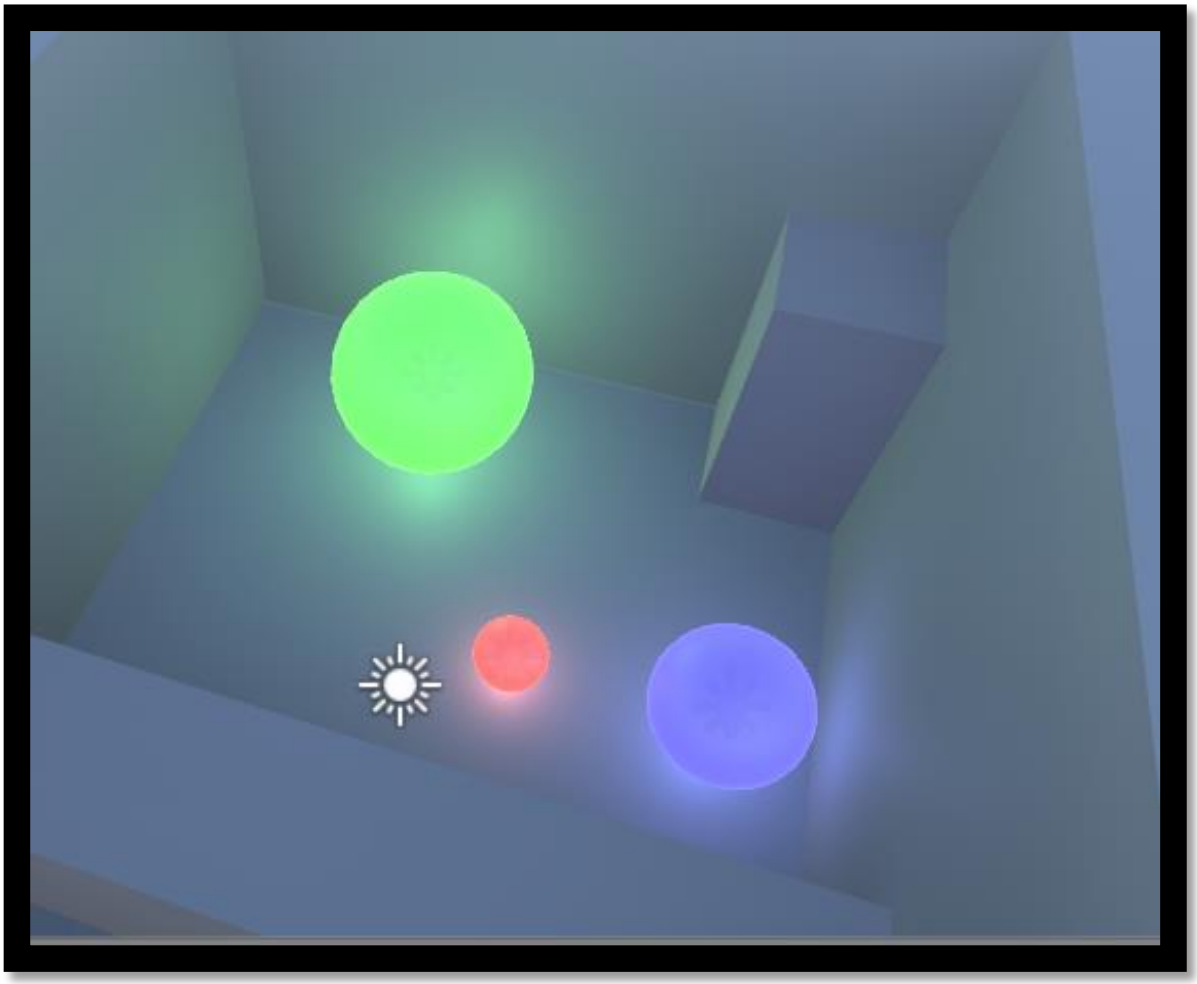
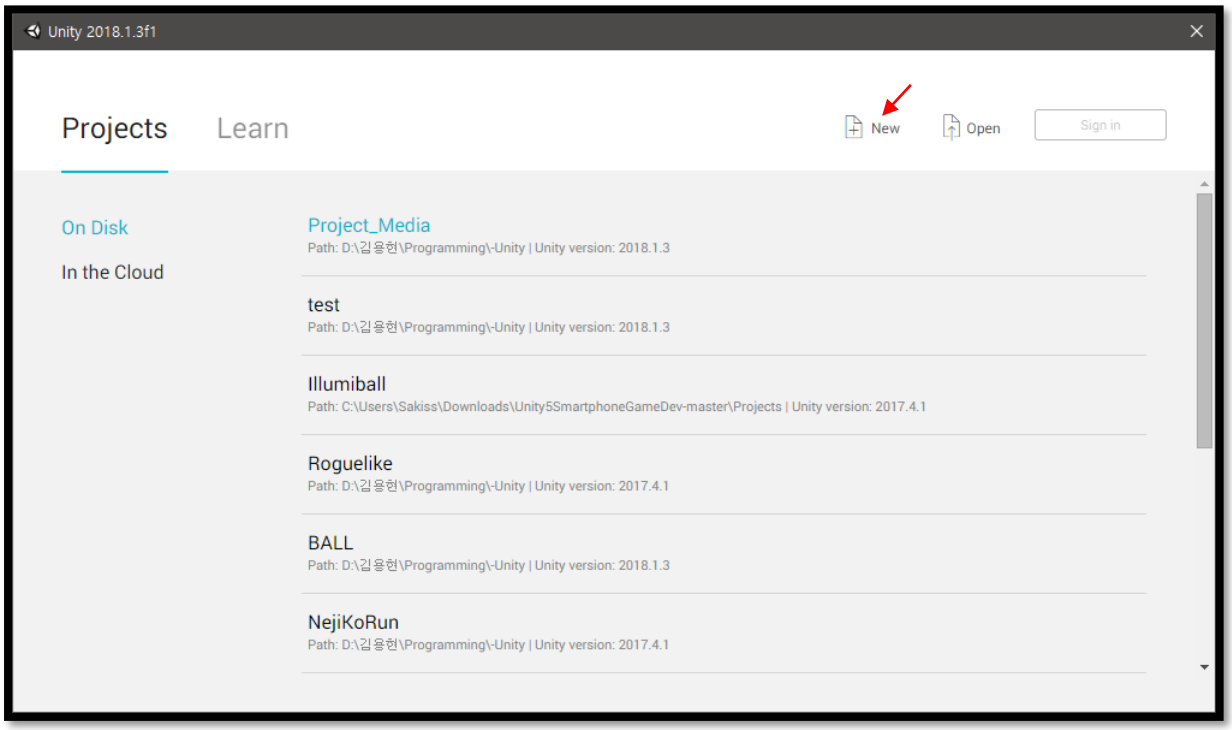


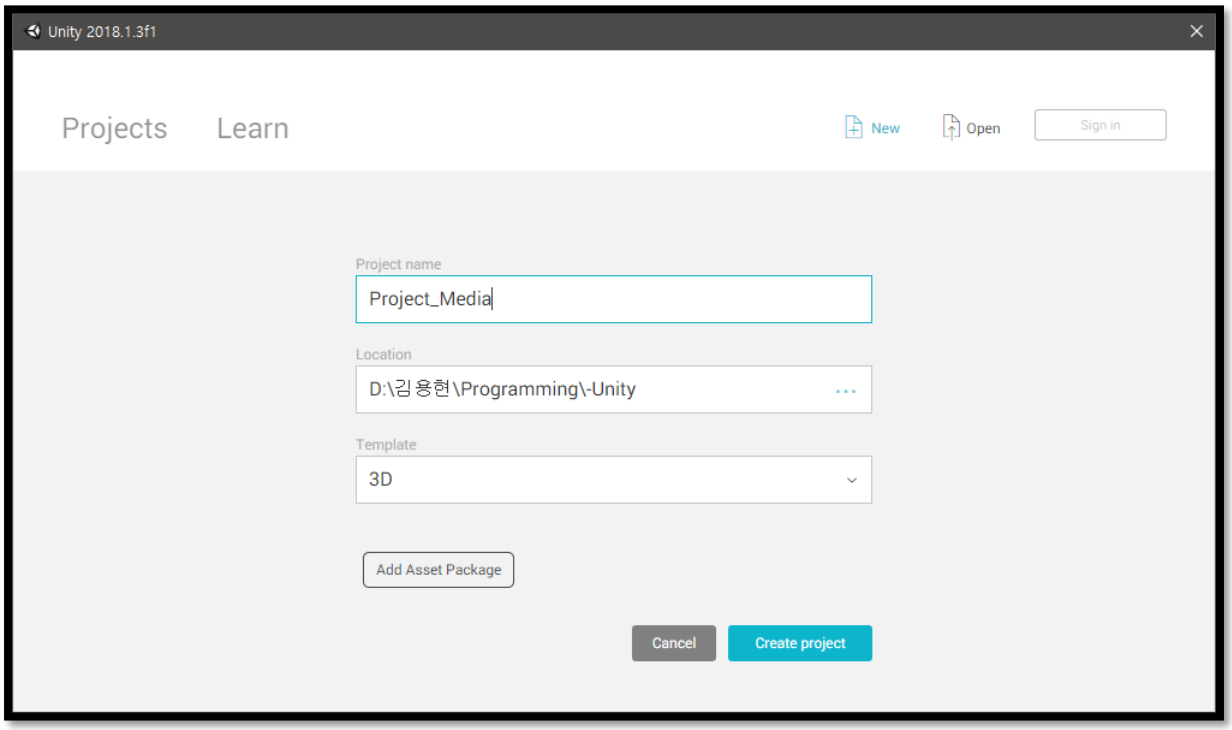
Unity와 함께하는 간단한 게임 만들기

김용현 | Unity 실습 | 18.06.11

Index

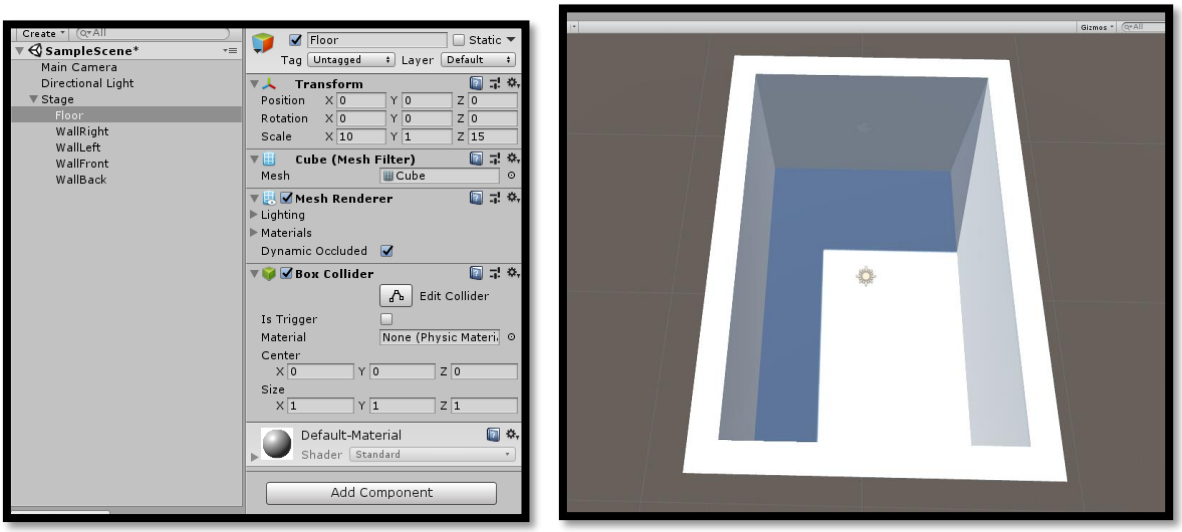
1. Project 생성





- 프로젝트 이름과 저장 위치, 3D로 설정하고 새로운 프로젝트를 만든다.
- 이번 프로젝트에선 에셋이 필요하지 않으므로 에셋을 추가하지 않고 만든다.

2. Stage 생성



- 새로운 빈 오브젝트를 하나 만든 후 Stage로 이름을 바꾼다.
  (GameObject - Create Empty)
- Cube 오브젝트 여러 개를 통해 Stage를 만들고 Stage에 상속 시킨다.
  (GameObject - 3D Object - Cube)

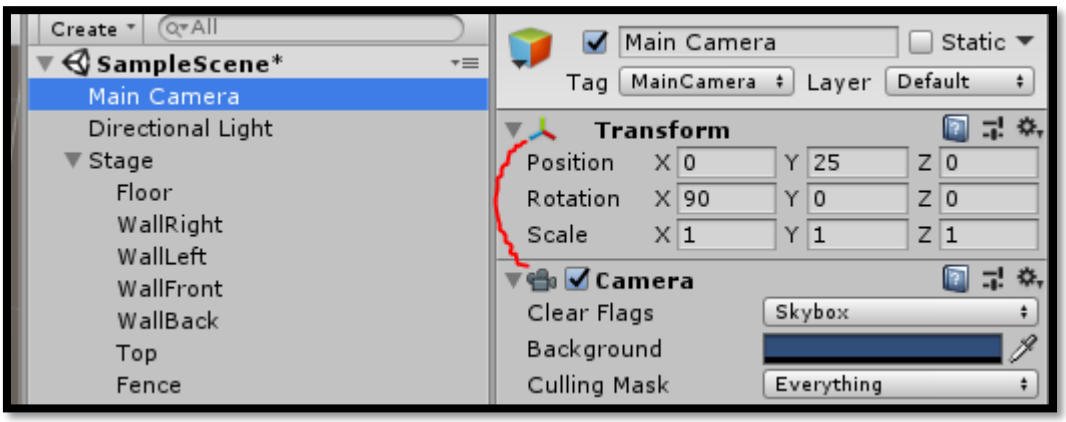
<Position>

Floor	0	0	0
WallRight	5.5	3.5	0
WallLeft	-5.5	3.5	0
WallFront	0	3.5	-8
WallBack	0	3.5	8

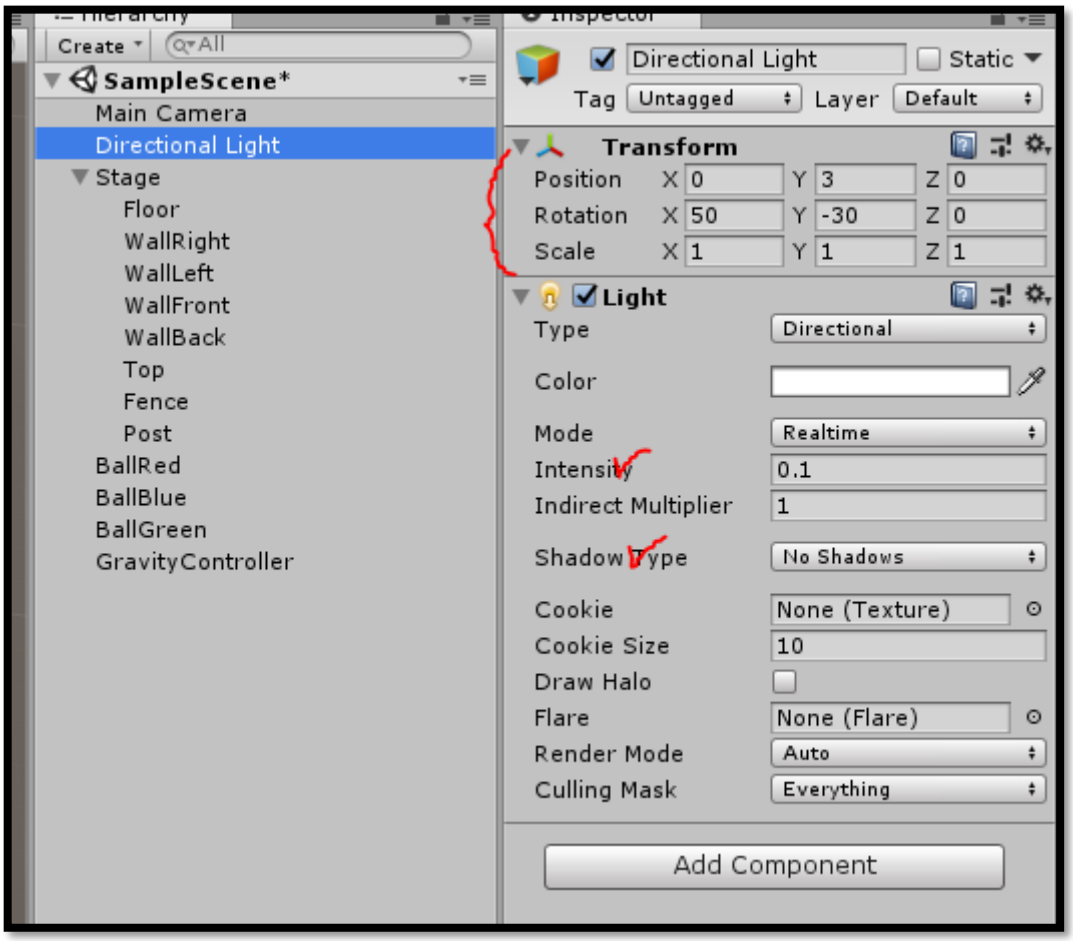
<Scale>

Floor	10	1	15
WallRight	1	8	17
WallLeft	1	8	17
WallFront	10	8	1
WallBack	10	8	1

3. Game View 및 Directional Light 설정



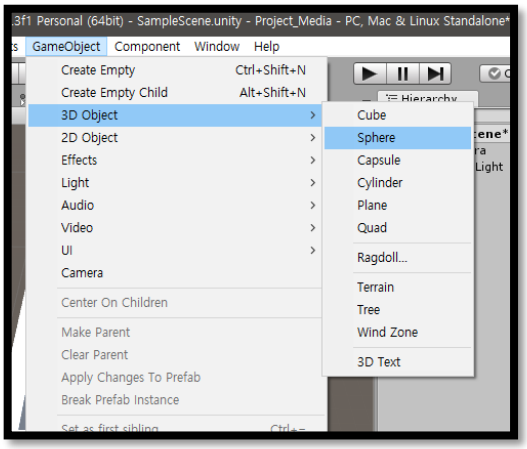
- Stage를 내려다 볼 수 있게 위와 같이 Position 및 Rotation을 수정

