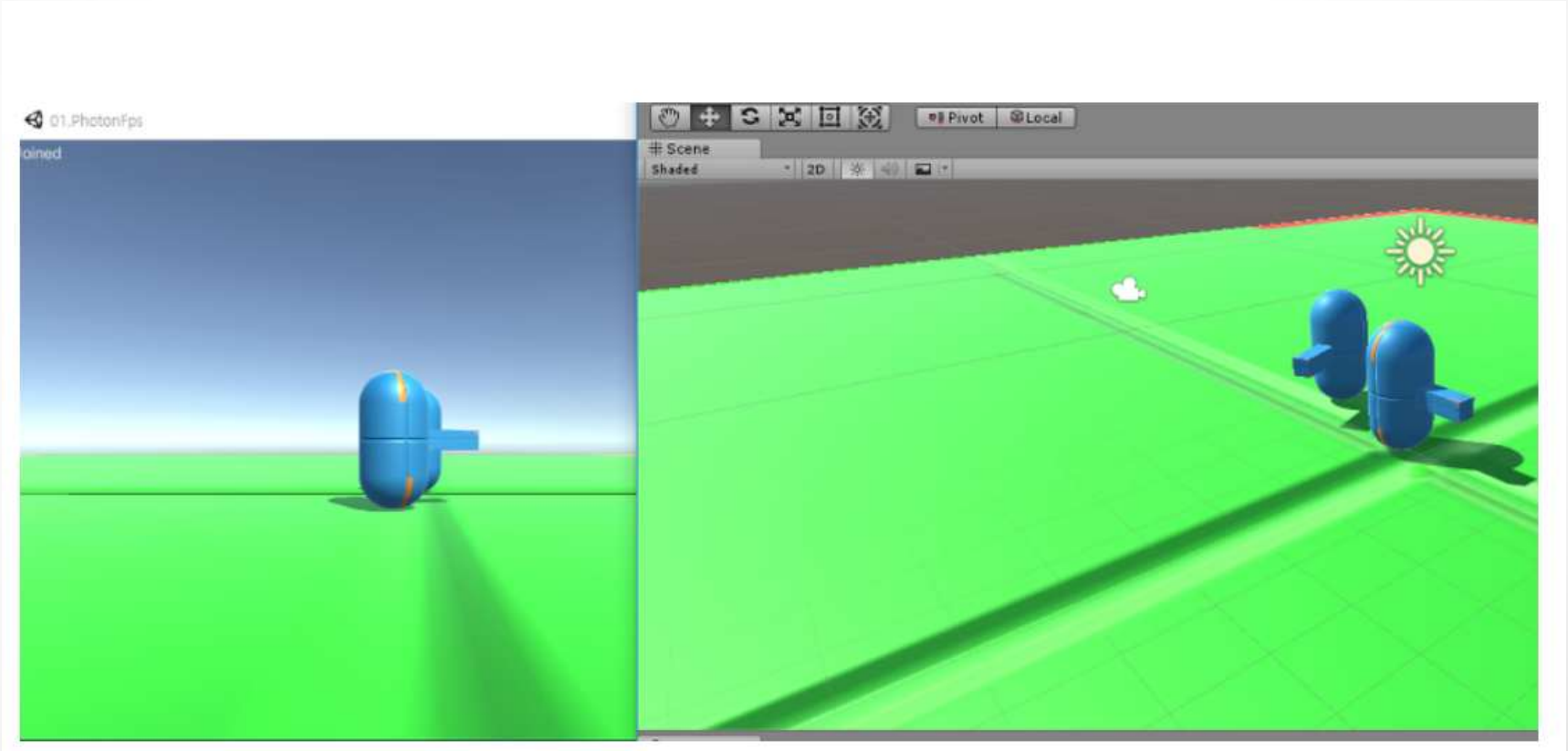
포톤네트워크를 사용한 멀티 플레이 게임 제작

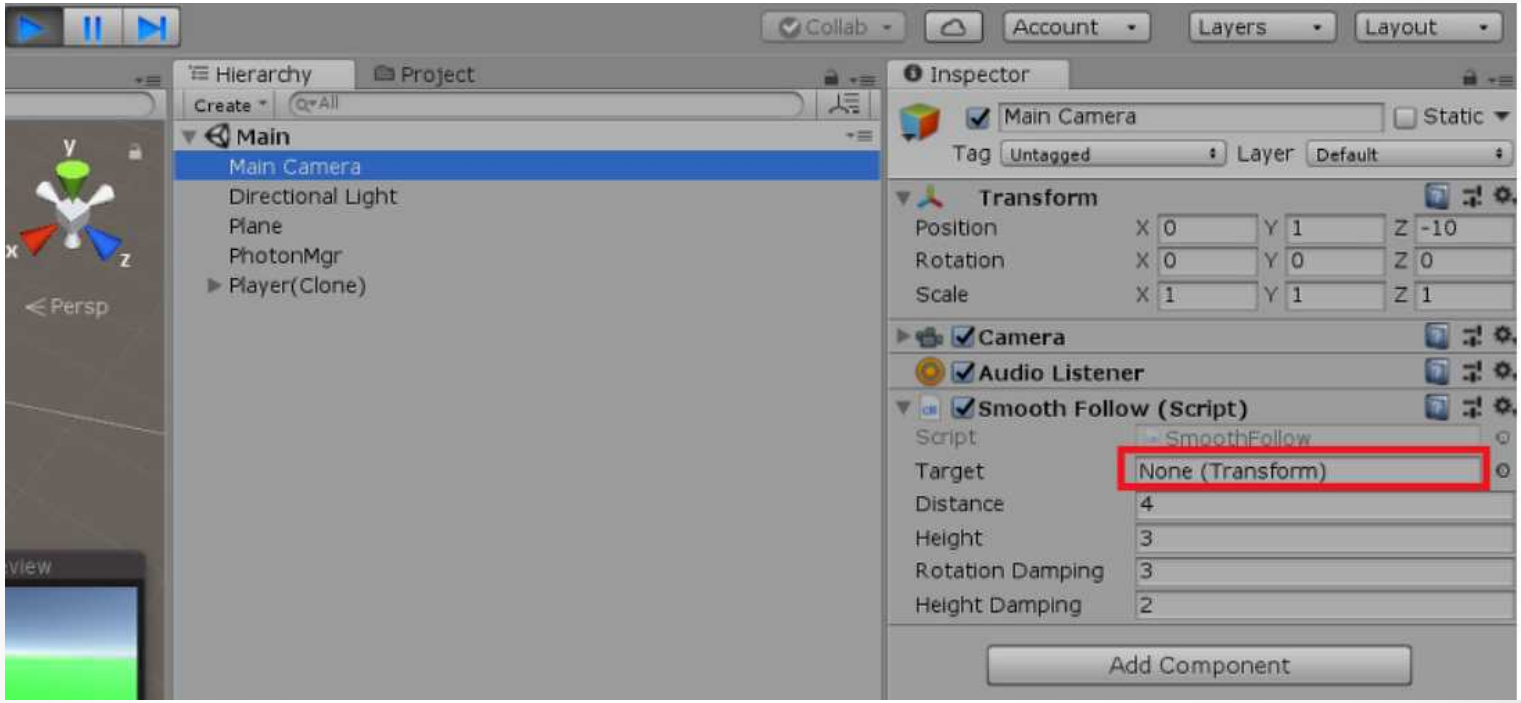
포톤네트워크를 사용한 FPS 개발

- 빌드후 실행 확인



포톤네트워크를 사용한 FPS 개발

- 메인 카메라 설정
  - 게임 실행을 해보니 Target에 오브젝트가 없음
  - SmoothFollow 스크립트에 target이 private인 것이 문제



포톤네트워크를 사용한 FPS 개발

- 메인 카메라 설정
  - 게임 실행을 해보니 Target에 오브젝트가 없음
  - SmoothFollow 스크립트에 target이 private인 것이 문제

```
3   namespace UnityStandardAssets.Utility
4   {
5       public class SmoothFollow : MonoBehaviour
6       {
7
8           // The target we are following
9           [SerializeField]
10          public Transform target;
11          // The distance in the x-z plane to the target
12          [SerializeField]
13          private float distance = 10.0f;
```

