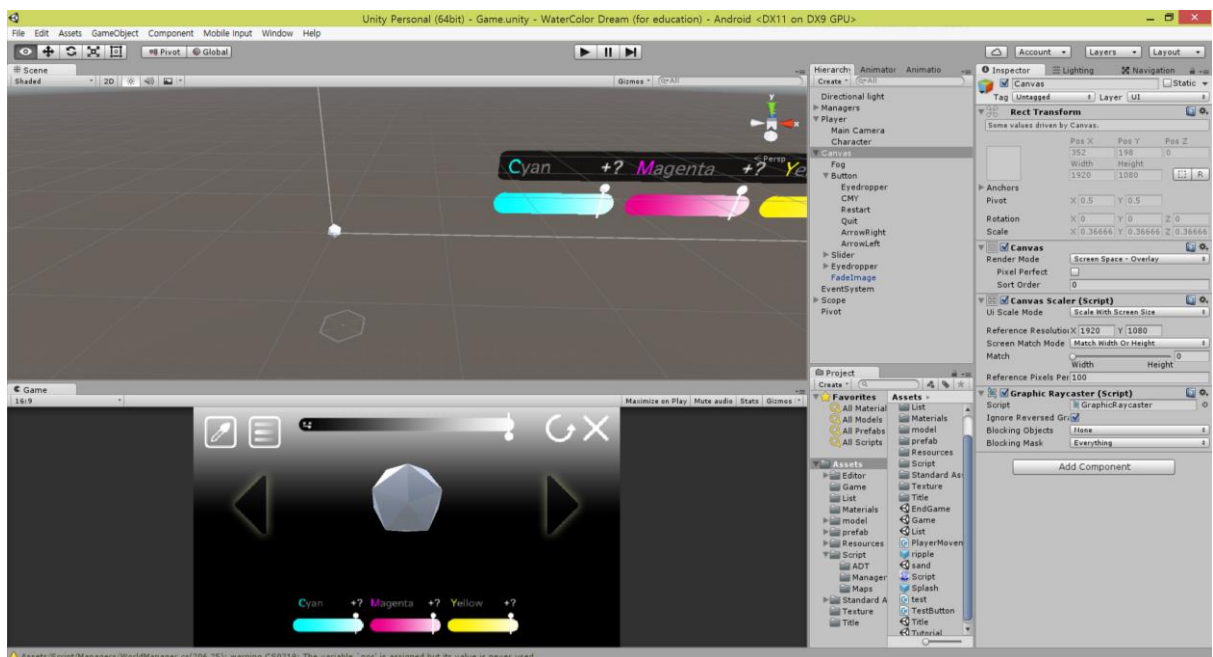
Unity는 Screen Space에 띄울 수 있는 버튼, 텍스트, 슬라이더 등의 GUI를 제공한다.

GUI는 World 좌표가 아닌 Screen 좌표계를 사용하고, 가장 마지막에 렌더링되어서 화면으로 출력되기 때문에 Unity 4버전까지는 Scene Window이 아닌 Game Window에서만 확인할 수 있었다.



Unity 4 버전까지는 GUI를 Scene Window에서 확인할 수 없었다.

Unity 5 버전부터는 World Space상에서 “Canvas”라는 특수한 GameObject를 통해 GUI를 제공한다. 즉, Scene window 내에서 GUI의 위치를 확인하고 조절할 수 있게 되었다.

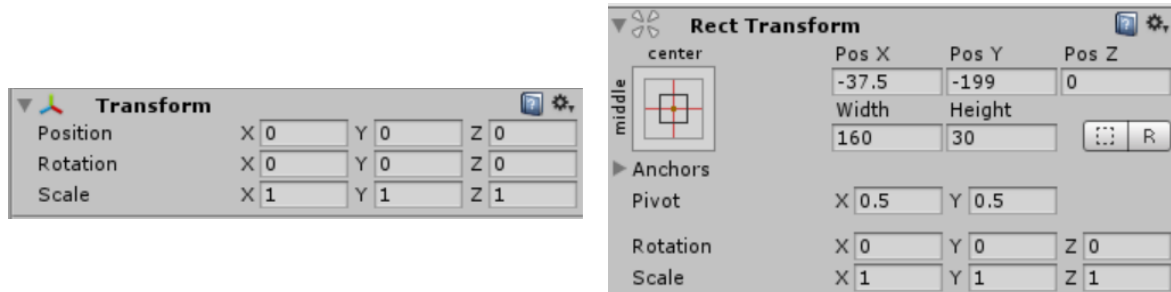


Unity 5버전부터 Scene Window (상단) 에서도 GUI 배치를 볼 수 있다.

“Canvas” 라는 GameObject의 Child에 있는 UI GameObject가 모두 렌더링되어 보여진다.

- RECT TRANSFORM

Canvas는 일종의, World Space에 존재하는 Screen Space이다. 때문에 World Space의 좌표계인 Vector3(float x, float y, float z) 뿐만 아니라 width, height 역시 사용하는데, 이를 관리하는 Component가 Rect Transform이다.