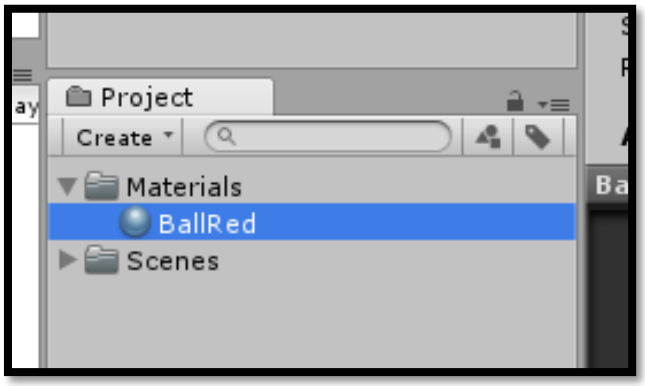


- 이 게임에서는 직접 조명이 거의 필요하지 않으므로 Intensity를 0.1로 줄임.
  그림자 또한 필요 없으므로 Shadow Type을 No Shadows로 바꾼다.
  (하지만 직접 조명 수치는 후에 수정하는 걸 추천)

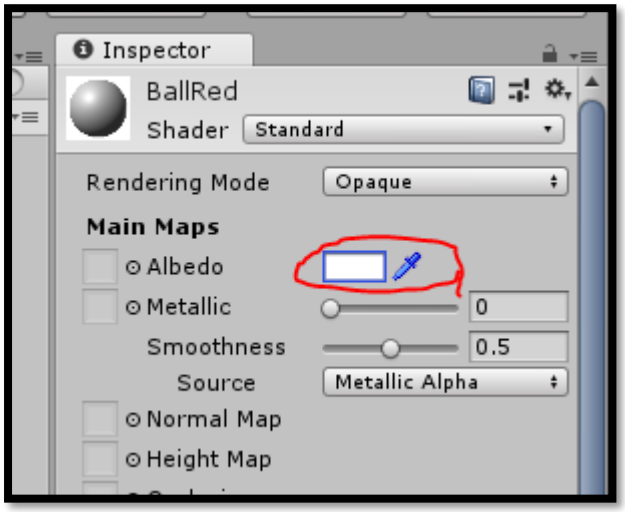
4. Ball 생성

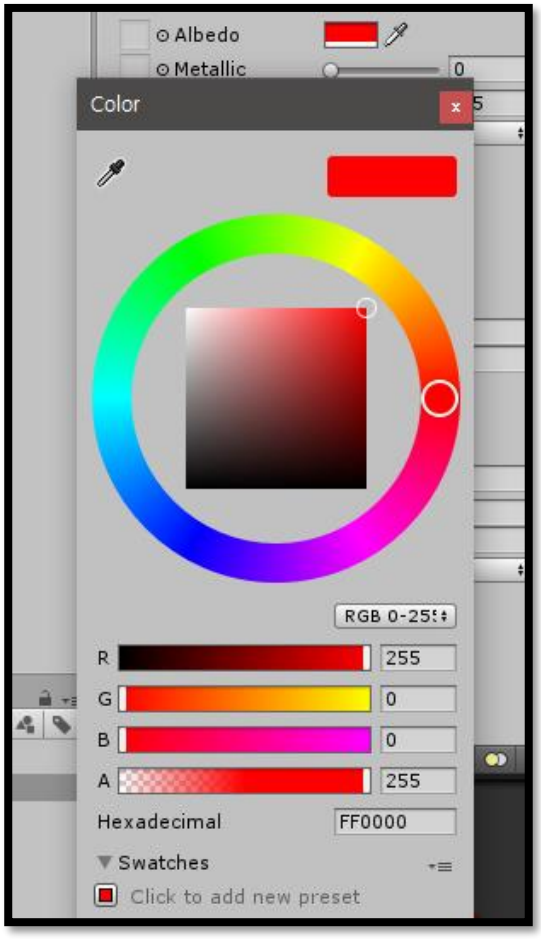
1. Material 설정



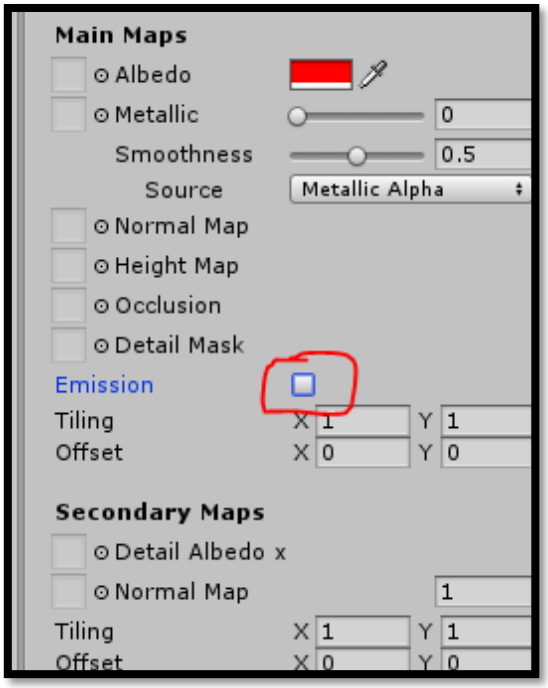


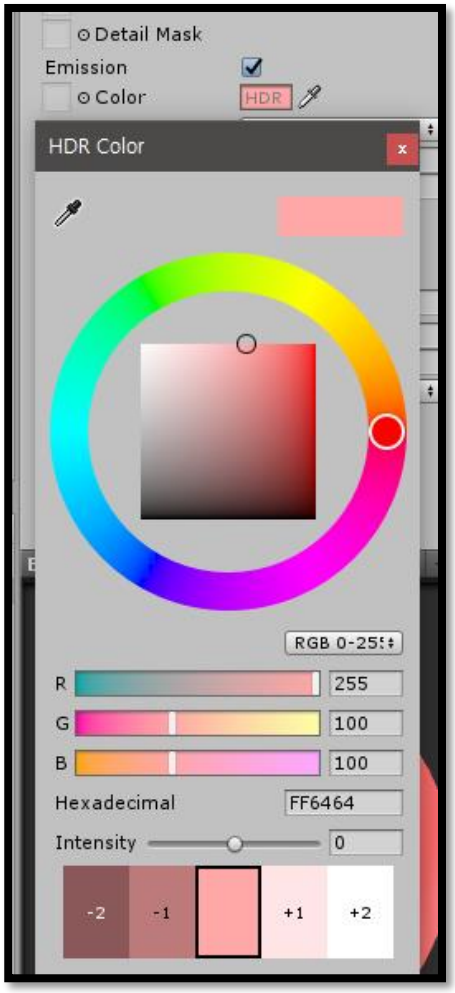
- Sphere 오브젝트 생성 후 BallRed로 이름을 바꾼다.
- Project 창에서 Material를 만들고 BallRed로 이름을 바꾼다.
  (Create - Material)

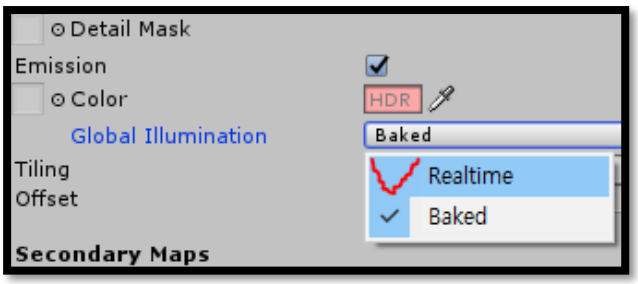




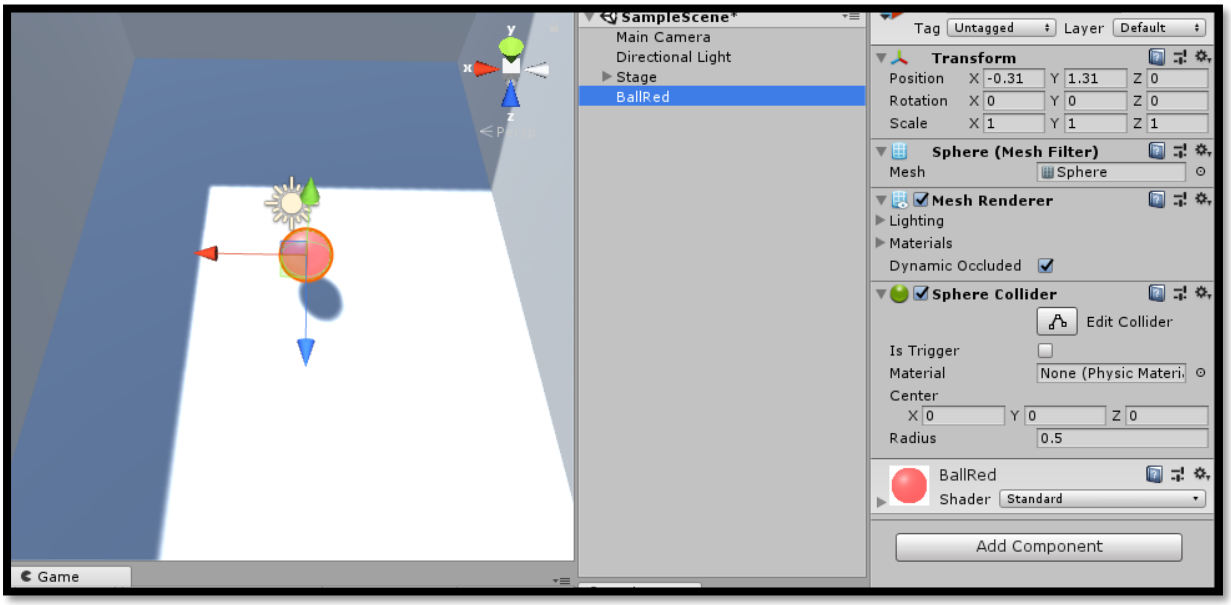
- 이후 Inspector 창에서
  Albedo와 Emission을 설정해준다.



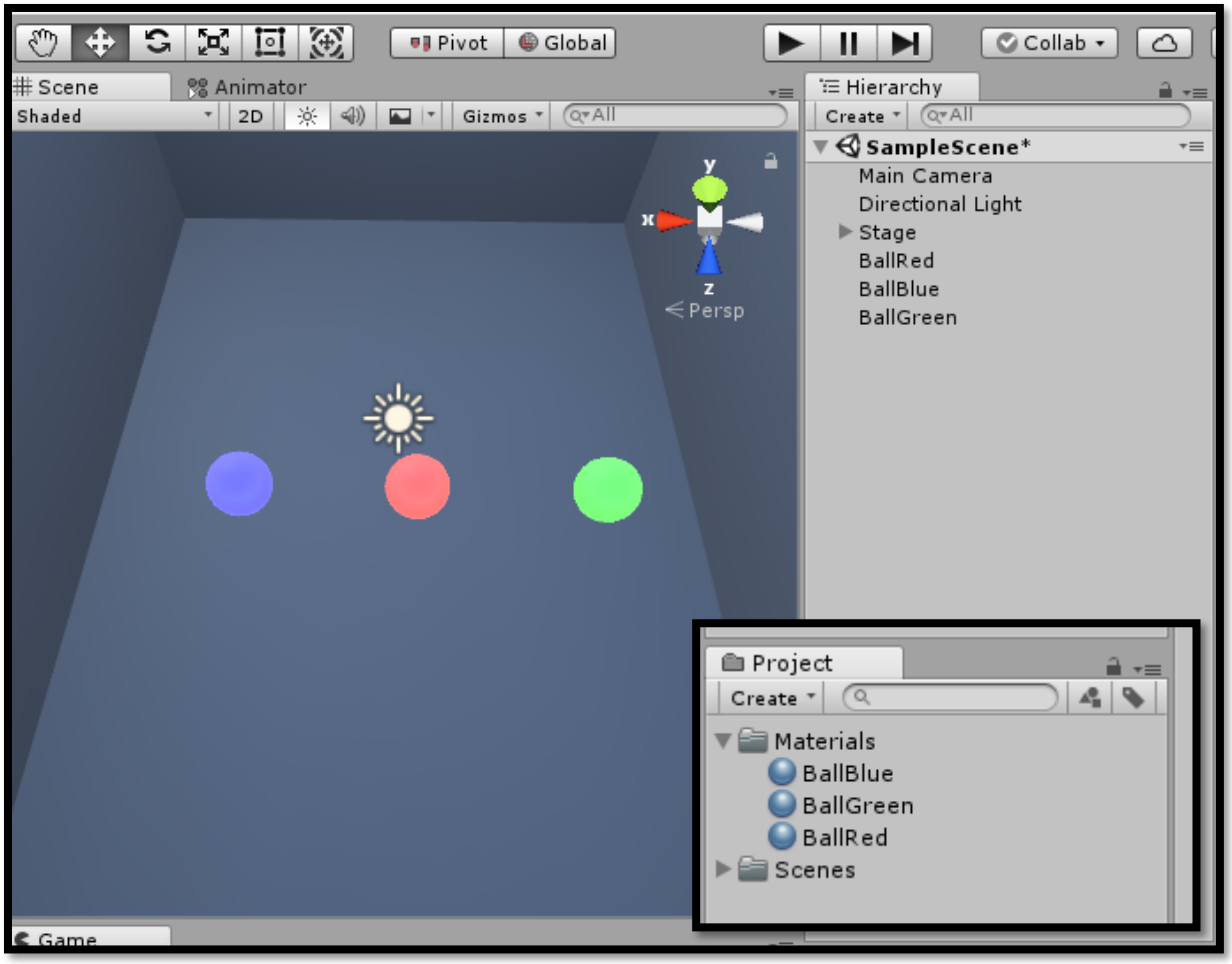




- Emission에서 Global Illumination은 Realtime으로 체크한다.



- 이제 완성된 Material을 BallRed에 드래그해서 적용한다.



- BallRed를 복제하여 각각 BallBlue, BallGreen을 만들어 3가지 공을 만든다.