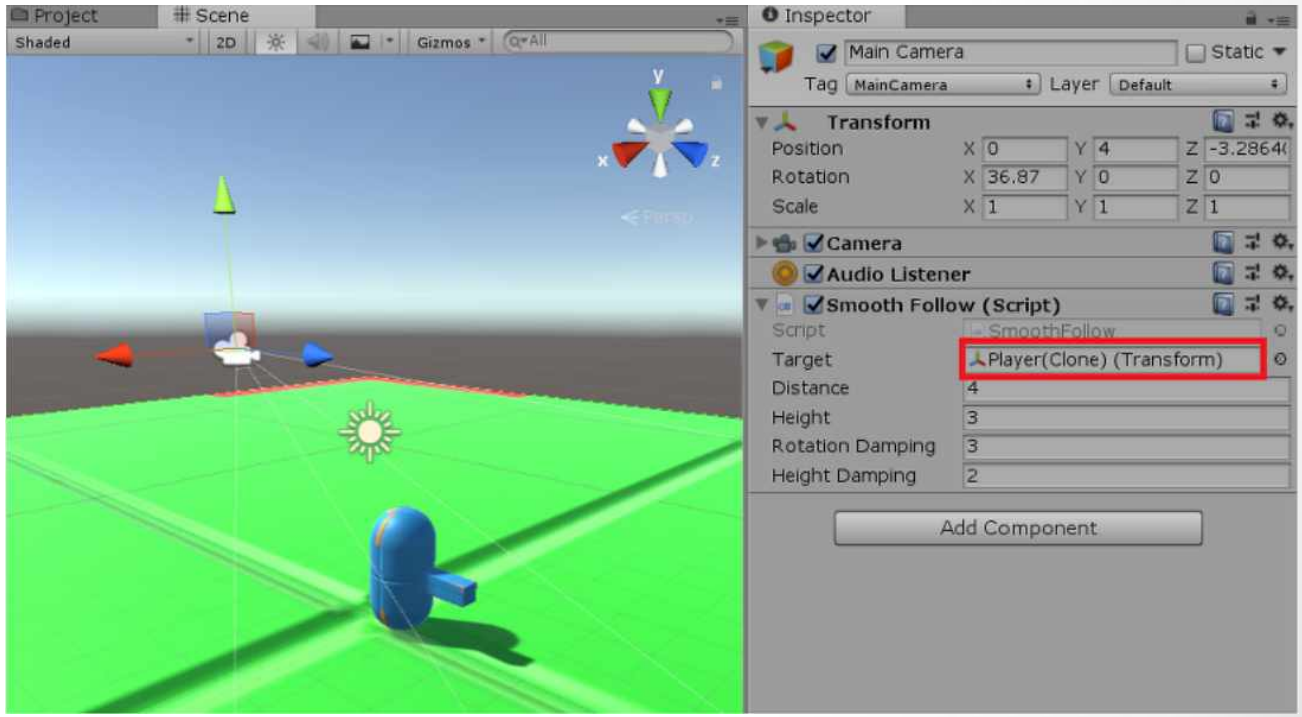
포톤네트워크를 사용한 FPS 개발

- 플레이어 스크립트 수정
  - 네트워크에 들어온 사람이 아닌 오직 나만이 카메라 제어권을가지도록설정하기

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

public class PlayerCtl : MonoBehaviour {

    public float speed = 5.0f;
    public float rotSpeed = 120.0f;

    private Transform tr;
    private PhotonView pv;

    void Start()
    {
        tr = GetComponent<Transform>();
        pv = GetComponent<PhotonView>();