Transform Component (좌), Rect Transform Component (우)

주목할 점은 Rect Transform Component는 Canvas GameObject가 가진 "Canvas Scaler" Component 의 영향을 받는다는 것이다. Canvas Scaler는 Rect Transform을 위한 직사각형 범위를 제한하는 역할을 한다. 그래서 Screen 좌표계를 따르는 UI들 (Button, UI Text, ...)을 렌더링하고 싶으면 Canvas의 Child로 존재해야 하고, Rect Transform을 쓰는 UI GameObject는 "Canvas Renderer"라는 Component를 가지고 있다.

Rect Transform에 대해서 좀 더 살펴보자. Rect Transform은 Transform 클래스를 상속받은 클래스이다. 즉, Transform이 가지고 있던 위치, 회전, 크기, 그리고 Hierarchy에 대한 요소를 모두 가지고 있으며 거기에 "Anchor"와 "Pivot"이 새로 추가되었다.

ProceduralTexture

Profiler

Projector

QualitySettings

Quaternion

Random

Ray

Ray2D

RaycastHit

RaycastHit2D

Rect

RectTransform

RectTransformUtility

ReflectionProbe

RelativeJoint2D

RenderBuffer

Renderer

RenderSettings

RectTransform

class in UnityEngine / Inherits from: [Transform](#)

[SWITCH TO MANUAL](#)

Description

Position, size, anchor and pivot information for a rectangle.

RectTransforms are used for GUI but can also be used for other things. It's used to store and manipulate the position, size, and anchoring of a rectangle and supports various forms of scaling based on a parent RectTransform.

Variables

anchoredPosition	The position of the pivot of this RectTransform relative to the anchor reference point.
anchoredPosition3D	The 3D position of the pivot of this RectTransform relative to the anchor reference point.
anchorMax	The normalized position in the parent RectTransform that the upper right corner is anchored to.
anchorMin	The normalized position in the parent RectTransform that the lower left corner is anchored to.

Rect Transform API. Transform을 상속받았다. <http://docs.unity3d.com/ScriptReference/RectTransform.html>

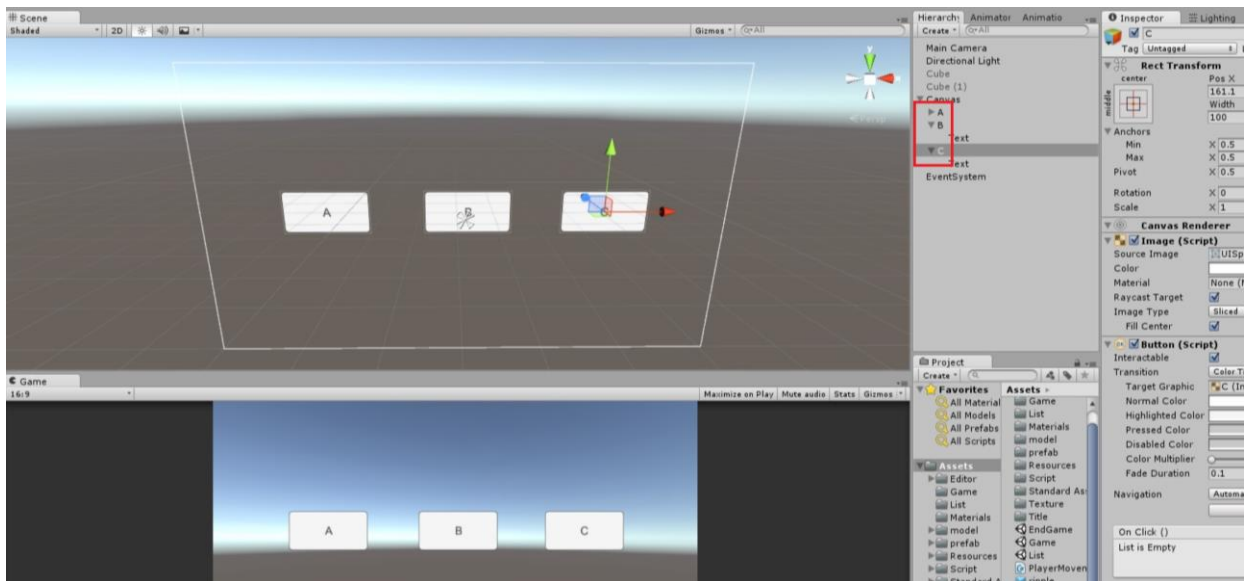
UI GameObject의 종류와 Anchor와 Pivot을 다루는 법은 디자인스터디 6회차 ppt에 자세히 나와 있다. Anchor와 Pivot에 대한 설명은 33p부터 ~ <https://kucatdog.net/viewtopic.php?id=717>