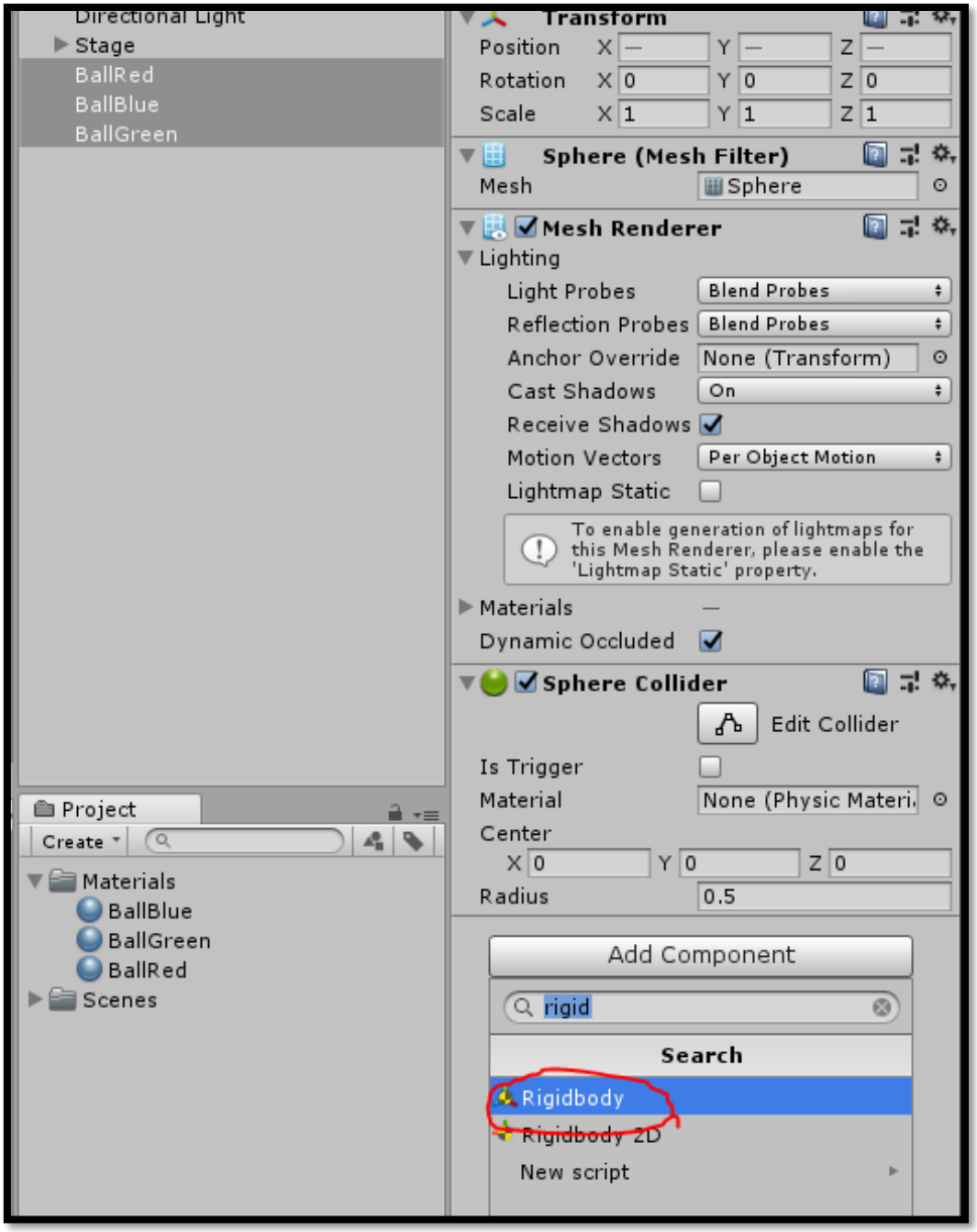
<Albedo>

	R	G	B
BallRed	255	0	0
BallBlue	0	0	255
BallGreen	0	171	0

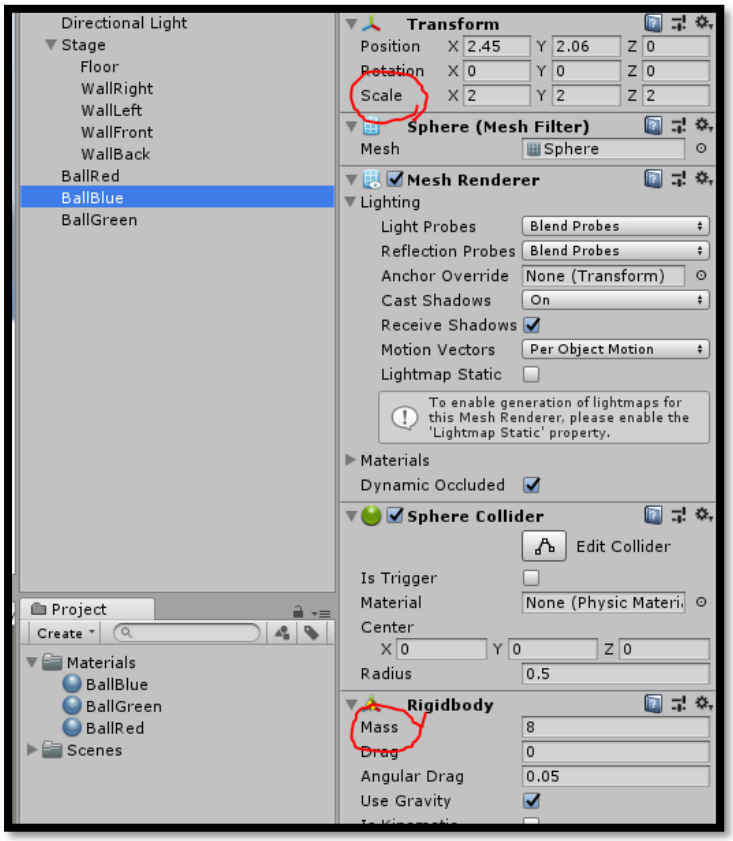
<Emission>

	R	G	B
BallRed	255	100	100
BallBlue	100	100	255
BallGreen	100	255	100

II. Rigidbody 설정

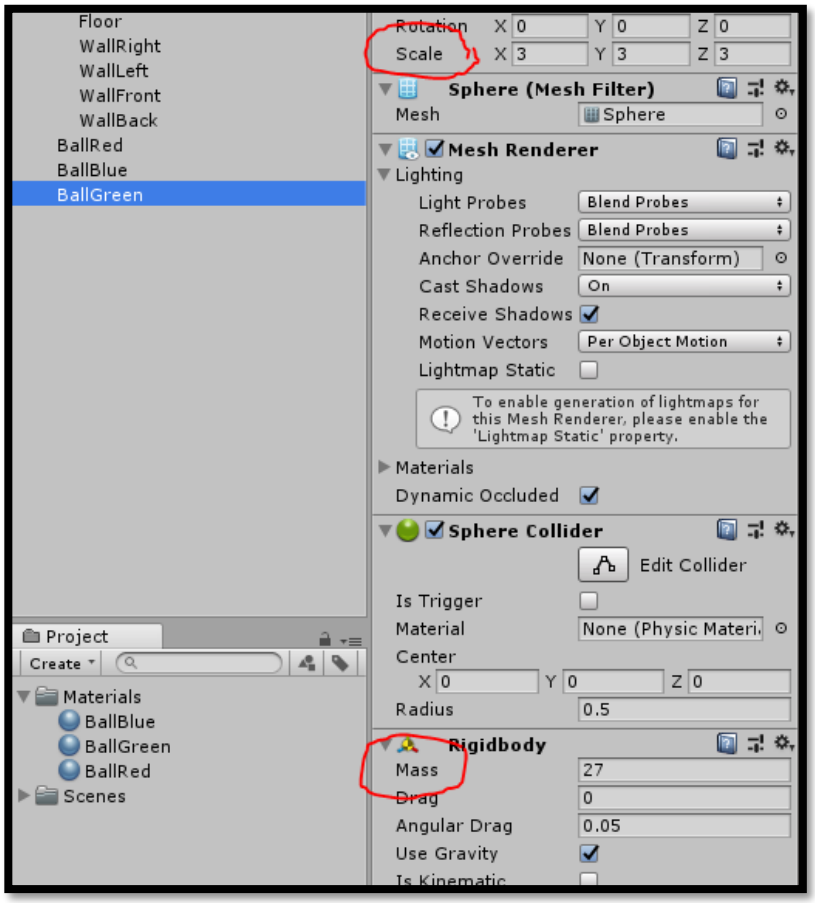


- Ball 오브젝트를 모두 선택한 후 Rigidbody 컴포넌트를 추가한다.
  (Add Component - Rigidbody)
- 이후 BallBlue와 BallGreen의 크기 및 질량을 다르게 설정한다.

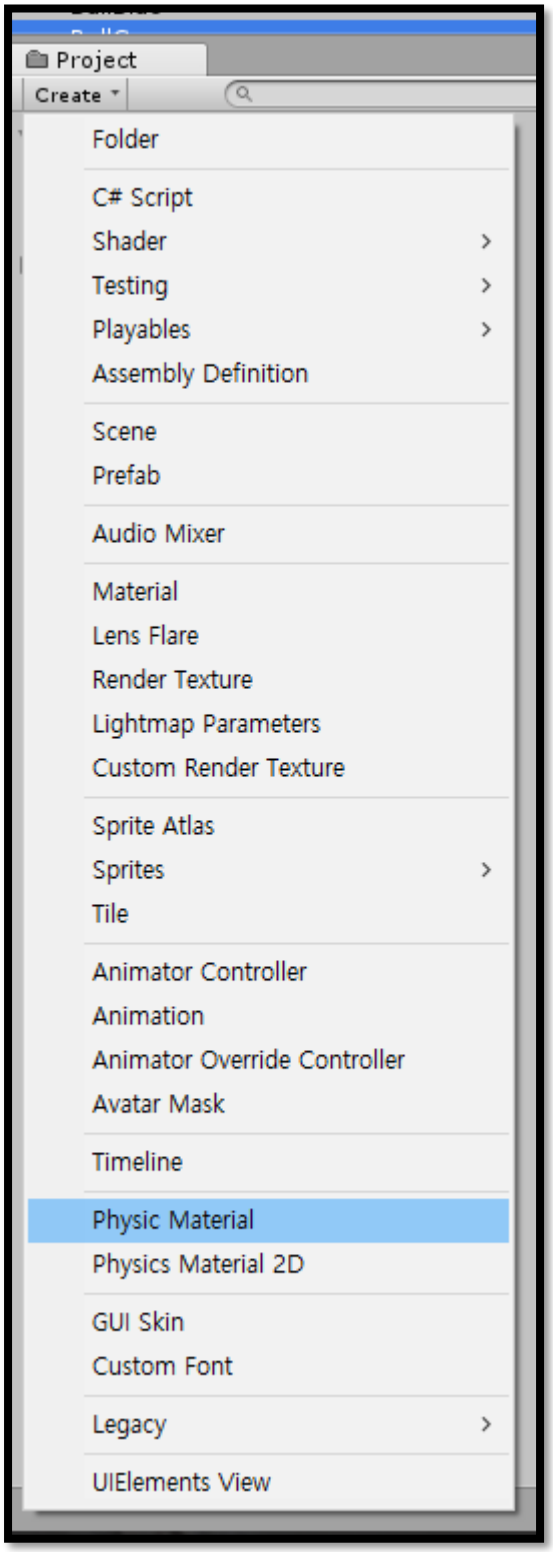


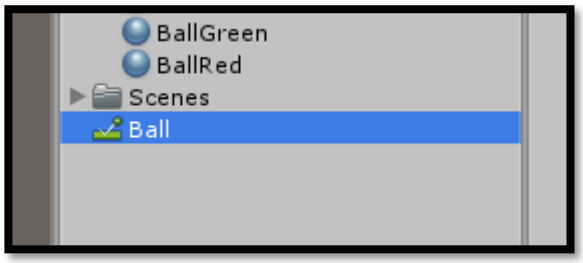
<BallBlue>
Scale: (2, 2, 2)
Mass: 8

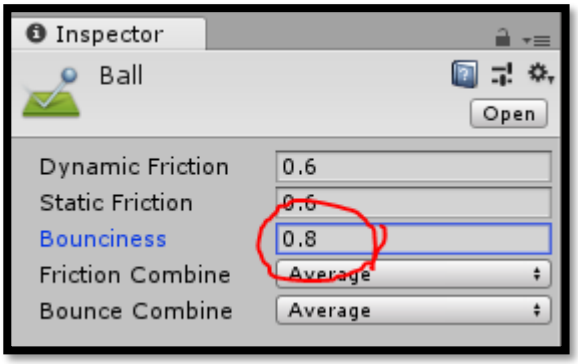
<BallGreen>
Scale: (3, 3, 3)
Mass: 27



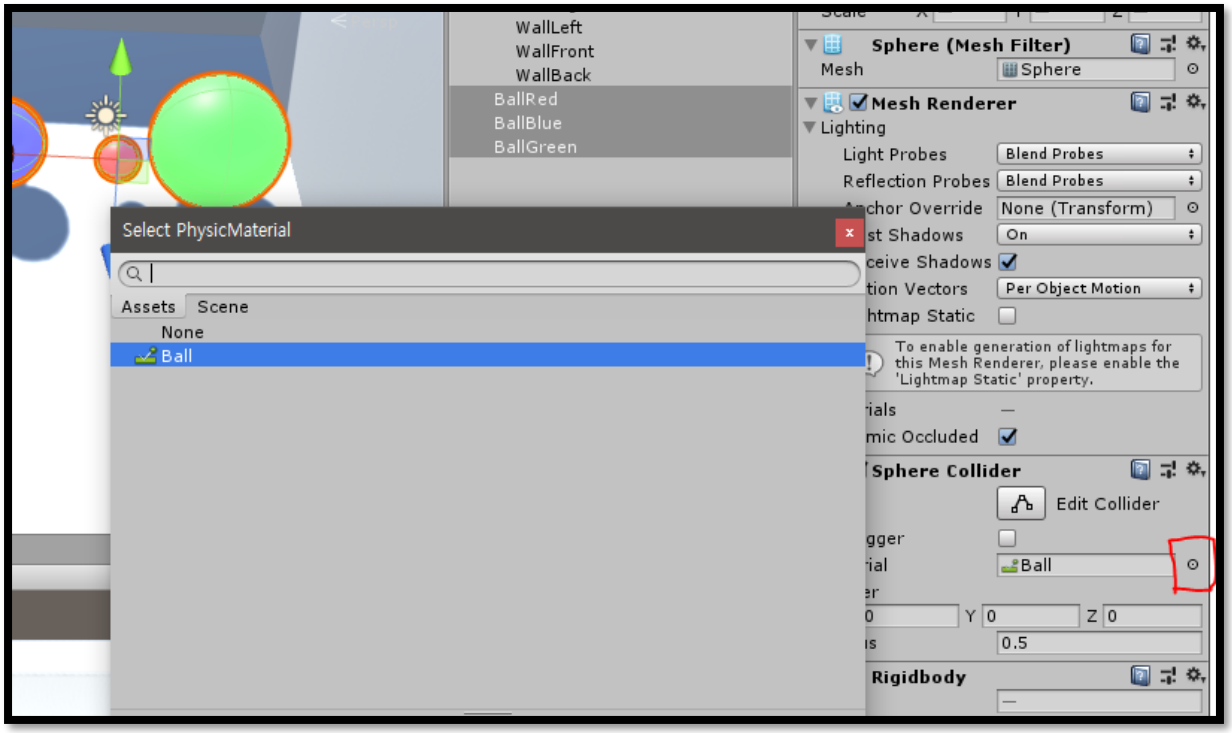
III. Physic Material 설정



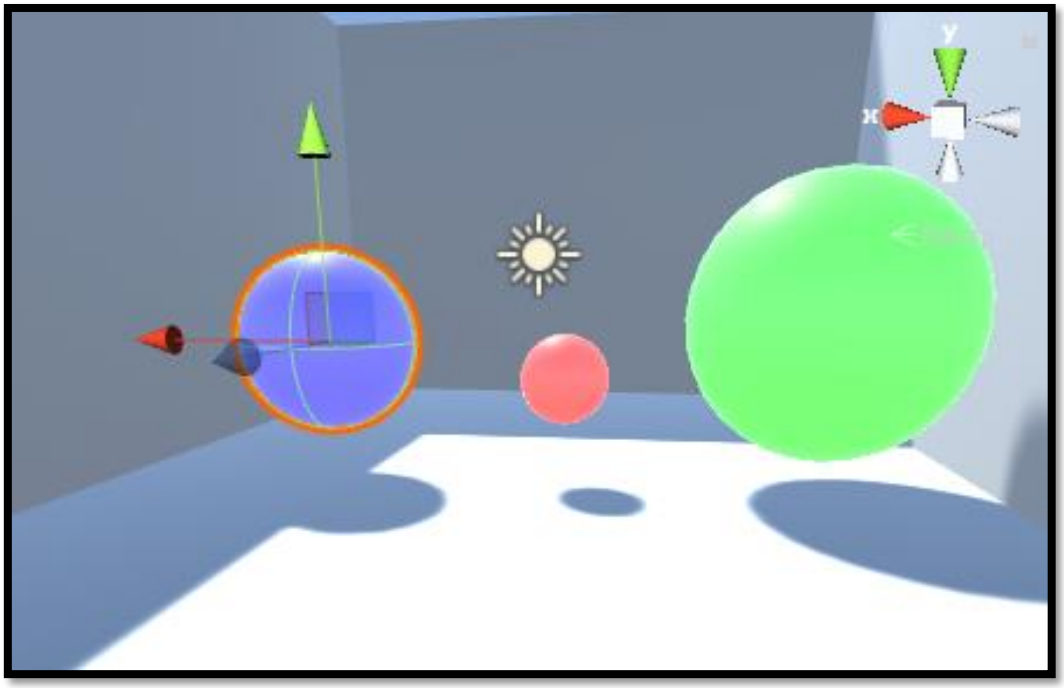




- Project 창에서 Physic Material을 만들고 Ball로 이름을 바꾼다.
- 이후 Inspector 창에서 Bounciness를 0.8로 설정한다.
  (0~1까지 설정 가능함)

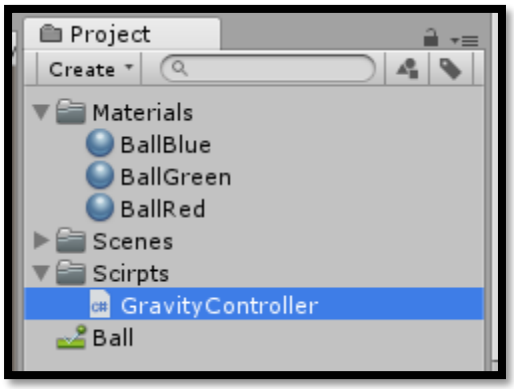


- 끝으로 Ball 오브젝트 모두 Sphere Collider의 Material에 Ball을 설정한다.