        if (pv.isMine)
        {
            // 자신의 플레이어에게만 카메라 제어권을 연결한다.
            Camera.main.GetComponent<SmoothFollow>().target = tr;
        }
```

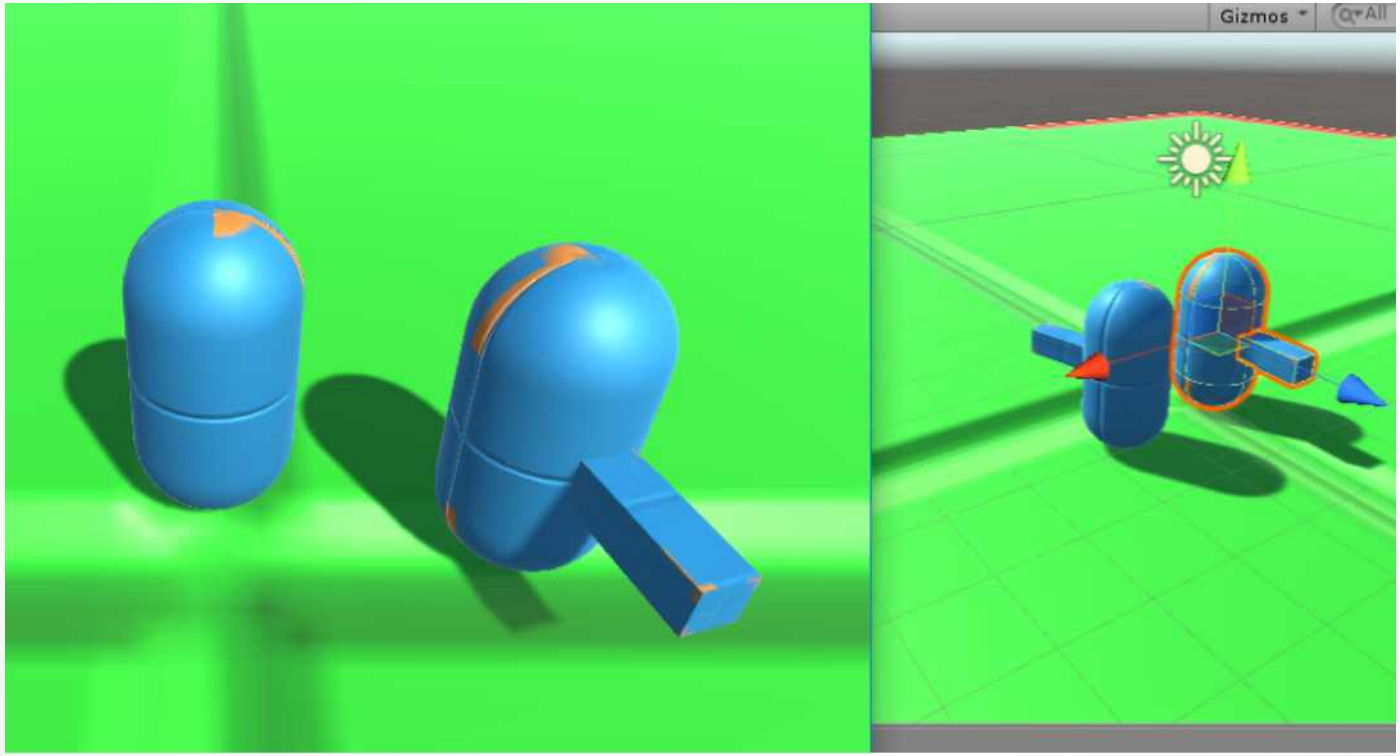
포톤네트워크를 사용한 FPS 개발

- 플레이어 스크립트 수정
  - 네트워크에 들어온 사람이 아닌 오직 나만이 카메라 제어권을가지도록설정하기



포톤네트워크를 사용한 FPS 개발

- 플레이어 캐릭터 구분
  - 머터리얼 속성을 변경하여 구분 하기, isMine을 활용



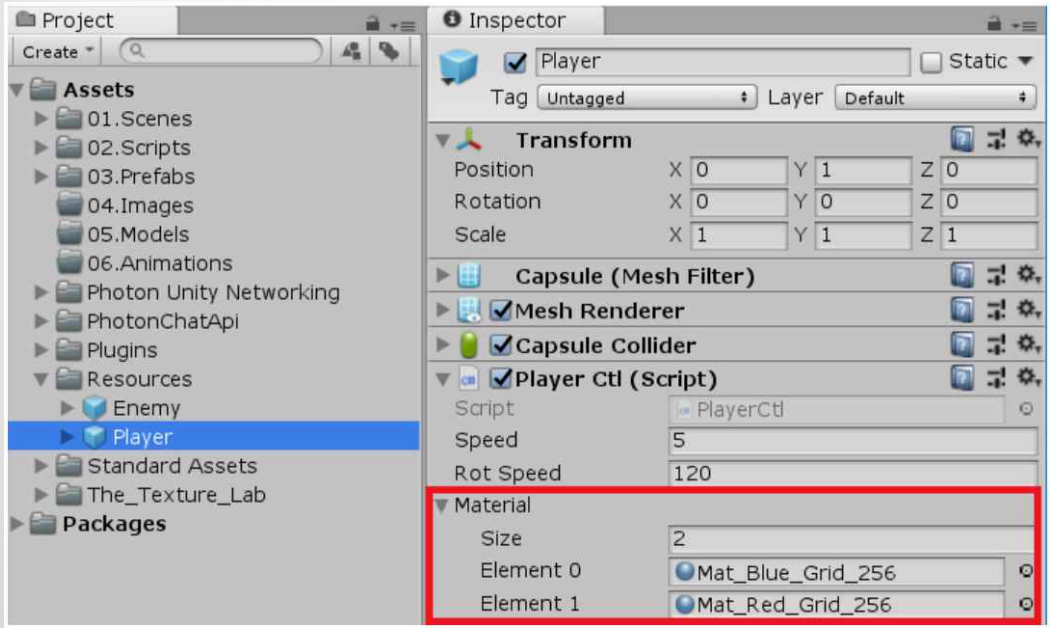
포톤네트워크를 사용한 FPS 개발

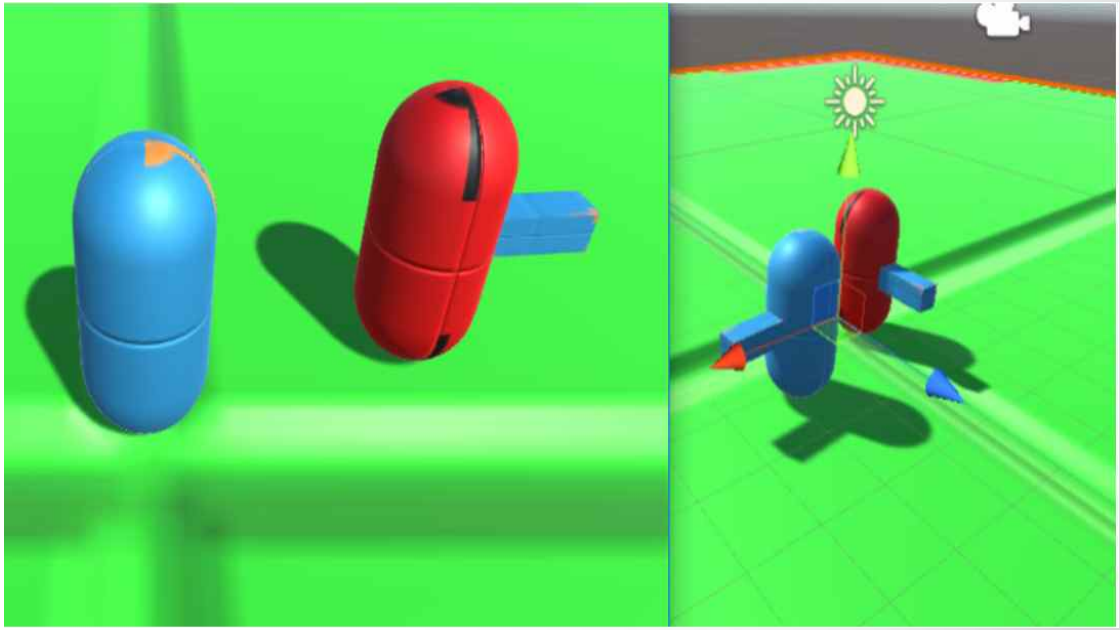
- 플레이어 캐릭터 구분
  - 머터리얼 속성을 변경하여 구분 하기, isMine을 활용

```
14      public Material[] _material;
15
16      void Start()
17      {
18          tr = GetComponent<Transform>();
19          pv = GetComponent<PhotonView>();
20
21          if (pv.isMine)
22          {
23              // 자신의 플레이어에게만 카메라 제어권을 연결한다.
24              Camera.main.GetComponent<SmoothFollow>().target = tr;
25              this.GetComponent<Renderer>().material = _material[0];
26          }
27          else
28          {
29              this.GetComponent<Renderer>().material = _material[1];
30          }
31      }
```

포톤네트워크를 사용한 FPS 개발

- 플레이어 캐릭터 구분
  - 머터리얼 속성을 변경하여 구분 하기, isMine을 활용





포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 적용
  - 움직임 적용하기, 나와 타인을 구분 해야 함!

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityStandardAssets.Utility;
5
6  public class PlayerCtl : MonoBehaviour {
7
8      public float speed = 5.0f;
9      public float rotSpeed = 120.0f;
10
11     private Transform tr;
12     private PhotonView pv;
13
14     public Material[] _material;
15
16     private Vector3 currPos;
17     private Quaternion currRot;
18
```