- UI EXAMPLE : SELECT STAGE BUTTON

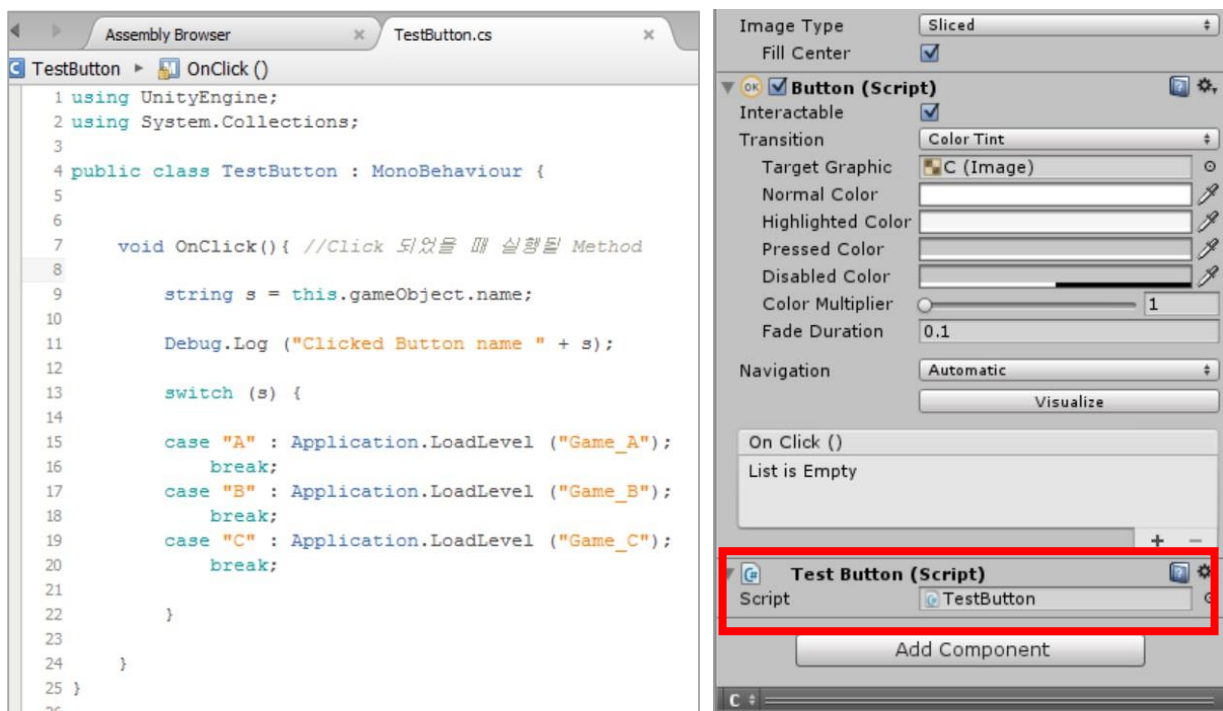
Scene Stage에서 버튼 A, B, C를 누르면 Scene Game_A, Game_B, Game_C로 이동하게 해보자.
고려해야 할 점은 다음과 같다.

- 1) 버튼의 배치 – Rect Transform?
- 2) 어떻게 해야 버튼이 눌러진 게 감지되었을 때 특정 함수가 실행되게 하는가?

링크된 ppt를 참고하여, Button GameObject 3개를 만들고 적절히 배치한 뒤, 각 버튼 GameObject 이름과 Child로 붙어있는 Text Component를 A, B, C라고 바꿔보자.



그리고 다음과 같은 스크립트를 짜서 각 Button GameObject 에 Component로 붙여보자.



이 상태에서 Editor Play 버튼을 눌러 Game Window에서 버튼을 클릭해보면, 당연히, 아무 일도 일어나지 않는다.

왜냐하면 OnClick()은 MonoBehaviour의 Message가 아니기 때문이다.

ParticleSystem	
ParticleSystemRenderer	
Microphone	
MonoBehaviour	
Motion	
MovieTexture	
NavMesh	
NavMeshAgent	
NavMeshHit	
NavMeshObstacle	
NavMeshPath	
NavMeshTriangulation	
Network	
NetworkMessageInfo	
NetworkPlayer	
NetworkView	
NetworkViewID	

OnBecameVisible	OnBecameVisible is called when the renderer became visible by any camera.
OnCollisionEnter	OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.
OnCollisionEnter2D	Sent when an incoming collider makes contact with this object's collider (2D physics only).
OnCollisionExit	OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.
OnCollisionExit2D	Sent when a collider on another object stops touching this object's collider (2D physics only).
OnCollisionStay	OnCollisionStay is called once per frame for every collider/rigidbody that is touching rigidbody/collider.
OnCollisionStay2D	Sent each frame where a collider on another object is touching this object's collider (2D physics only).
OnConnectedToServer	Called on the client when you have successfully connected to a server.
OnControllerColliderHit	OnControllerColliderHit is called when the controller hits a collider while performing a Move.
OnDestroy	This function is called when the MonoBehaviour will be destroyed.