<현재 Scene View의 모습>

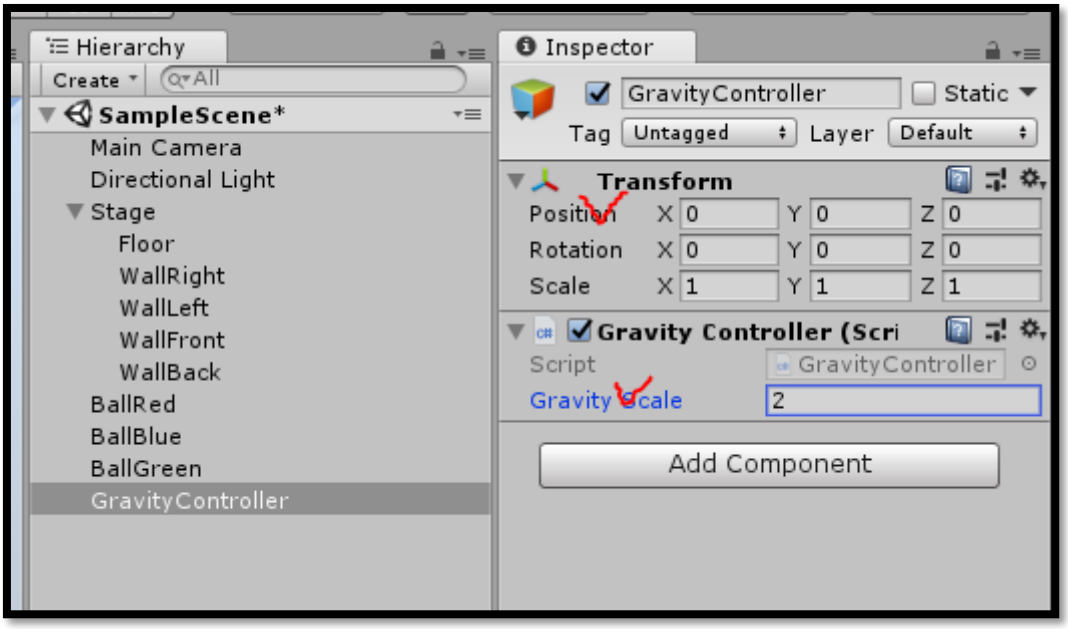
5. GravityController 생성



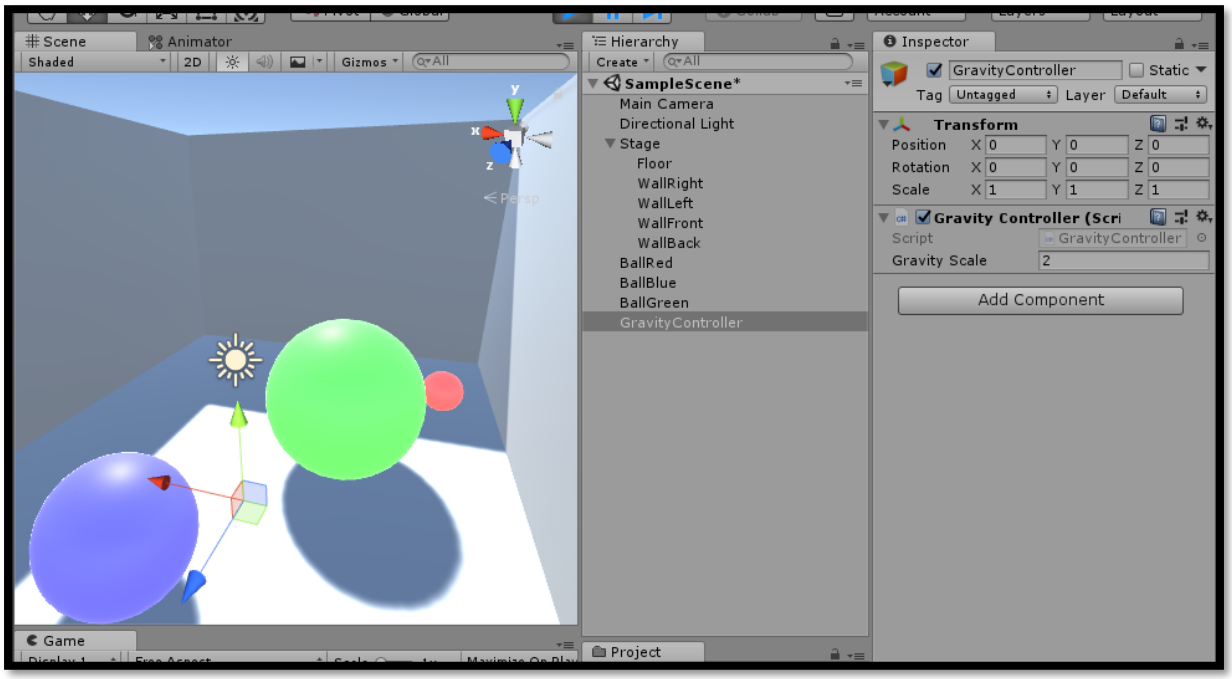
- Project 창에서 C# Script를 만들고 GravityController로 이름을 바꾼다. (Create - C# Script)
- Visual Studio로 가서 Script를 작성한다.