포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 적용
  - 움직임 적용하기, 나와 타인을 구분 해야 함!

```
19    void Start()
20    {
21        tr = GetComponent<Transform>();
22        pv = GetComponent<PhotonView>();
23
24        // 동기화 콜백함수가 발생하려면 반드시 본 스크립트를 연결 시켜준다.
25        pv.ObservedComponents[0] = this;
26
27        if (pv.isMine)
28        {
29            // 자신의 플레이어에게만 카메라 제어권을 연결한다.
30            Camera.main.GetComponent<SmoothFollow>().target = tr;
31            this.GetComponent<Renderer>().material = _material[0];
32        }
33        else
34        {
35            this.GetComponent<Renderer>().material = _material[1];
36        }
37    }
```

포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 적용
  - 움직임 적용하기, 나와 타인을 구분 해야 함!

```
38    void Update()
39    {
40        if (pv.isMine)
41        {
42            // 자신의 플레이어만 키보드 조작을 허용한다.
43            float h = Input.GetAxis("Horizontal");
44            float v = Input.GetAxis("Vertical");
45
46            tr.Translate(Vector3.forward * v * Time.deltaTime * speed);
47            tr.Rotate(Vector3.up * h * Time.deltaTime * rotSpeed);
48        }
49        else
50        {
51            // 네트워크로 연결된 다른 유저일 경우에는 실시간 전송 받는 변수를 이용해 이동시켜준다.
52            tr.position = Vector3.Lerp(tr.position, currPos, Time.deltaTime * 10.0f);
53            tr.rotation = Quaternion.Lerp(tr.rotation, currRot, Time.deltaTime * 10.0f);
54        }
55    }
56
```

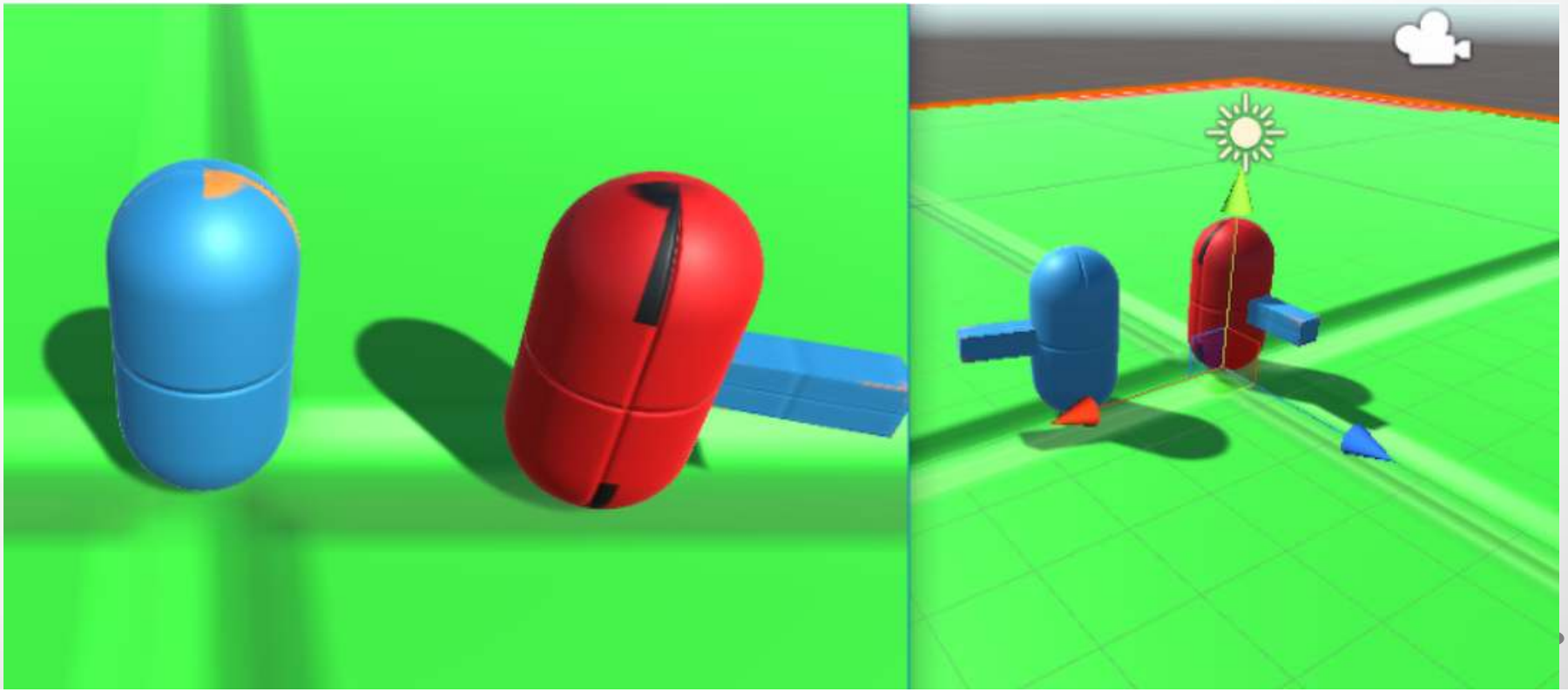
포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 적용
  - 움직임 적용하기, 나와 타인을 구분 해야 함!

```
57      // 동기화 콜백함수
58      void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)
59      {
60          if (stream.isWriting)
61          {
62              // 자신의 플레이 정보는 다른 네트워크 사용자에게 송신한다.
63              stream.SendNext(tr.position);
64              stream.SendNext(tr.rotation);
65          }
66          else
67          {
68              // 타 플레이어의 정보는 수신한다.
69              currPos = (Vector3)stream.ReceiveNext();
70              currRot = (Quaternion)stream.ReceiveNext();
71          }
72      }
73  }
```