MonoBehaviour API 문서엔 Message 목록이 ABC 순서대로 정리되어있다. OnClick()은 Message 목록에 없다.

즉, 다른 방식으로 불러야 한다. 어떻게?

- EVENT HANDLING

예제의 2번 고려사항을 해결하기 위해 필요한 개념인 Event에 대해서 알아보자.

Event는 사실 우리가 알게 모르게 많이 접해왔다. Html과 javascript로 이루어진 페이지 안에서 어떤 버튼을 클릭했을 때, 다음 페이지로 넘어가거나 경고 창이 뜨는 등의 처리가 일어난 것이 바로 Event이다.

Unity API Reference에서는 Event를 다음과 같이 설명한다.

DynamicGI	
EdgeCollider2D	
Effector2D	
EllipsoidParticleEmitter	
Event	
FixedJoint	
FixedJoint2D	
Flare	

<h2>Event</h2>
class in UnityEngine
<h3>Description</h3>
A UnityGUI event.
Events correspond to user input (key presses, mouse actions), or are UnityGUI layout or rendering events.
http://docs.unity3d.com/ScriptReference/Event.html

Event는, 키 입력이나 마우스 액션과 같은 유저의 인풋에 상응하는 것이다.

마우스 커서의 위치나 마우스 클릭, 키보드 입력, 터치 입력 등 주로 Interface에서 발생하는 입력에 대해 상응하는 어떠한 처리를 한다. 웹페이지의 경우 그 '처리'가 팝업이나 경고를 띄우거나 새 html 페이지를 여는 것이고, Unity의 경우에는 특정 input에 대해 미리 등록된 메서드를 실행하는 것이다.

Event가 일어나기 위해선 2가지가 필요하다..

- 1) Input을 감지해야 한다.
- 2) Input이 감지될 때 실행시킬 method를 미리 등록해야 한다.

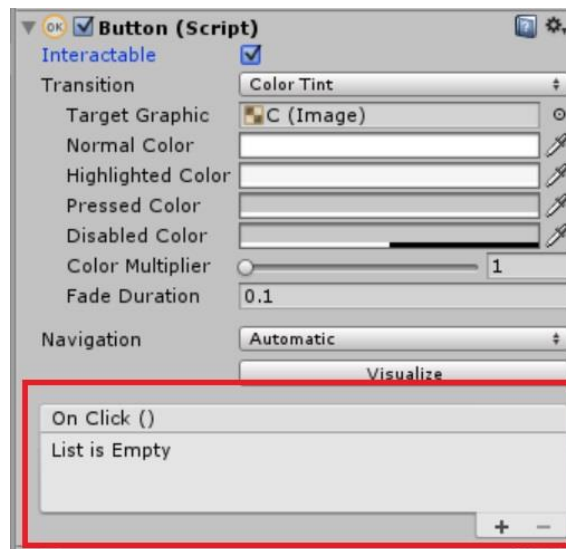
1번 조건을 해결하기 위해선 4회차 예제에 나왔던 Event System 이라는 Component가 필요하다.

(Event System이라는 Component는 정확히 무슨 역할을 할까?에 대해서는 다음 회차인 부록으로 >>)

2번 조건을 해결해보자.

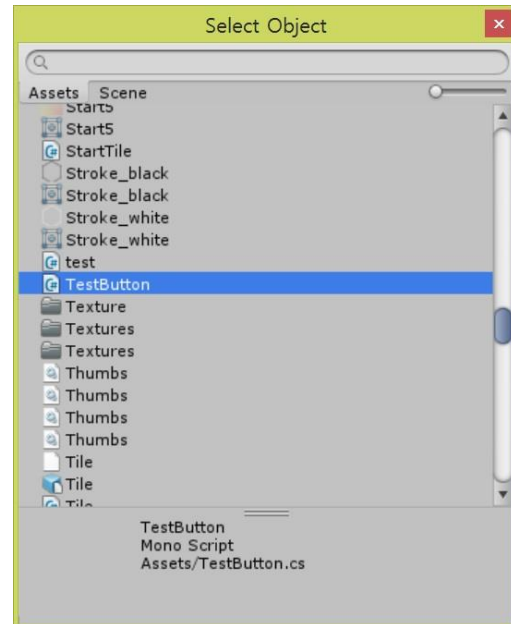
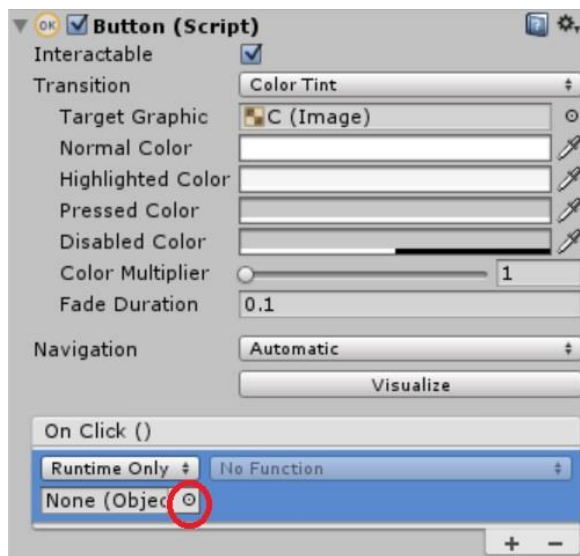
약속된 이름만을 사용하면 언제든 method가 호출되는 Message와는 달리, Event를 통해 특정 method 를 호출하기 위해선 Event에 그 method를 미리 '등록' 해야한다.