Assembly-CSharp | GravityController

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class GravityController : MonoBehaviour
5  {
6      // 중력 가속도
7      const float Gravity = 9.81f;
8
9      // 중력의 적용 상태
10     public float gravityScale = 1.0f;
11
12     void Update()
13     {
14
15         Vector3 vector = new Vector3();
16
17             // 키 입력을 검출하는 벡터를 설정
18             vector.x = Input.GetAxis("Horizontal");
19             vector.z = Input.GetAxis("Vertical");
20
21             // 높이 방향의 판정은 z키로 한다
22             if (Input.GetKey("z"))
23             {
24                 vector.y = 1.0f;
25             }
26             else
27             {
28                 vector.y = -1.0f;
29             }
30
31             // 혹은 이렇게도 정의할 수 있다
32             // vector.y = Input.GetKey("z") ? 1.0f : -1.0f;
33
34         // 씬의 중력을 입력 벡터의 방향에 맞추어 변화시킨다
35         Physics.gravity = Gravity * vector.normalized * gravityScale;
36     }
37 }
```

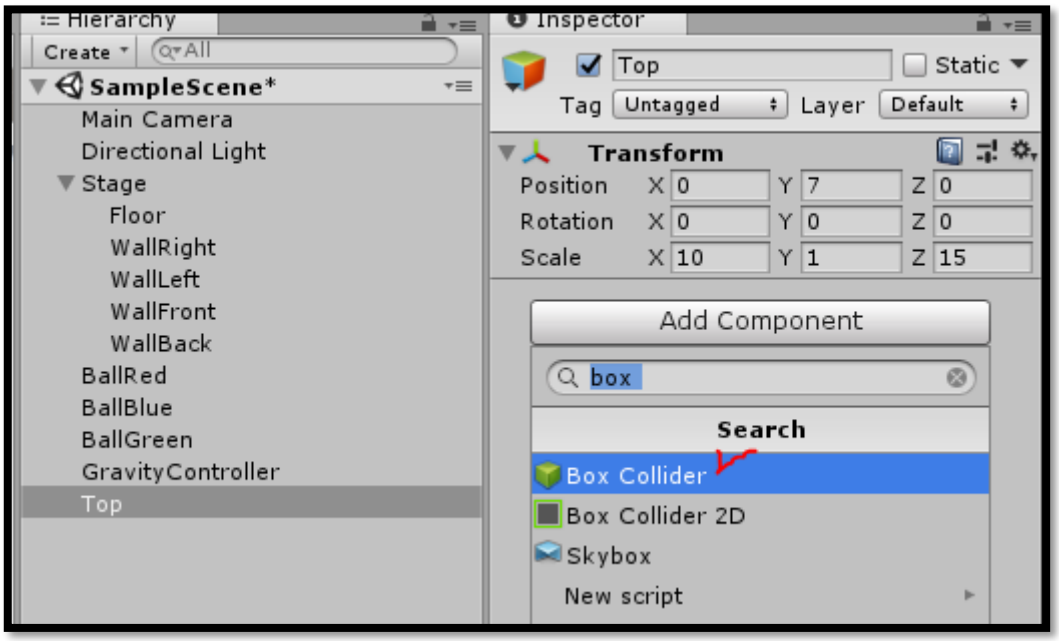


- 새로운 빈 오브젝트를 만든 후 GravityController로 이름을 바꾼다.
- GravityController Script를 지금 만든 오브젝트에 드래그해서 적용한다.
- 이후 Inspector 창에서 Position과 Gravity Scale 값을 설정한다.

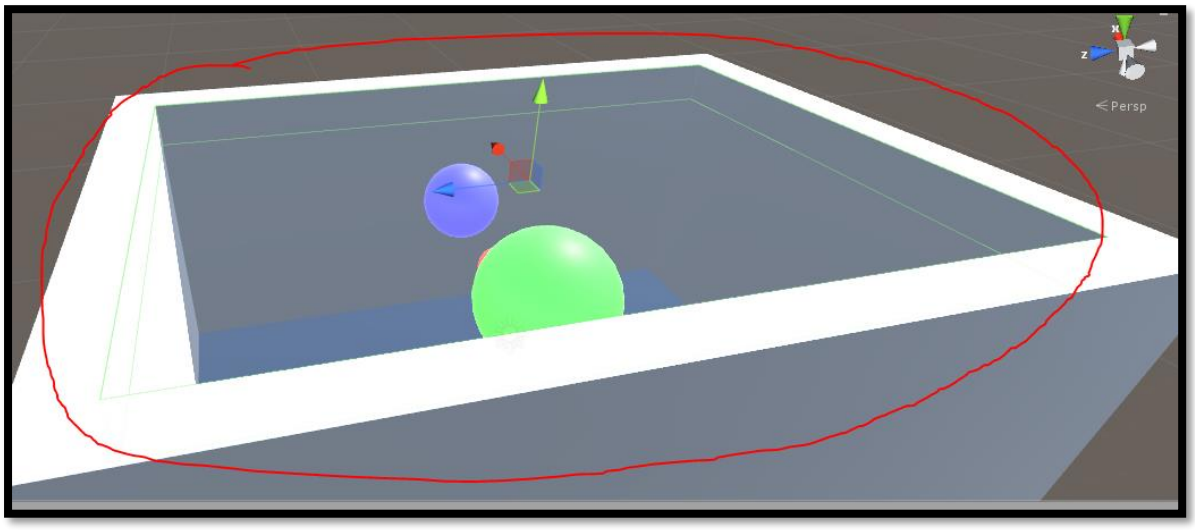


- Play 시 화살표(또는 W,A,S,D), Z키를 입력하면
  Ball이 중력의 영향을 받아 움직이는 걸 확인할 수 있다.

6. Stage 수정 및 Obstacle 추가

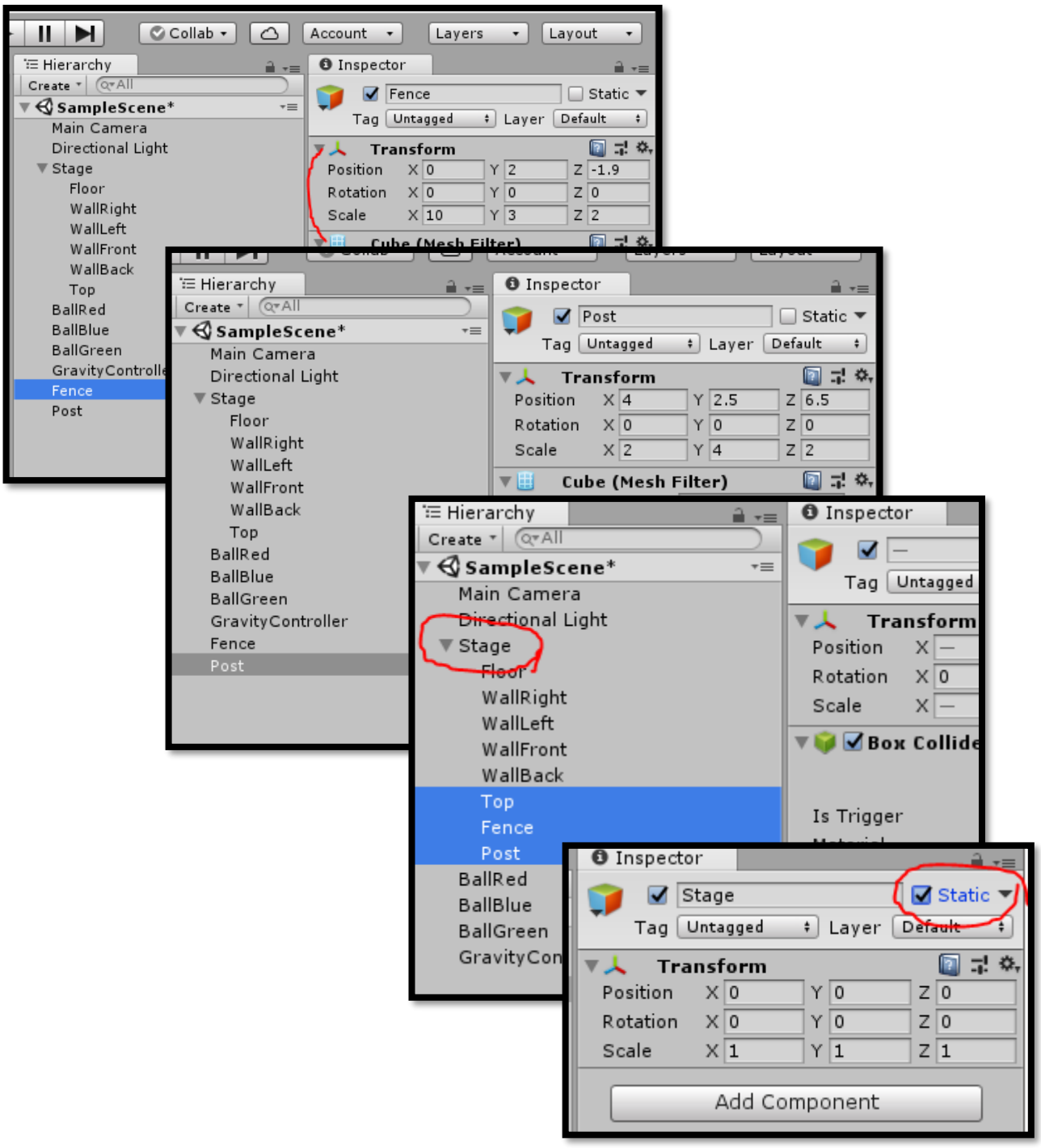


- 새로운 빈 오브젝트를 만든 후 Top으로 이름을 바꾼다.
- Position 및 Scale 값을 위와 같게 설정한 후 Box Collider를 추가한다.



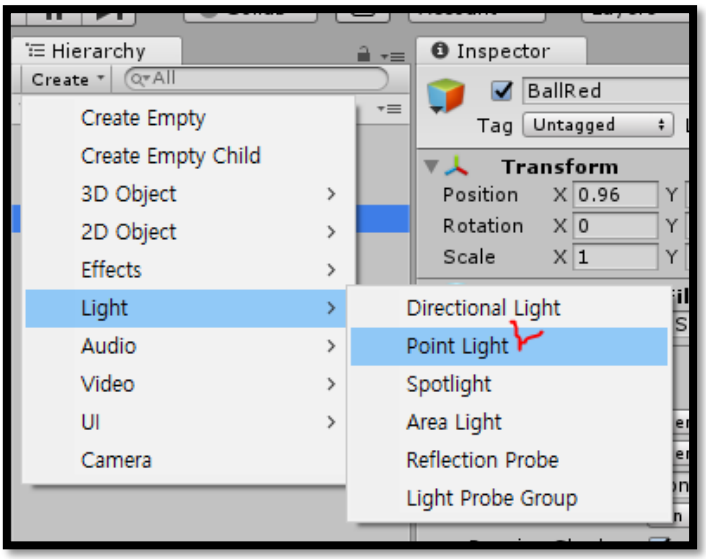
- 실제 Game View에선 보이진 않지만
  공이 Stage 밖으로 나가지 않도록 막아주는 투명 천장을 만듦.

- 먼저 2개의 Cube 오브젝트를 생성한다.

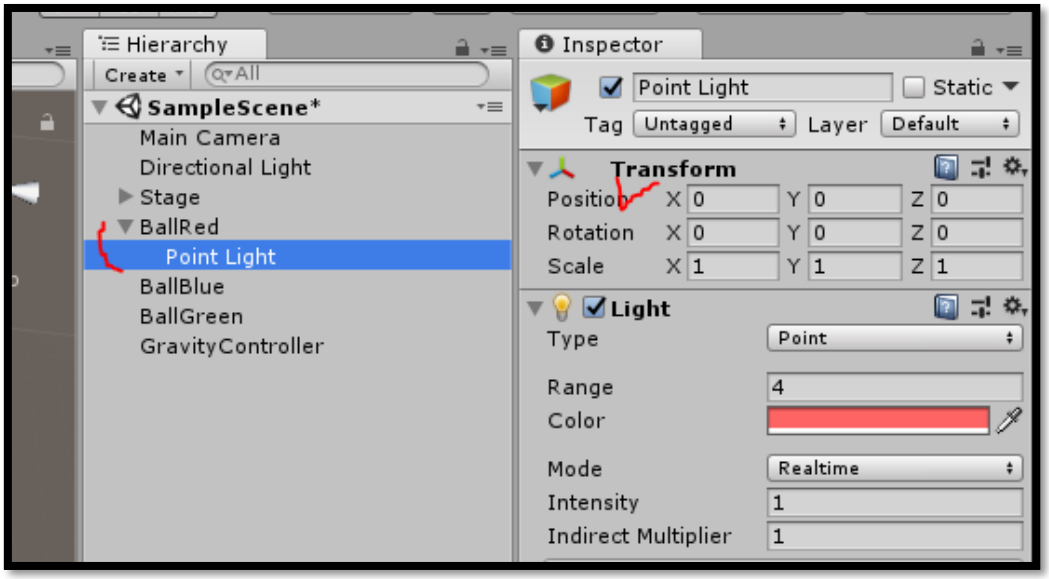


- 각각 Fence, Post로 이름을 바꾸고 Position 및 Scale 값을 위와 같이 설정
- 이후 Top, Fence, Post 오브젝트를 Stage 오브젝트에 상속 시킨다.
- 마지막으로 Stage를 Static으로 설정해서 간접 조명을 받도록 하자.

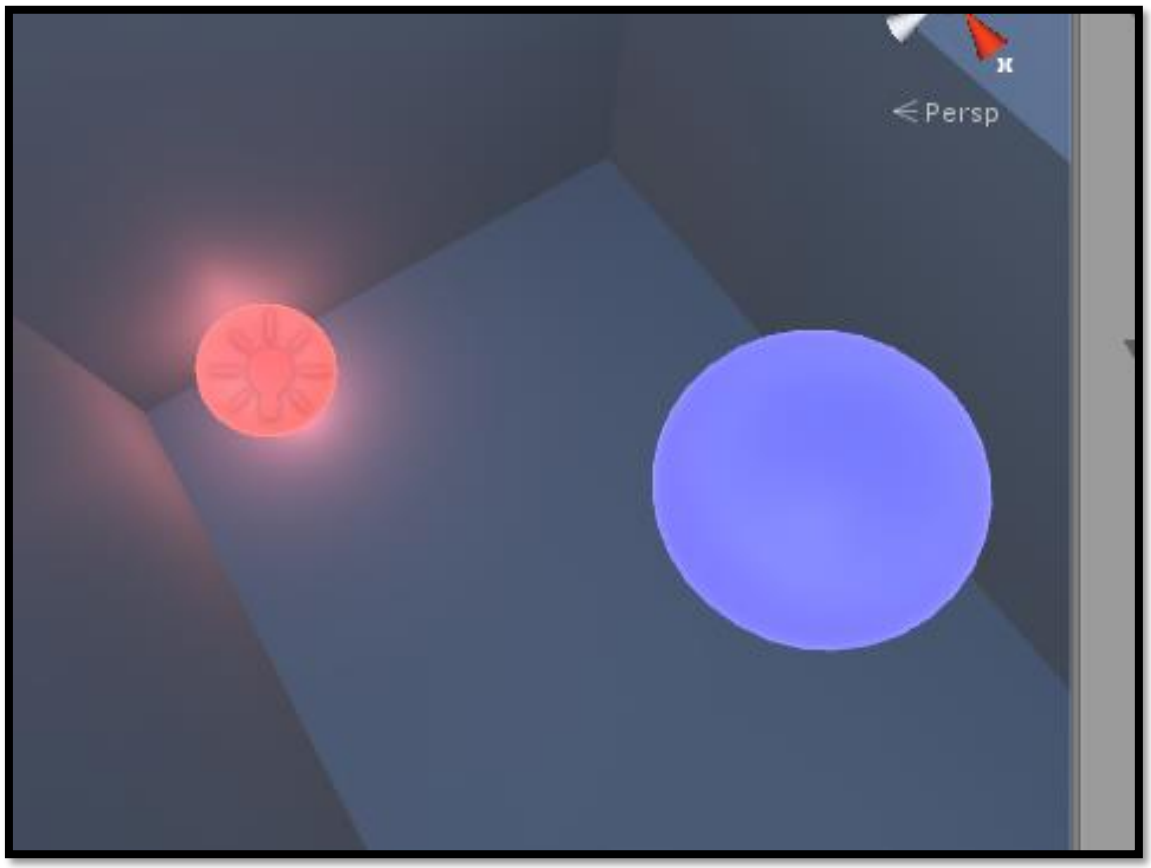
7. Point Light 추가



- Hierarchy 창에서 Point Light를 추가 (Create - Light - Point Light)
- 이후 Inspector 창에서 Transform, Light 컴포넌트를 설정한다.
- Range 값이 커질수록 밝게 빛나므로 적당히 4로 설정한다.
- Color는 BallRed Material의 Emission 값과 같게 한다.

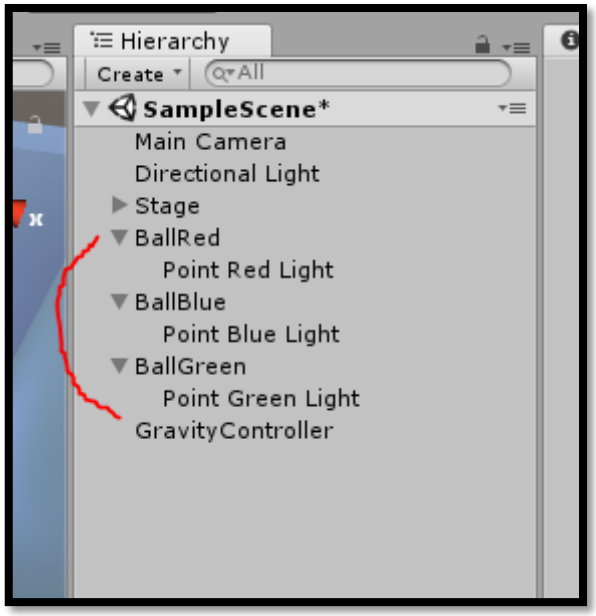


- 이후 Point Light를 BallRed에 상속 시키고 Position을 원점으로 둔다.

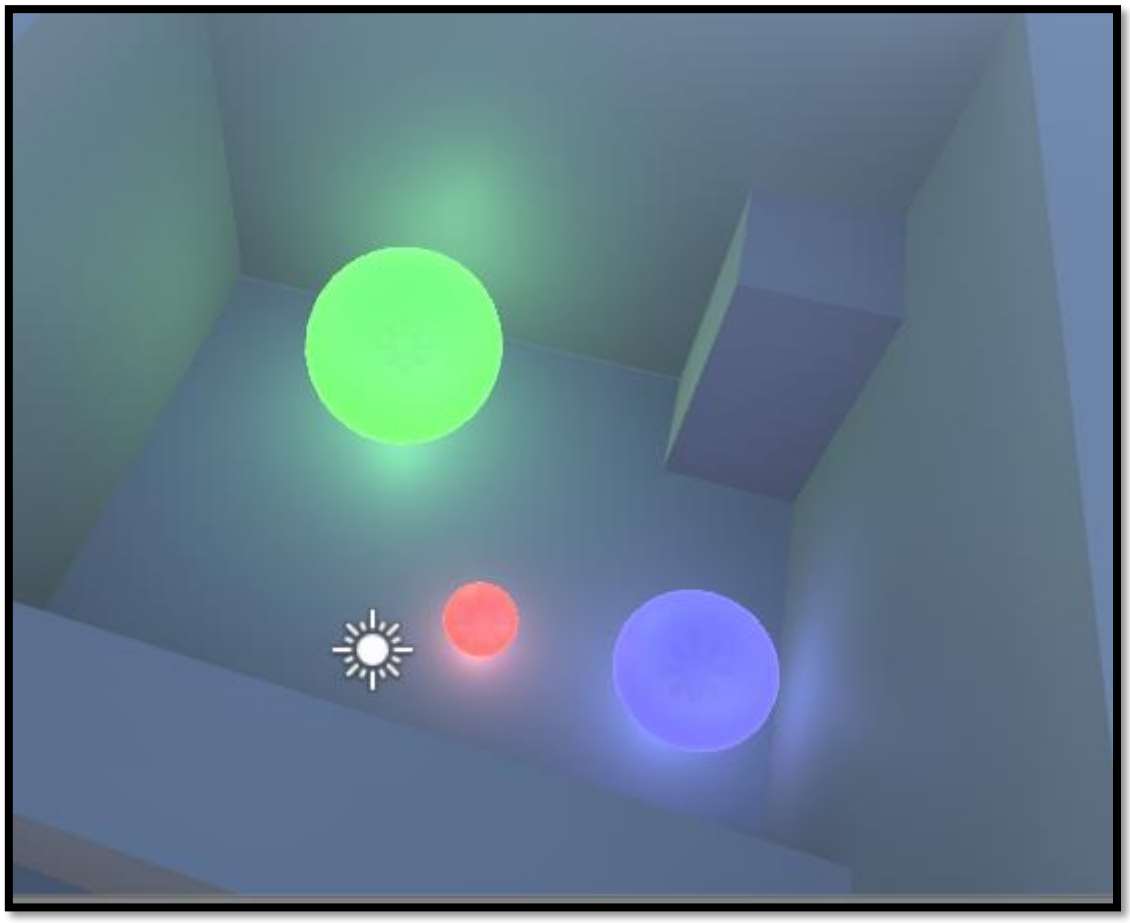


<이제 BallRed가 스스로 빛나고 있는 걸 확인할 수 있음>

- 다음으로 Point Blue Light, Point Green Light를 추가해서 Transform, Light 컴포넌트에 각각 알맞은 수치를 입력한다.



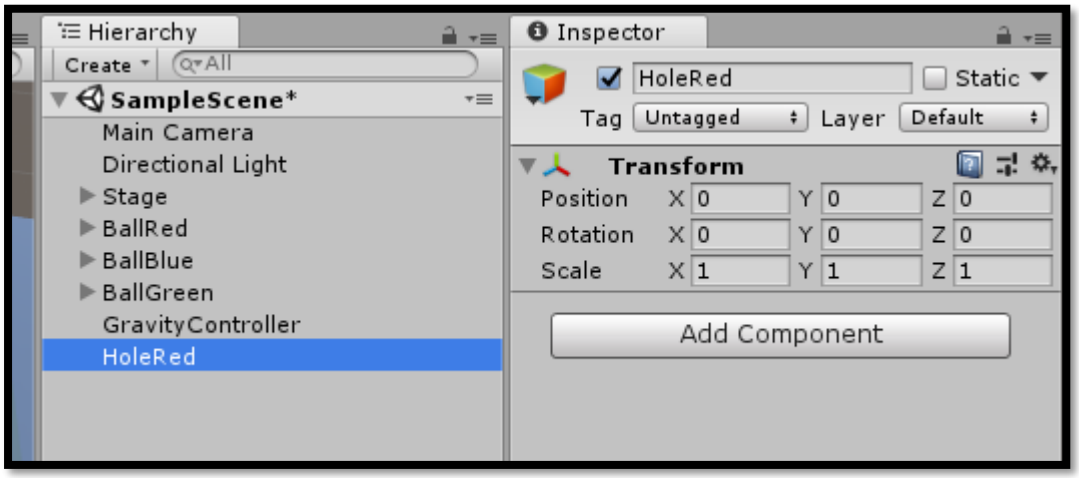
- 만든 Point Light를 각자 알맞은 Ball에 상속 시킨다.

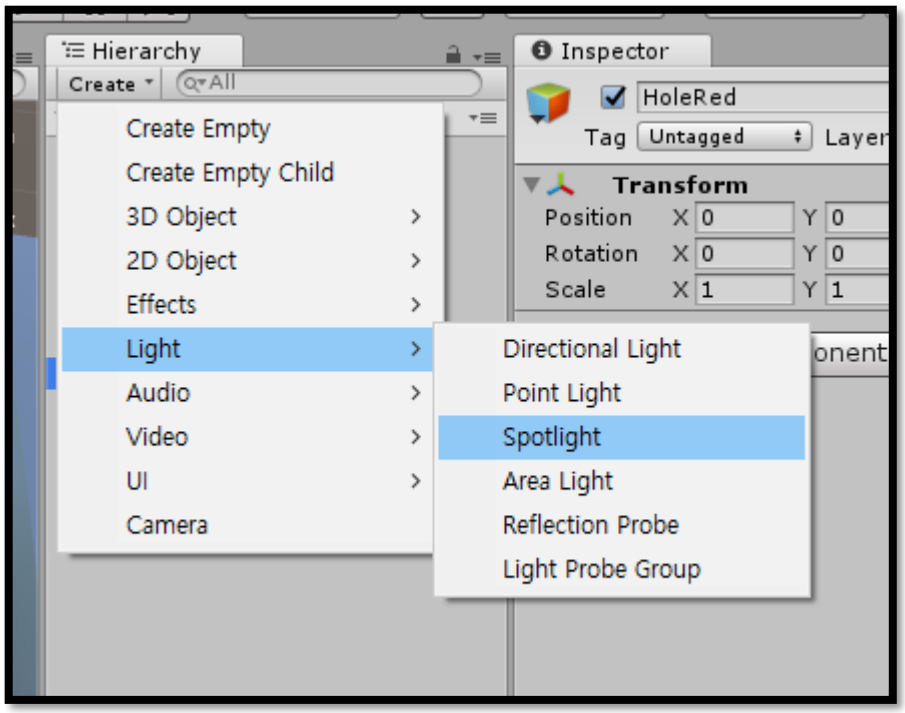


<이제 모든 Ball에서 빛이 나고 있음>

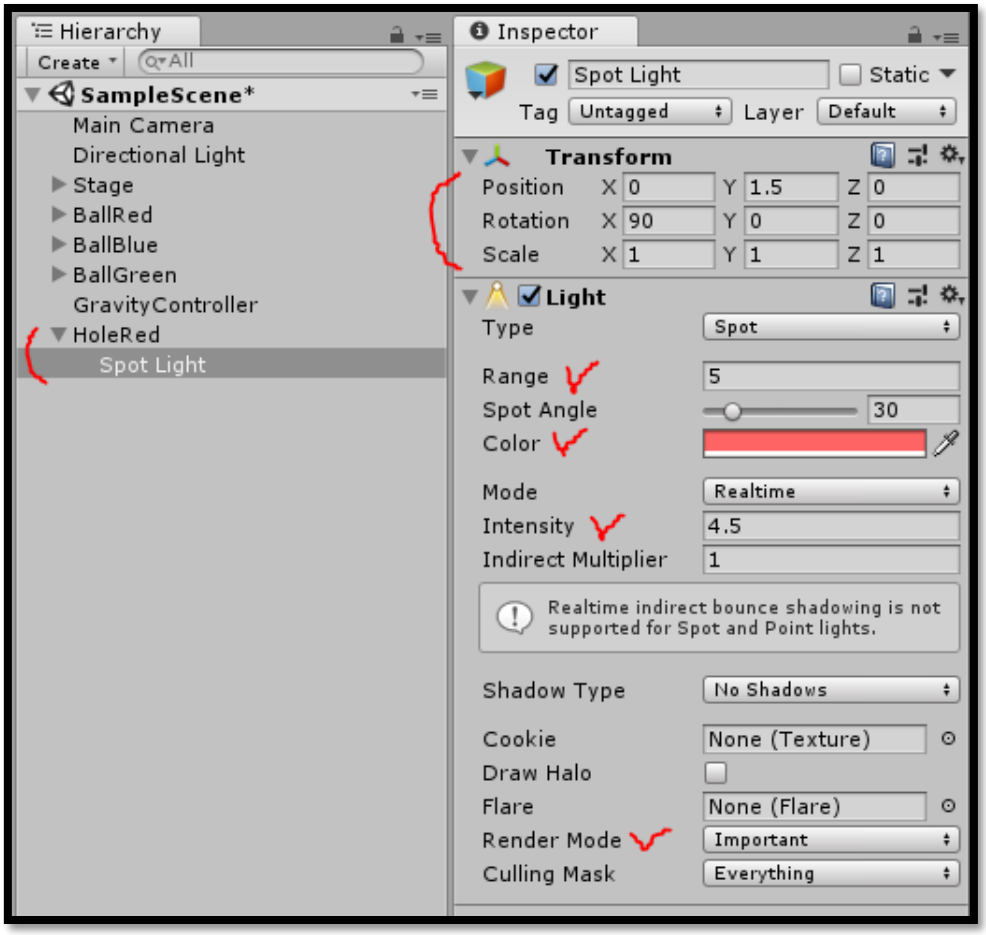
8. Hole 생성

1. Hole 생성 및 Spot Light 추가



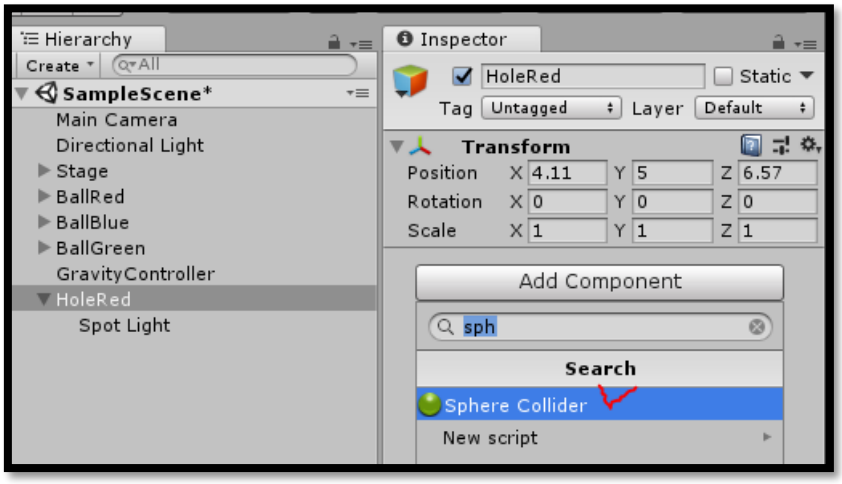


- 새로운 빈 오브젝트를 만든 후 HoleRed로 이름을 바꾼다.
- 그리고 Hierarchy 창에서 Spotlight를 추가한다.

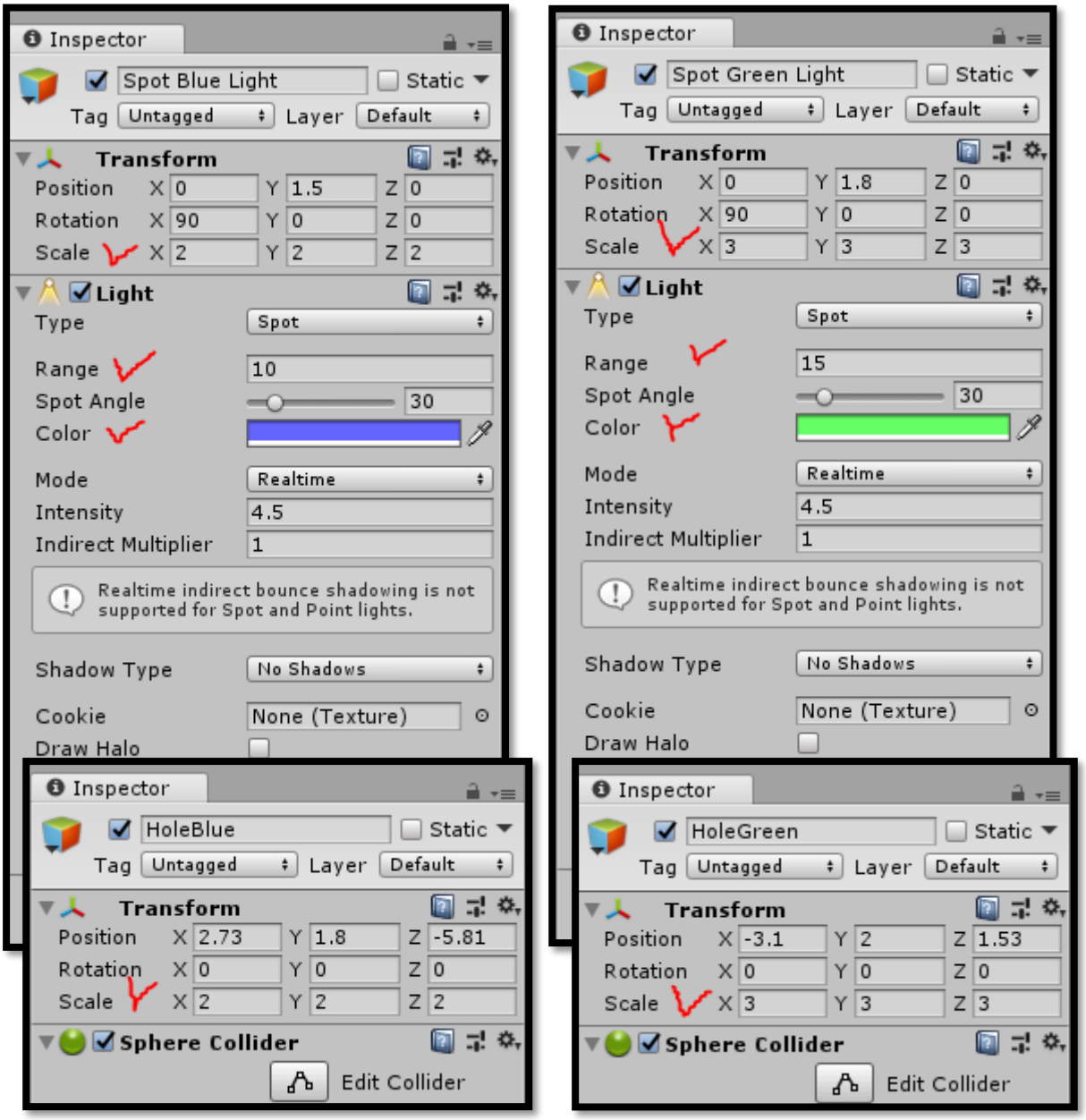


- 먼저 Spot Light를 HoleRed에 상속 시키고
  이후 Transform과 Light의 컴포넌트에 알맞은 수치를 입력한다.

- 이번 게임은 빛이 가장 중요하므로 Render Mode는 Important로 설정한다.