포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 적용
  - 움직임 적용하기, 나와 타인을 구분 해야 함!



포톤네트워크를

- 플레이어 컨트롤 수정하기
  - Firepos 설정
  - Bullet 설정

```
public class PlayerCtl : MonoBehaviour {

    public float speed = 5.0f;
    public float rotSpeed = 120.0f;

    private Transform tr;
    private PhotonView pv;

    public Material[] _material;

    private Vector3 currPos;
    private Quaternion currRot;

    public Transform firePos;
    public GameObject bullet;

    void Start()
    {
        tr = GetComponent<Transform>();
        pv = GetComponent<PhotonView>();

        // 동기화 콜백함수가 발생하려면 반드시 본 스크립트를 연결 시켜준다.
        pv.ObservedComponents[0] = this;

        if (pv.isMine)
        {
            // 자신의 플레이어에게만 카메라 제어권을 연결한다.
            Camera.main.GetComponent<SmoothFollow>().target = tr;
            this.GetComponent<Renderer>().material = _material[0];
        }
        else
        {
            this.GetComponent<Renderer>().material = _material[1];
        }
    }
```

포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 수정하기
  - Firepos 설정
  - Bullet 설정

```
void Update()
{
    if (pv.isMine)
    {
        // 자신의 플레이어만 키보드 조작을 허용한다.
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");

        tr.Translate(Vector3.forward * v * Time.deltaTime * speed);
        tr.Rotate(Vector3.up * h * Time.deltaTime * rotSpeed);

        if (Input.GetButtonDown("Fire1"))
        {
            Fire();
        }
    }
    else
    {
        // 네트워크로 연결된 다른 유저일 경우에는 실시간 전송 받는 변수를 이용해 이동시켜준다
        tr.position = Vector3.Lerp(tr.position, currPos, Time.deltaTime * 10.0f);
        tr.rotation = Quaternion.Lerp(tr.rotation, currRot, Time.deltaTime * 10.0f);
    }
}
```

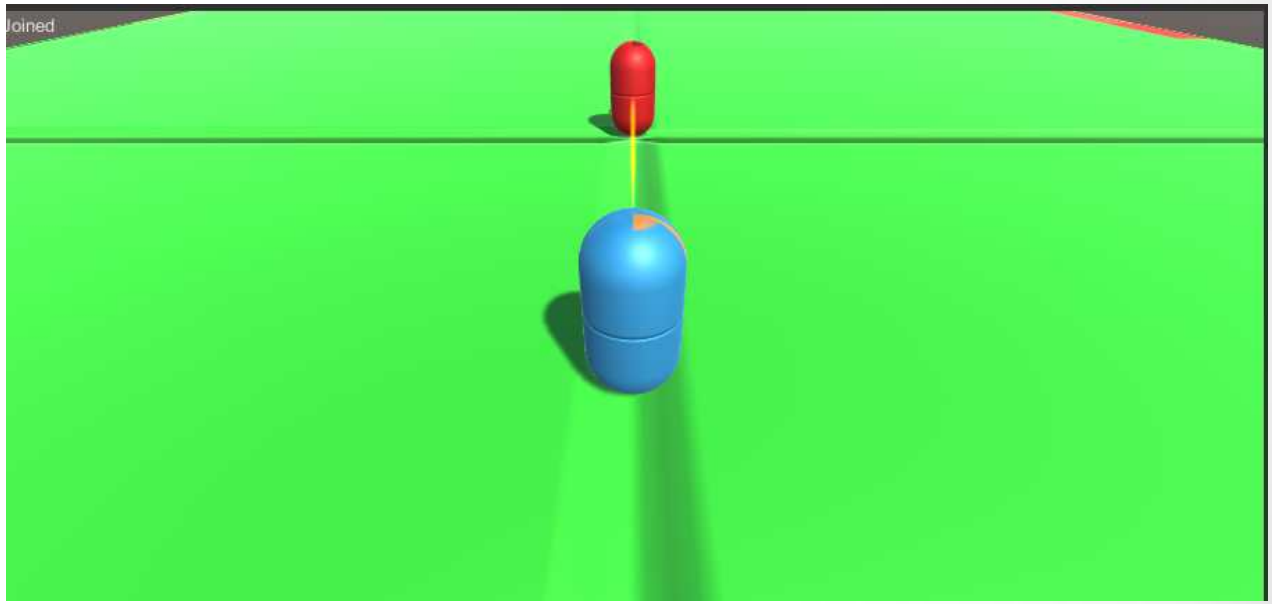
포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 수정하기
  - Firepos 설정
  - Bullet 설정

```
void Fire()
{
    StartCoroutine(this.CreateBullet());
    pv.RPC("FireRPC", PhotonTargets.Others);
}

IEnumerator CreateBullet()
{
    Instantiate(bullet, firePos.position, firePos.rotation);
    yield return null;
}

[PunRPC]
void FireRPC()
{
    StartCoroutine(this.CreateBullet());
}
```