Unity의 Button Class는 Inspector에서 OnClick()이라는 Event에 method를 등록할 수 있도록 인터페이스를 제공한다.



OnClick() List에 메서드를 등록하면, 해당 버튼이 클릭되는 input이 들어올 때 List 안의 메서드가 모두 실행된다.

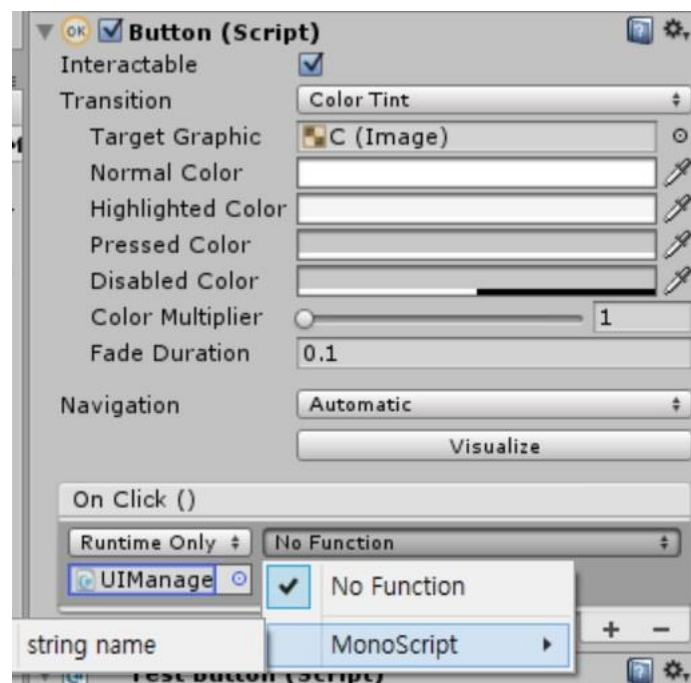
주목해야 할 점은, Event에 의해 실행되는 method는 모두 "Component"의 함수로 존재할 때에만 OnClick() 에서 실행할 수 있다는 것이다. 다음 장을 보자.



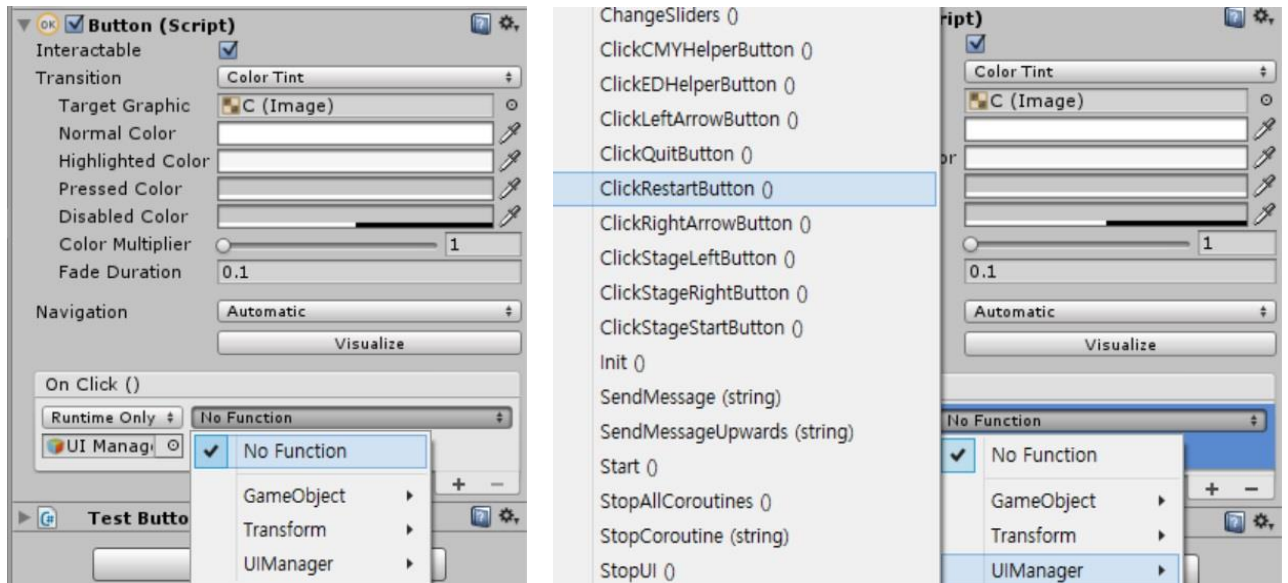
OnClick() List 아래 + 버튼을 누르면 메서드를 등록할 수 있다. Object를 등록한 후 해당 Object에 포함되어있는 Functions을 등록하면 메서드가 등록된다.