  (더 높을수록 게임 내에서 더 자세하게 Lighting 표현이 됨)

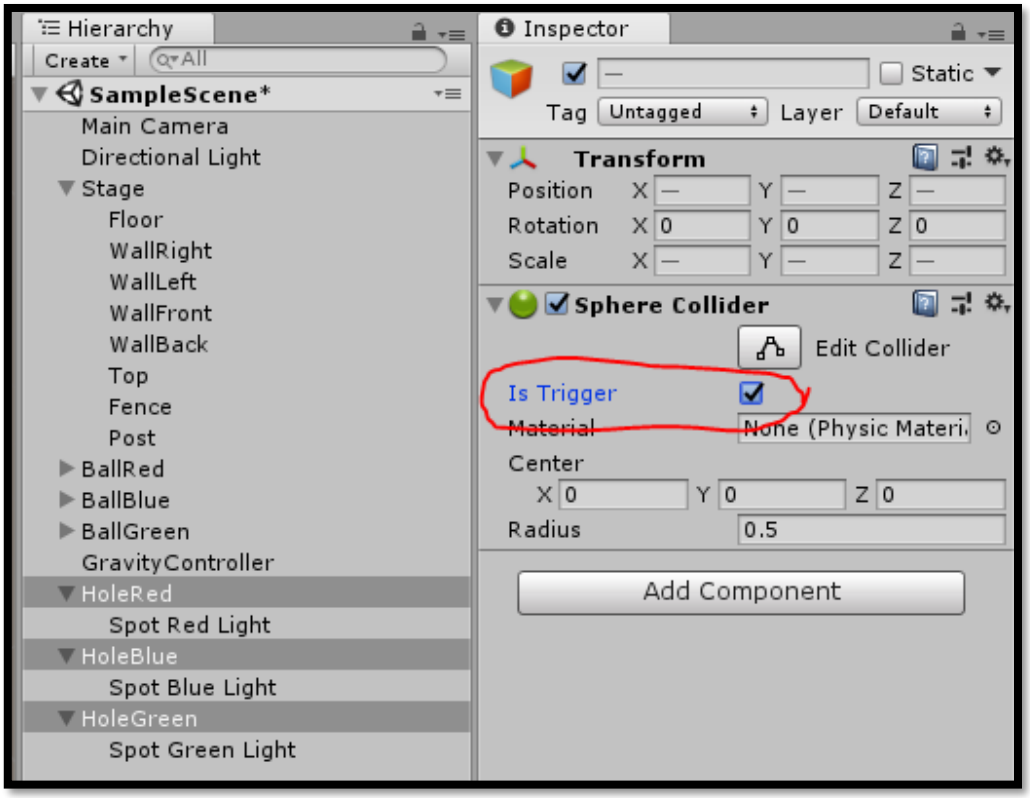


- 그 후 HoleRed에 Sphere Collider 컴포넌트를 추가한다.

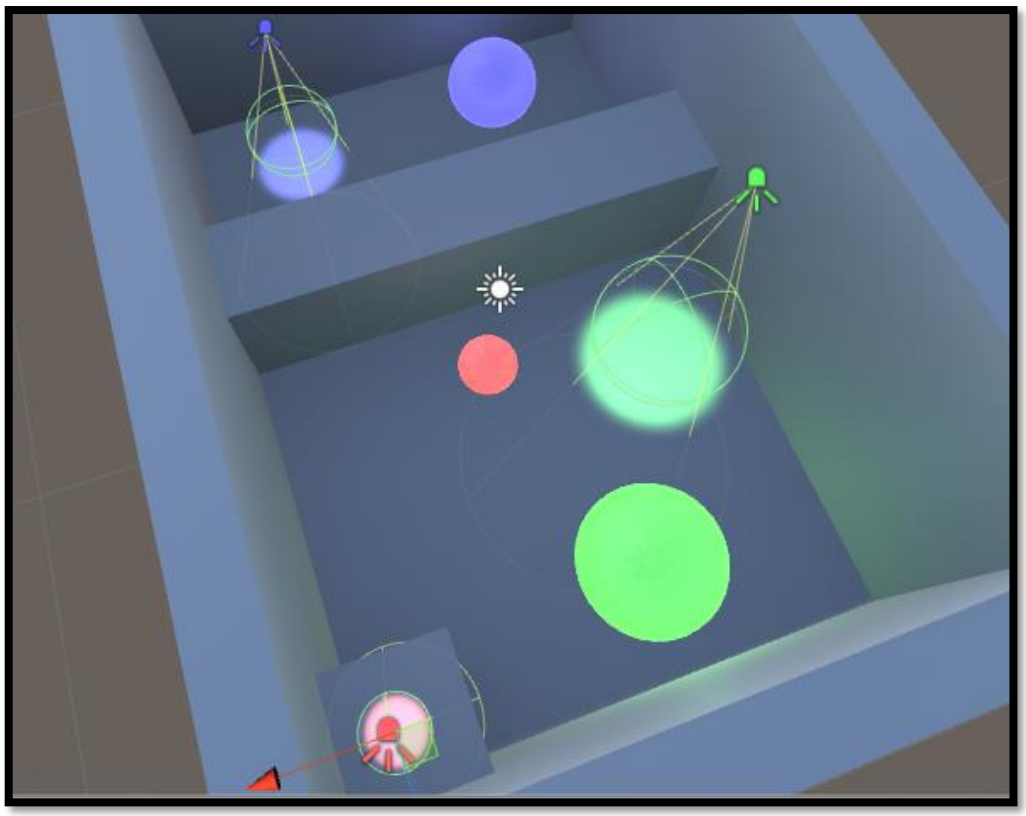


- 마찬가지로 Spot Blue Light, Spot Green Light를 추가해서
  Transform, Light 컴포넌트에 각각 알맞은 수치를 입력하고 상속 시킨다.

- 각각 HoleBlue, HoleGreen의 크기를 2, 3으로 수정한다.

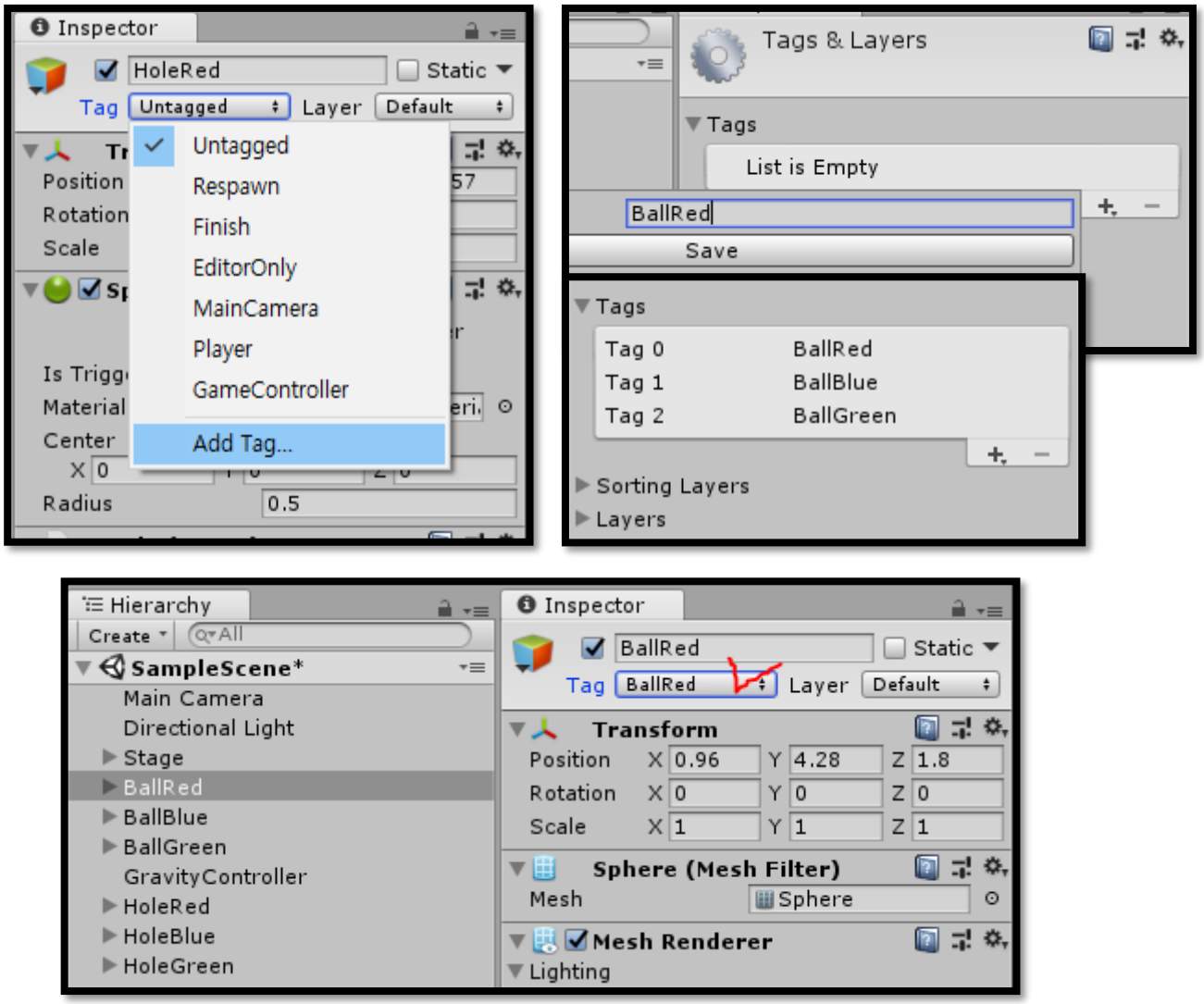


- 그 다음 모든 Hole을 Is Trigger를 활성화 시켜 트리거로 만든다.



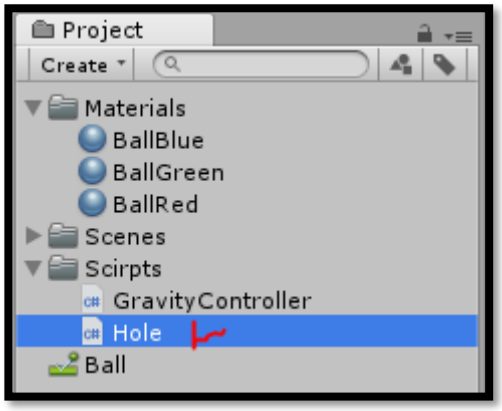
- 마지막으로 모든 Hole의 Position을 Stage의 적당한 위치로 옮긴다.

II. Tag 추가

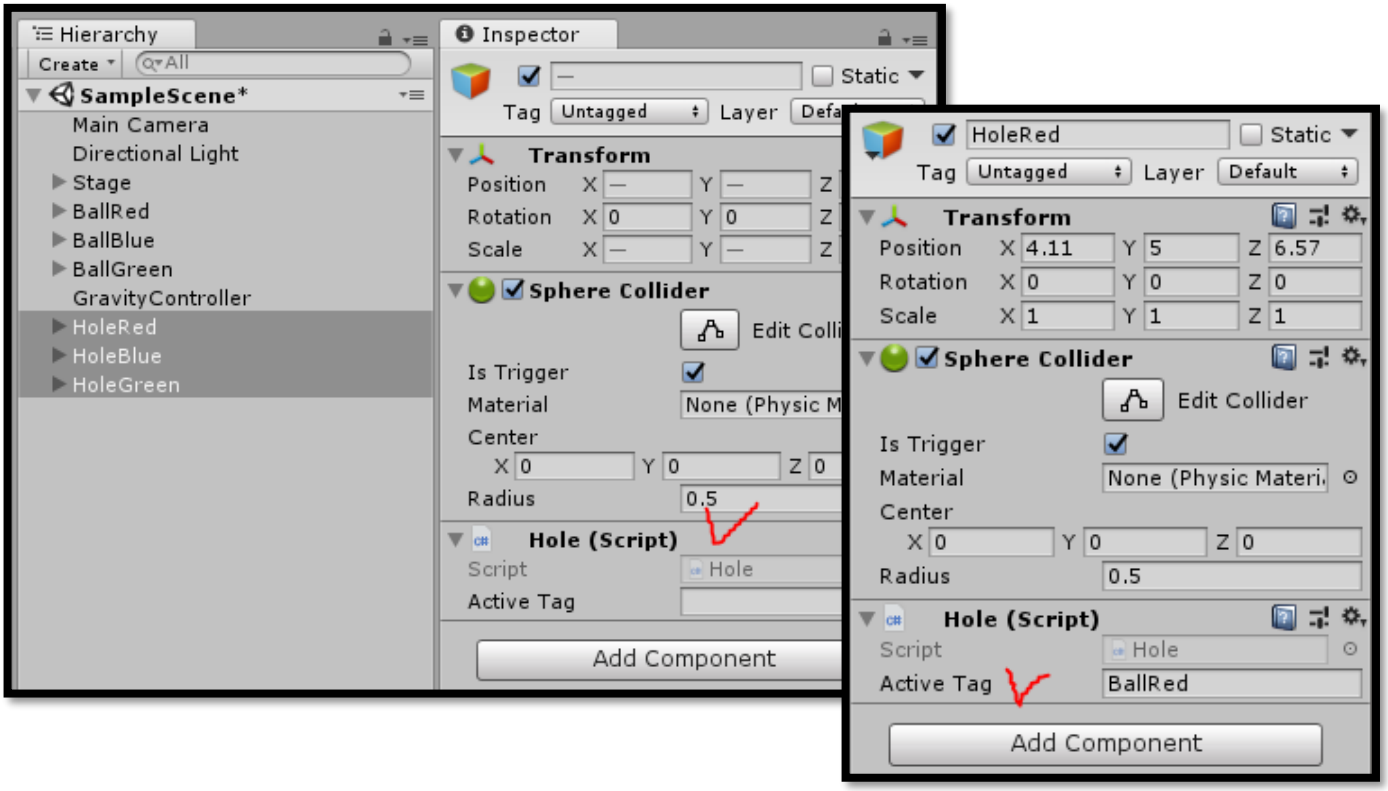


- Inspector 창에서 Tag를 추가(Add Tag)
- +버튼을 눌러 BallRed, BallBlue, BallGreen 추가
- 이후 각 Ball 마다 알맞은 Tag 설정

III. Script 작성



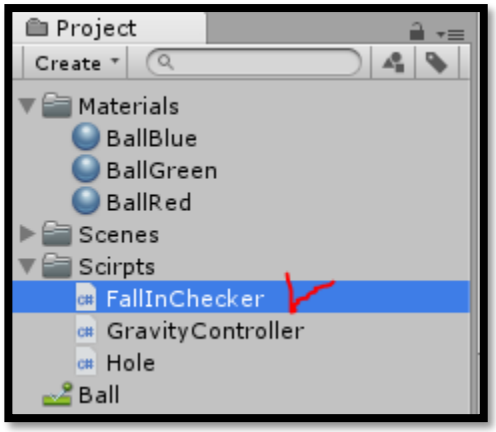
- Project 창에서 C# Script를 만든 후 Hole로 이름을 바꾼다.
- Visual Studio로 가서 Script를 작성한다. (다음 장 참고)



- Script 작성 후 모든 Hole에게 Hole Script를 적용시키고 알맞은 Tag를 적어 넣는다.

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

public class Hole : MonoBehaviour {



    bool fallIn; // 공이 Hole 안에 들어와 있는지 체크할 boolean 변수
    public string activeTag; // 같은 색 공을 Hole에 넣기 위해 태그 지정


    // 공이 Hole 안에 들어와 있는지 확인해주는 함수, fallIn을 반환함
    public bool IsFallIn()
    {
        return fallIn;
    }


    // 모든 Hole은 Sphere Collider에서 IsTrigger를 활성화 했으므로 Trigger로 작동하게 됨

    // Hole로 무언가(공)이 들어왔을 때(Enter) 호출 되는 함수
    private void OnTriggerEnter(Collider other)
    {
        // 접촉한 공의 태그가 Hole의 activeTag와 같을 경우, fallIn을 true로 설정
        if (other.gameObject.tag == activeTag)
            fallIn = true;
    }

    // Hole로 무언가(공)이 나갈 때(Exit) 호출 되는 함수
    private void OnTriggerExit(Collider other)
    {
        // 접촉한 공의 태그가 Hole의 activeTag와 같을 경우, fallIn을 false로 설정
        // 태그가 서로 다른 경우에는 아무런 변화가 일어나지 않음
        if (other.gameObject.tag == activeTag)
            fallIn = false;
    }

            fallIn = false;
        }

        // Hole로 무언가(공)이 머무를 때(Stay) 호출 되는 함수
        private void OnTriggerStay(Collider other)
        {
            // 공의 Rigidbody 컴포넌트를 지역변수 r로 설정
            Rigidbody r = other.gameObject.GetComponent<Rigidbody>();

            // Hole의 현재 위치와 공의 위치의 차를 이용해 (Hole 쪽으로 향하는) 방향을 설정함
            Vector3 direction = transform.position - other.gameObject.transform.position;

            // 방향의 크기를 1로 만듦 (Normalize() 메소드와 normalized 멤버 변수의 차이를 알아 두자
            direction.Normalize();

            // 공의 태그가 Hole의 activeTag와 같을 경우
            if (other.gameObject.tag == activeTag)
            {
                // 공의 속력을 점차 줄임
                r.velocity *= 0.9f;

                // 공을 Hole 방향으로 힘을 가함
                r.AddForce(direction * r.mass * 20.0f);
            }
            // 태그가 같지 않을 경우
            else
                // 공을 Hole의 반대 방향으로 힘을 가함
                r.AddForce(-direction * r.mass * 80.0f);
        }

}
```