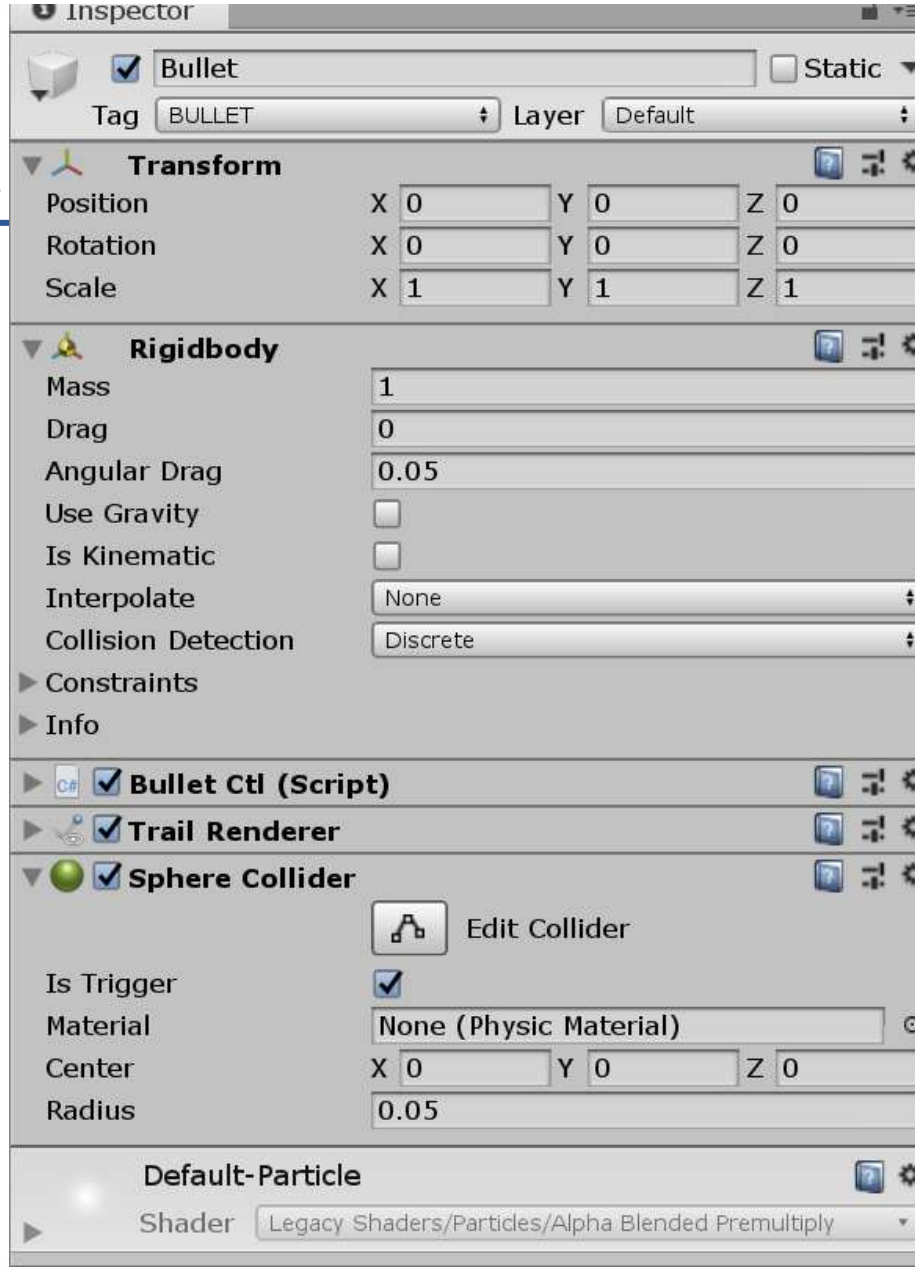
포톤네트워크를 사용한 FPS 개발

- 총알 발사 테스트하기



포톤네트워크를 사용

- 총알 수정하기!
  - BULLET 객체에 다음과 같이 Sphere Collider 추가 (IsTrigger 활성화)
  - TAG 정보에서 BULLET 으로 이름 추가하기



포톤네트워크를 사용한 FPS 개발

- Rigidbody

함수명	설명
Mass	질량
Drag	이동할 때의 저항
Angular Drag	회전할 때의 저항
Use Gravity	중력 사용 여부
Is Kinematic	체크되면 물리를 이용한 움직임을 않하고 Transform으로 이동
Interpolate	물리력을 이용한 움직임이 끊길 때 보간
Collision Detection	아주 빠른 물리 운동에 따른 충돌 검사
Freeze Position	해당 축으로의 이동을 막음
Freeze Rotation	해당 축으로의 회전을 막음