왼쪽 스샷의 동그라미 버튼을 누르면 Object를 등록할 수 있다. 버튼을 누르면 오른쪽 스샷처럼 Object를 선택하는 창이 뜨는데, Scene 내부 Hierarchy상에 존재하는 GameObject 뿐만 아니라 Assets 폴더 하위에 있는 기타 Object들 (Script, prefab, 리소스, ...) 도 등록할 수 있다.

UIManager.cs의 예시를 먼저 보겠다. Asset에 있는 UIManager.cs Object를 등록시킨 후 Method를 등록하려고 Function을 눌러보니, 어떤 함수나 변수도 인식하지 못하는 것을 볼 수 있다.



반면, 빈 GameObject에 UIManager Component를 붙인 'UI Manager' GameObject를 Scene에서 찾아 등록시키면 아래와 같이 UIManager 라는 Component 내의 public method들을 인식한다.



해당 Component 내의 public method와 variable만 인식한다.

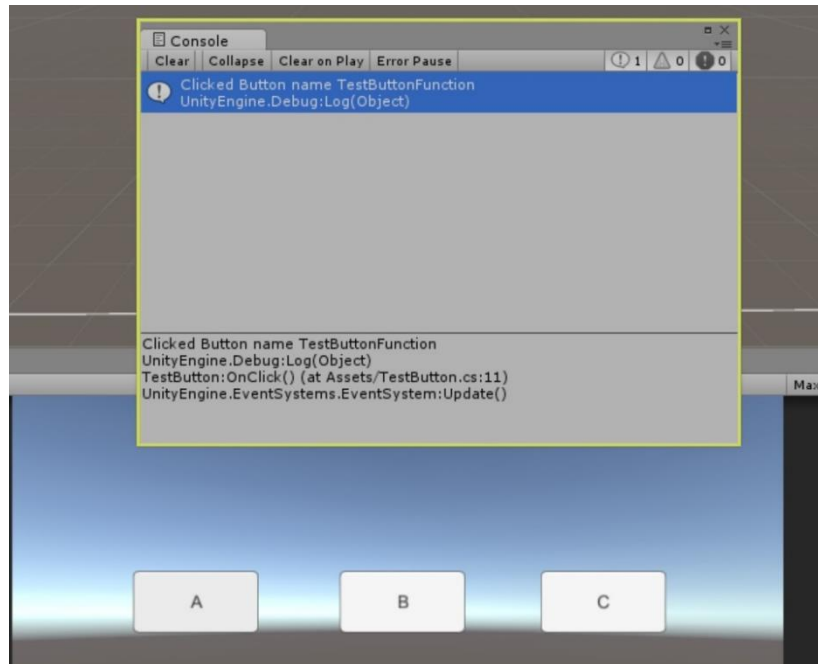
따라서, 위에서 짰던 TestButton.cs 스크립트를 등록시키기 위해 OnClick(); 함수의 접근제한자를 public으로 바꾸고, 어떠한 GameObject에 TestButton Component를 추가한 후 그 GameObject를 Button에 등록해야 한다.

```

1 using UnityEngine;
2 using System.Collections;
3
4 public class TestButton : MonoBehaviour {
5
6
7     public void OnClick() { //Click 되었을 때 실행될 Method
8
9         string s = this.gameObject.name;
10
11         Debug.Log ("Clicked Button name " + s);
12
13         switch (s) {
14
15             case "A" : Application.LoadLevel ("Game_A");
16                         break;
17             case "B" : Application.LoadLevel ("Game_B");
18                         break;
19             case "C" : Application.LoadLevel ("Game_C");
20                         break;
21
22         }
23     }
24 }
25
26

```

위와 같이 OnClick 함수의 접근제한자를 바꾸고, 이 Script가 Component로 붙을 새 GameObject를 만들자. 이 GameObject의 이름을 "TestButtonFunction" 으로 한 후, Component를 붙이고 버튼 A,B,C에 각각 등록시켜보자. 그리고, Play를 해보자.



버튼을 누르면 Log는 찍히는데 Scene이 그대로이다. 왜일까?

Script를 다시 보자. `OnClick()` method를 짤 때, 해당 Button에 Component가 붙을 걸 예상하고 코드를 짤 때 때문이다. Button에 Component가 붙을 걸 예상해서, 이 Component가 붙어있는 `gameObject`의 이름(string)을 받아와 switch문을 체크했는데, 정작 `TestButton` Component가 붙은 `GameObject`의 이름은 "TestButtonFunction" 이므로 switch문을 통과하지 못했다.

Script를 고치는 방법은 여러가지가 있겠지만, 그 중 하나의 방법을 소개하고자 한다. 아래와 같이 Script를 고쳐보자.