9. UI 생성

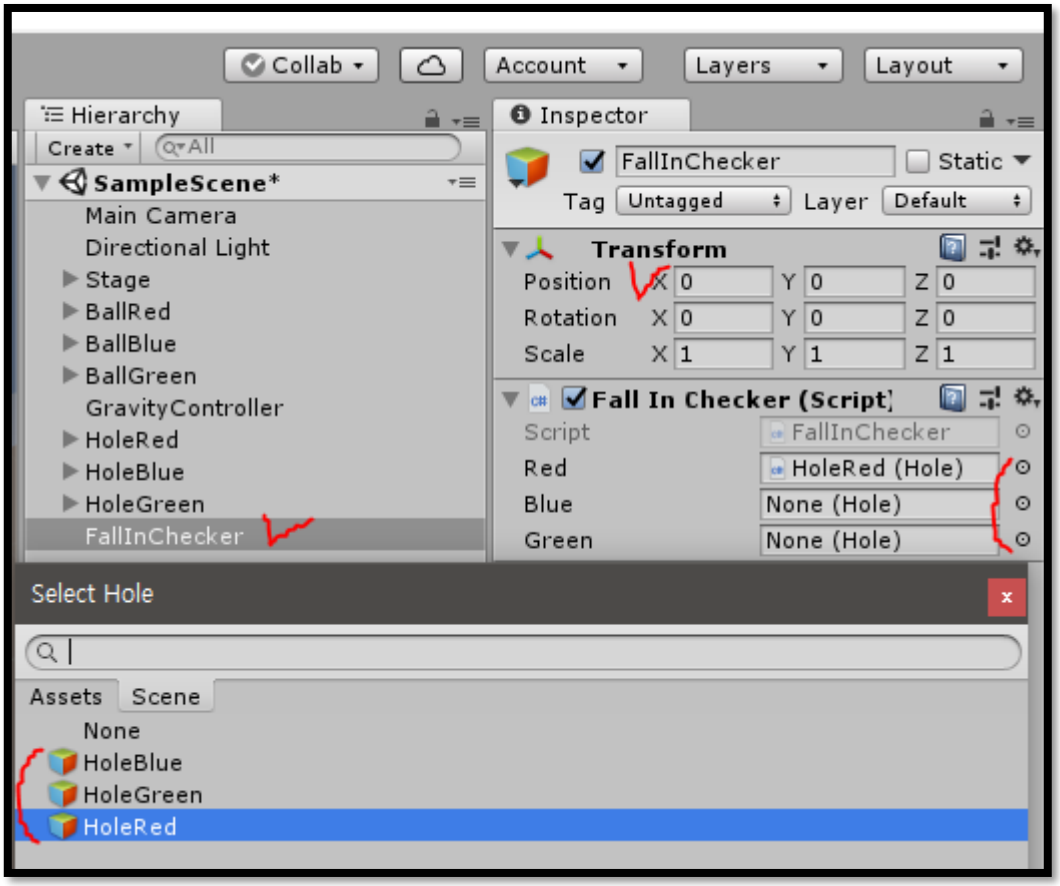


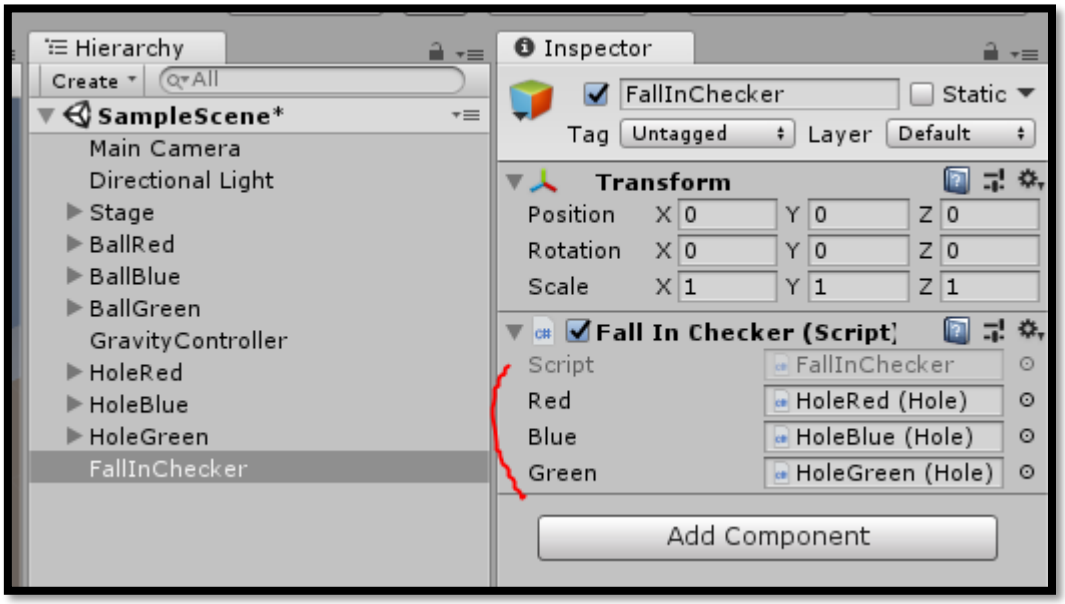
- Project 창에서 C# Script를 만든 후 FallInChecker로 이름을 바꾼다.
- Visual Studio로 가서 Script를 작성한다.

FallInChecker.cs* | Hole.cs | GravityController.cs

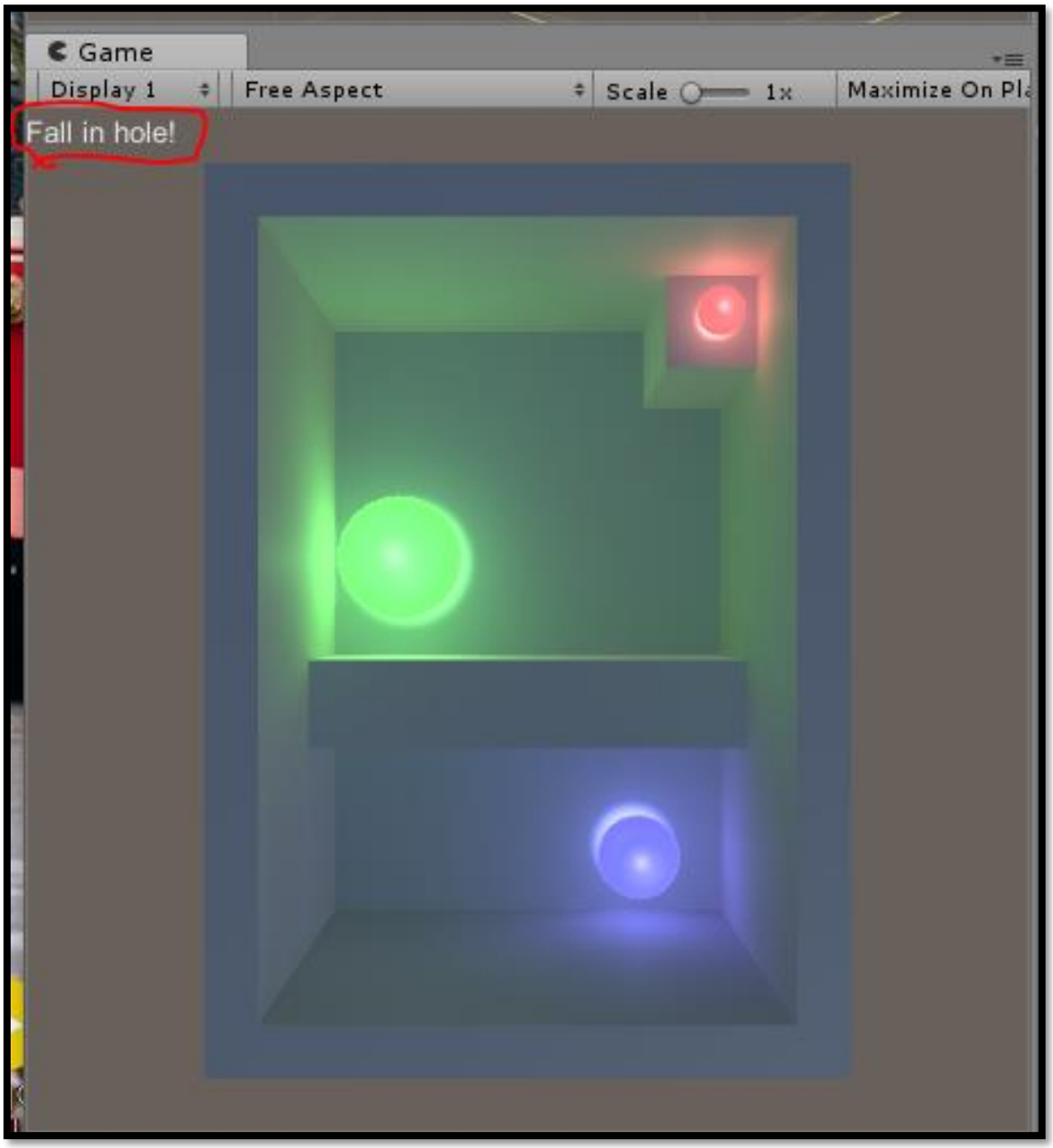
Assembly-CSharp | FallInChecker

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class FallInChecker : MonoBehaviour
5  {
6      // 각 Hole에 대해 지정
7      public Hole red;
8      public Hole blue;
9      public Hole green;
10
11     private void OnGUI()
12     {
13         // 화면에 띄울 문구를 string 변수로 선언
14         string label = " ";
15
16         // 모든 Hole 안에 공이 있을 경우 label을 아래와 같이 바꿈
17         if (red.IsFallIn() && blue.IsFallIn() && green.IsFallIn())
18             label = "Fall in hole!";
19
20         // 좌측 상단에 100 * 30 크기의 직사각형에 label을 표시
21         // 공이 하나라도 안들어가 있을 경우엔 " "이 출력
22         // 공이 다 들어가 있을 경우엔 "Fall in hole!"이 출력
23         GUI.Label(new Rect(0, 0, 100, 30), label);
24     }
25 }
26
```





- 새로운 빈 오브젝트를 만든 후 FallInChecker로 이름을 바꾼다.
- 그 다음 FallInChecker Script를 방금 만든 오브젝트에 드래그해서 적용한다.
- 마지막으로 알맞은 색깔의 Hole을 지정해준다.



- 이제 Ball이 각자 자기 색깔과 같은 Hole에 빨려 들어가고
  셋 다 들어 갔을 시 좌측 상단에 Fall in hole! 이라는 문구가 나타난다

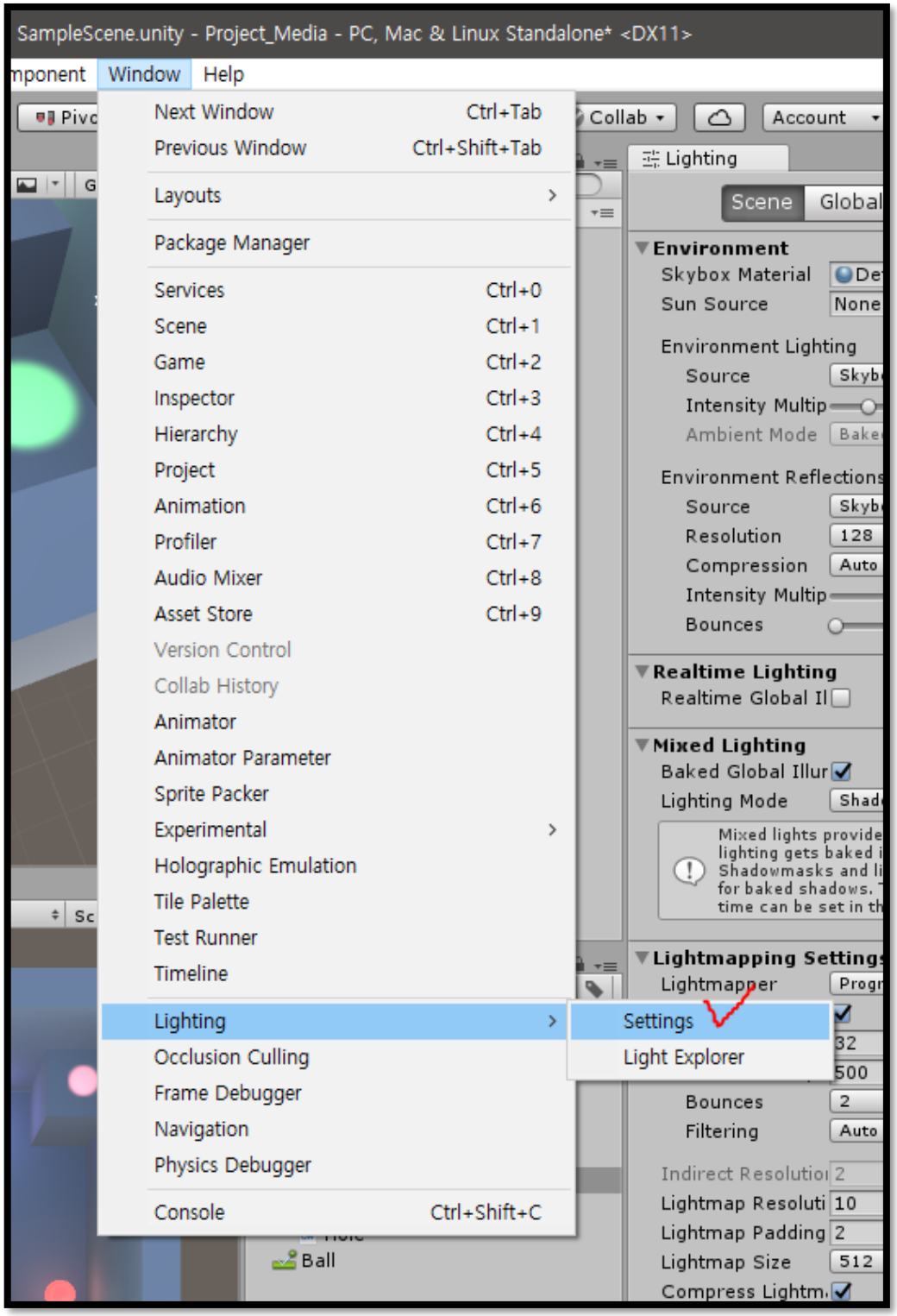
10. 마무리

1. GravityController Script 수정

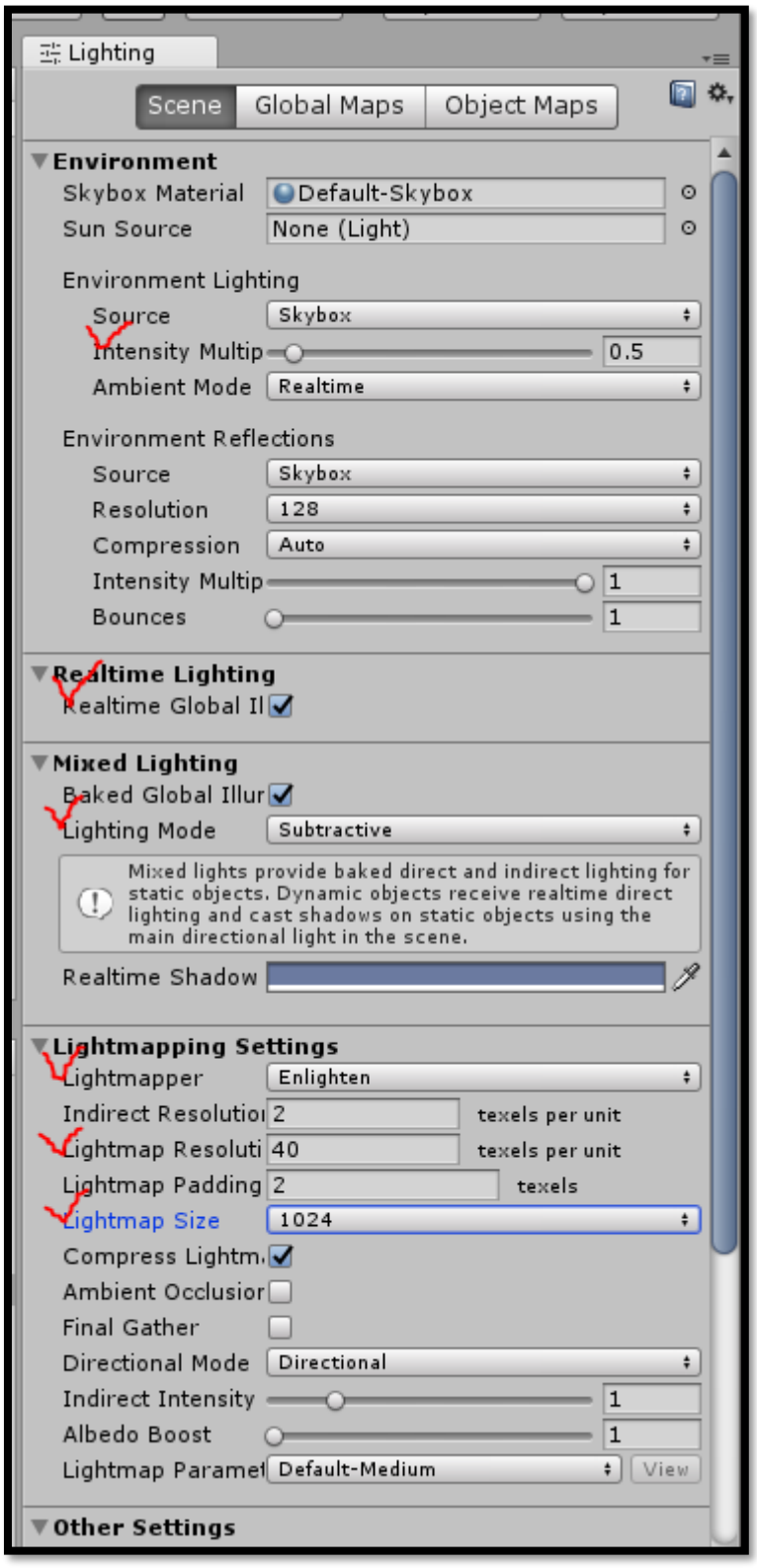
```
12      void Update()
13      {
14
15          Vector3 vector = new Vector3();
16
17          // Unity 에디터에서 Debug 시 사용할 방향키
18          if (Application.isEditor)
19          {
20              // 키 입력을 검출하는 벡터를 설정
21              vector.x = Input.GetAxis("Horizontal");
22              vector.z = Input.GetAxis("Vertical");
23
24              // 높이 방향의 판정은 z키로 한다
25              if (Input.GetKey("z"))
26              {
27                  vector.y = 1.0f;
28              }
29              else
30              {
31                  vector.y = -1.0f;
32              }
33
34              // 혹은 이렇게도 정의할 수 있다
35              // vector.y = Input.GetKey("z") ? 1.0f : -1.0f;
36          }
37
38          // 실제 Android 내에서 사용할 중력 센서 키
39          else
40          {
41              vector.x = Input.acceleration.x;
42              vector.z = Input.acceleration.y;
43              vector.y = Input.acceleration.z;
44          }
```

- Android로 Platform 변경 시 기기 내 중력 센서를 vector로 이용할 것이고,
  Unity 내에서 편집할 때 중력 센서를 이용할 수 없으므로
  Update 함수에서 Editor 환경일 때와 아닐 때(if절)을 나누어 작성한다.

II. Lighting 설정(Optional)



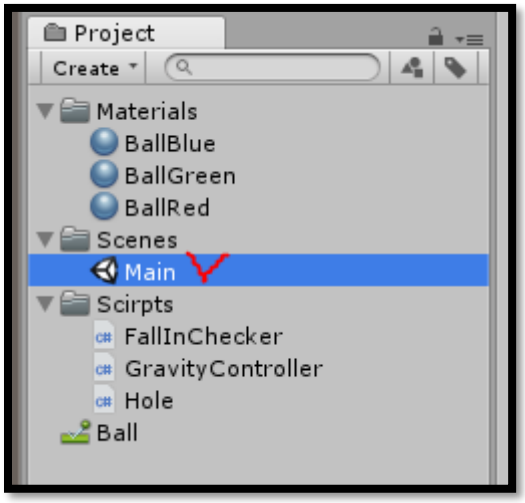
- Lighting 설정 창을 열어 게임 조명을 좀 더 화려하게 바꾸자.
  (Window - Lighting – Settings)



- 체크되어 있는 부분을 위 사진과 같이 동일하게 만들자.

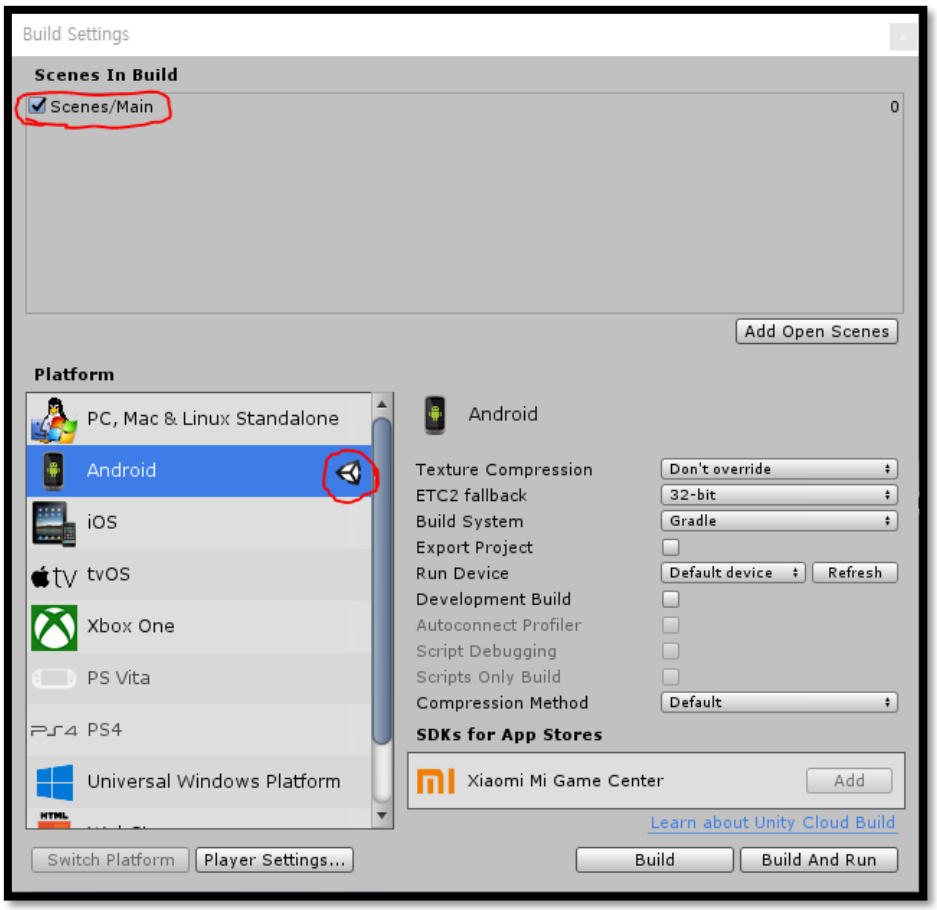
III. Android Platform 설정

i) 먼저 Scene을 Main으로 저장한다.

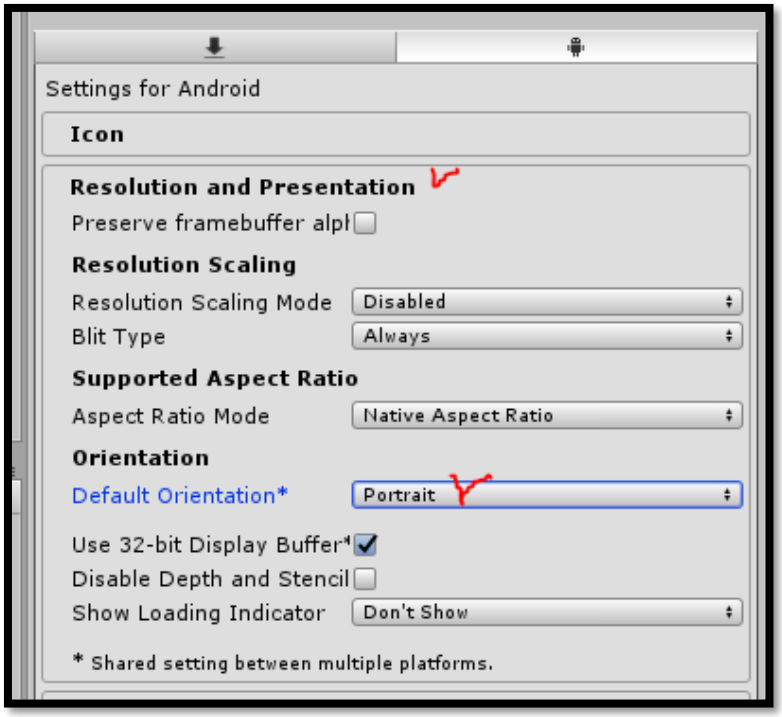


ii) Build Settings에서 Scenes In Build에 Main이 있는지 확인한다.

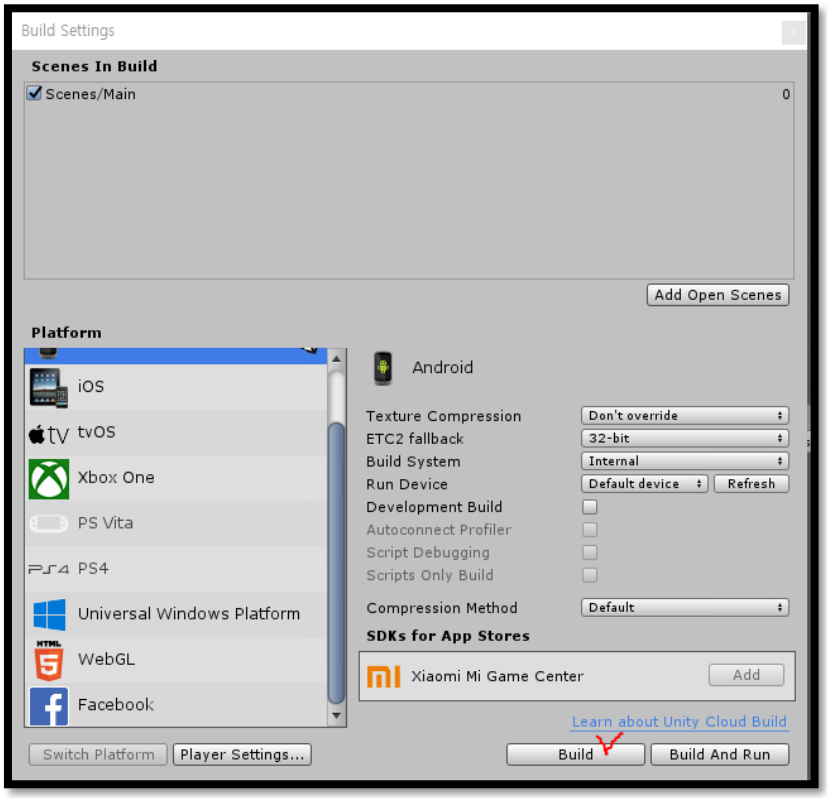
iii) Android로 Switch Platform을 하고 Build System을 Internal로 설정



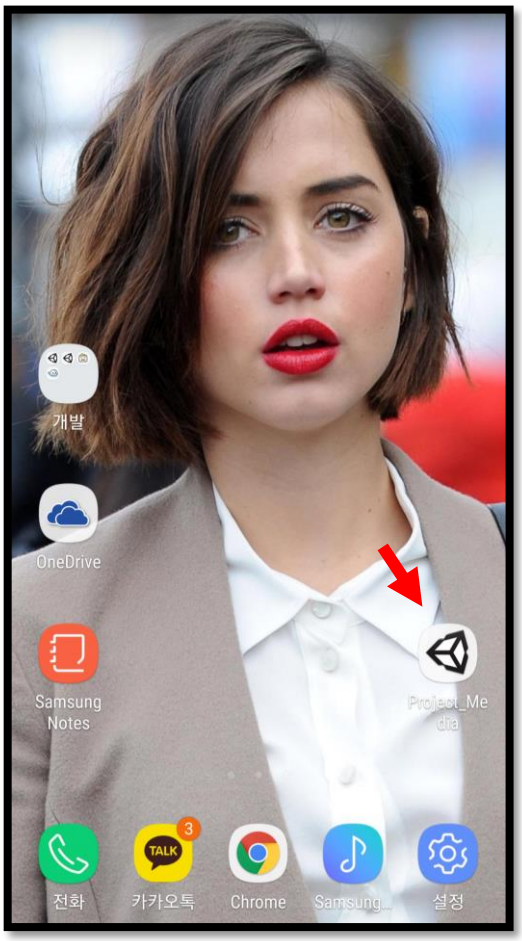
iv) Player Settings - Resolution and Presentation에서
Default Orientation을 Portrait로 설정한다.
(이 게임은 항상 세로로 유지되어야 하기 때문)

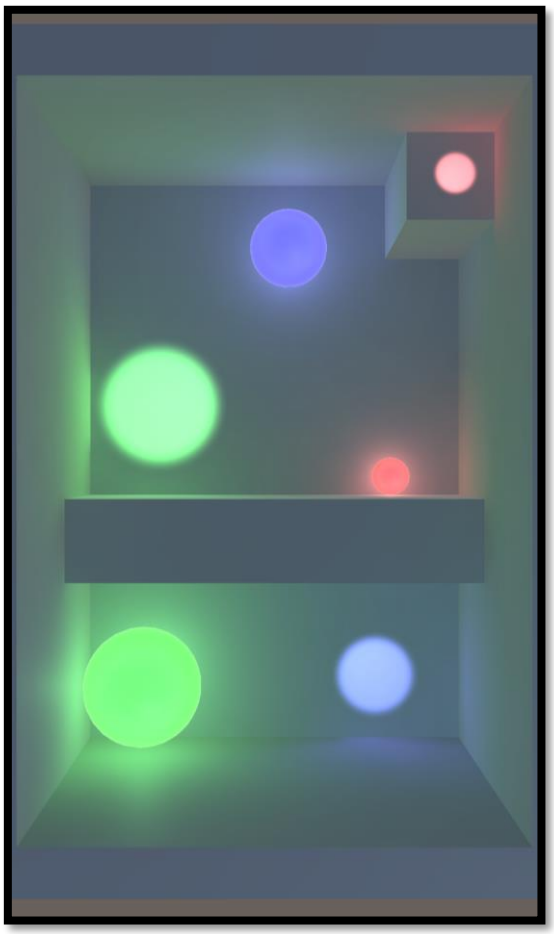


v) Build Settings - Build



vi) 확인





수고하셨습니다.