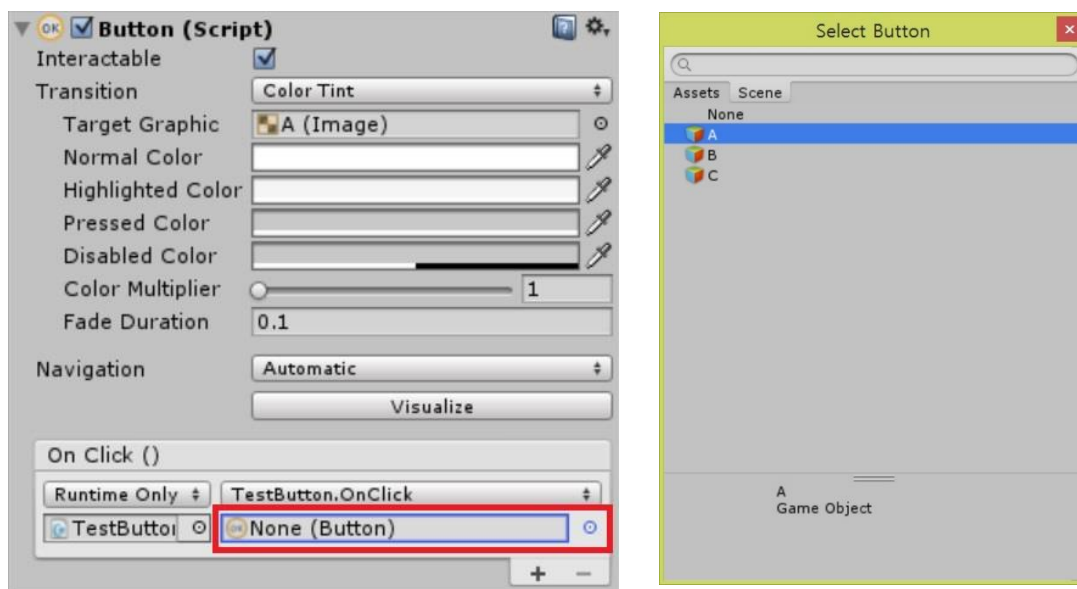
```
1 using UnityEngine;
2 using System.Collections;
3 using UnityEngine.UI;
4
5 public class TestButton : MonoBehaviour {
6
7
8     public void OnClick(Button b){ //Click 되었을 때 실행될 Method
9
10         string s = b.gameObject.name;
11
12         Debug.Log ("Clicked Button name " + s);
13
14         switch (s) {
15
16             case "A" : Application.LoadLevel ("Game_A");
17                         break;
18             case "B" : Application.LoadLevel ("Game_B");
19                         break;
20             case "C" : Application.LoadLevel ("Game_C");
21                         break;
22
23         }
24
25     }
26 }
```


Button Type의 변수를 파라미터로 받아서, gameObject의 이름 대신 해당 Button Component가 붙어있는 GameObject의 name으로 switch문을 실행하는 Script이다. Button Class를 Script에서 사용하기 위해서, UnityEngine.UI 를 import했다.

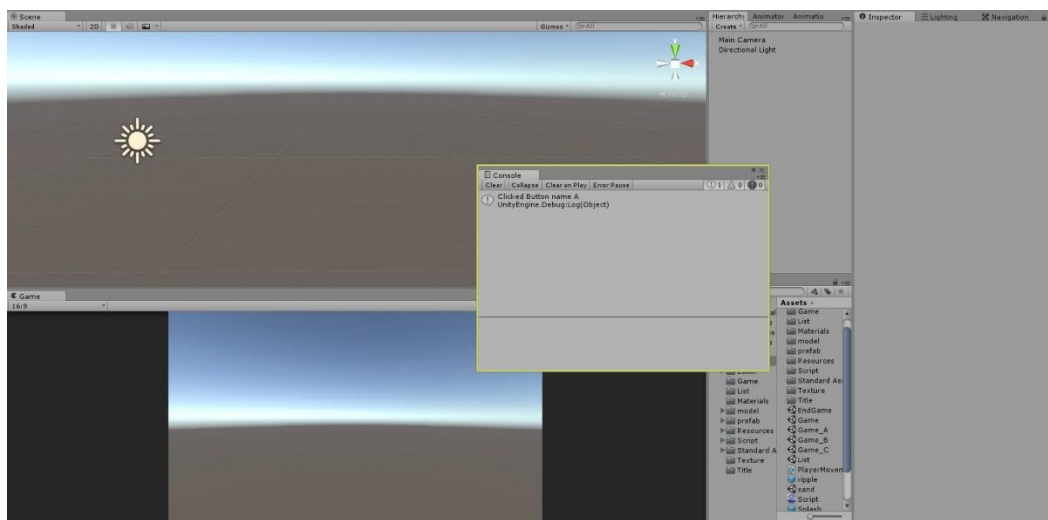
고친 Script를 빌드한 후 (F7), Button Inspector에서 기존에 등록했던 method를 지우고 다시 GameObject와 method를 등록하면 못 보던 칸이 새로 생김을 확인할 수 있다.

바로, parameter를 등록하는 칸이다. TestButton.OnClick method는 Button type을 Parameter로 받기 때문에 등록할 수 있는 object도 Button type뿐이다.

우측의 동그란 버튼을 눌러 자기 자신을 등록시키자. 엄밀히는, Scene 내부에서 "A"라는 이름의 GameObject에 Component로 달려있는 Button을 등록하는 것이다.



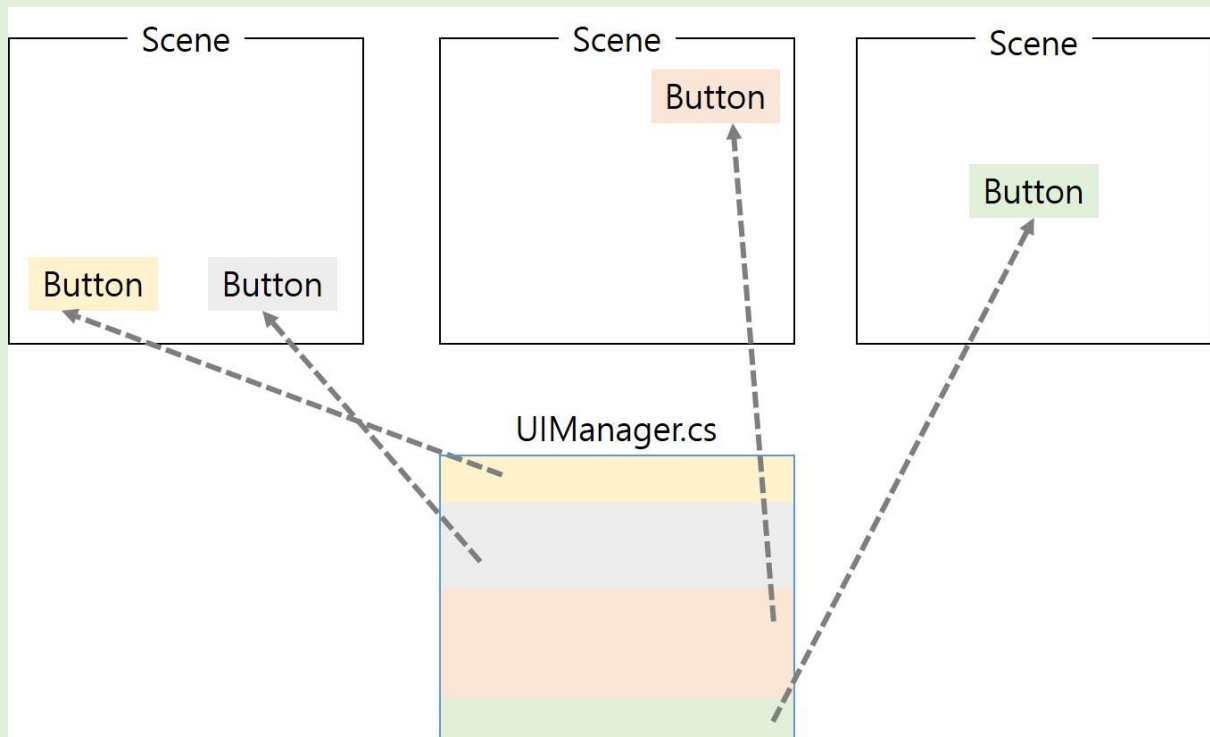
Play를 해 보면 클릭한 Button의 이름이 Log로 찍히고 Scene도 정상적으로 변하는 것을 볼 수 있다.



- UI MANAGER CLASS

버튼에 `OnClick()` Event를 등록하려면 반드시 Script가 Component의 형태로 GameObject에 붙어 있어야 한다. 이런 '실체가 없는 Method뿐인 Class'가 많아질수록 불필요한 Script의 개수가 많아질 것이다.

이를 피하기 위해, 버튼에 등록시키기 위한 method들을 모두 하나의 Class에 모아 관리하자! 라는 아이디어에서 나온 Class가 바로 UI Manager Class이다.



UI Manager는 Button에 등록할 public method들을 모아놓은 Class이다.
이런 특성 때문에, 계층구조를 가진 설계를 할 때 게임상의 UI GameObject를 관리하기도 한다.

5 회차 과제) 버튼을 꼭 누르는 동안 GameObject가 각각 `Vector3.left`와 `Vector3.right` 방향으로 움직이는 버튼 UI를 만들어보자!

버튼으로 움직이는 GameObject는 tag가 "Player"인 것으로 제한한다. 만약 Scene 상에 "Player" 태그를 가진 GameObject가 없을 경우 "Player 없음" 로그를 찍는다.

힌트 : `FindGameObjectWithTag()` 메서드, Event Trigger Component와 Pointer Down, Pointer Up Event