포톤네트워크를 사용한 FPS 개발

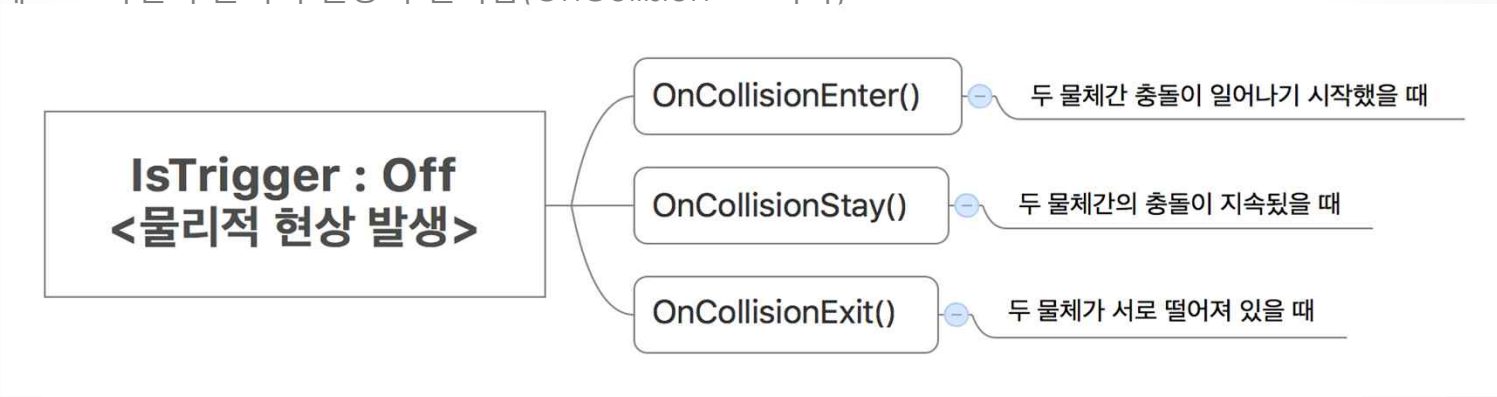
- Collider컴포넌트는 충돌을 감지하는 일종의 센서
  - Box Collider
    - 가장 일반적인 충돌 체크 (건물등에 사용)
  - Sphere Collider
    - 가장 처리속도가 빠름
  - Capsule Collider
    - 주로 Player나 적 캐릭터의 충돌체로 사용할 때가 많음
  - Mesh Collider
    - 충돌 감지를 위한 CPU부하가 제일 많은 Collider
  - Wheel Collider
    - 차량의 바퀴에 사용할 목적으로 제공되는 Collider
  - Terrain Collider
    - 지형의 충돌을 위한 Collider

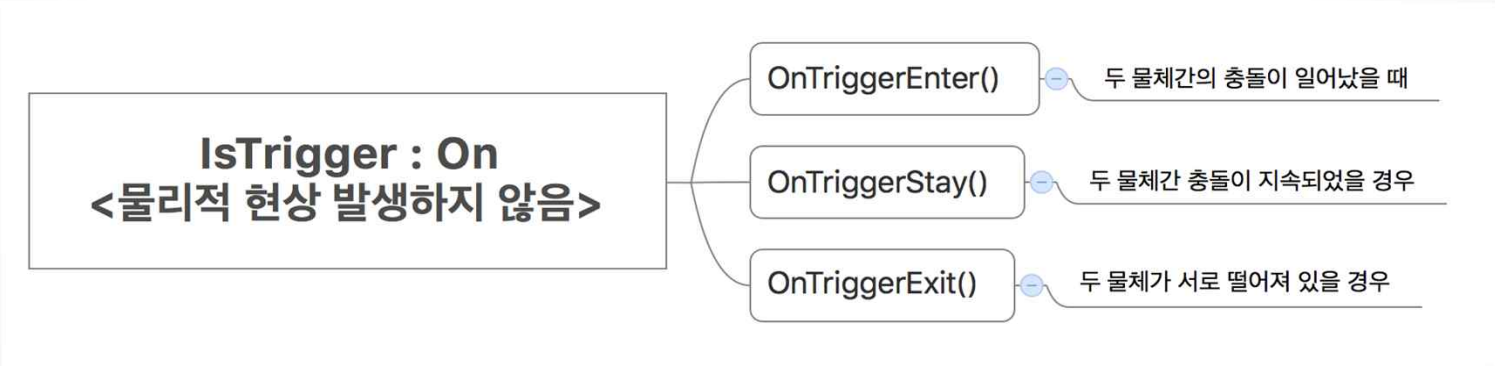
포톤네트워크를 사용한 FPS 개발

- 충돌 검사 조건
  - 충돌을 일으키는 양쪽 모두 Collider 컴포넌트가 있어야 함
  - 두 게임 오브젝트중 움직이는 쪽에는 반드시 Rigidbody컴포넌트가 있어야 함
  - Component > Physics > Sphere Collider 추가 (Radius : 0.6)

포톤네트워크를 사용한 FPS 개발

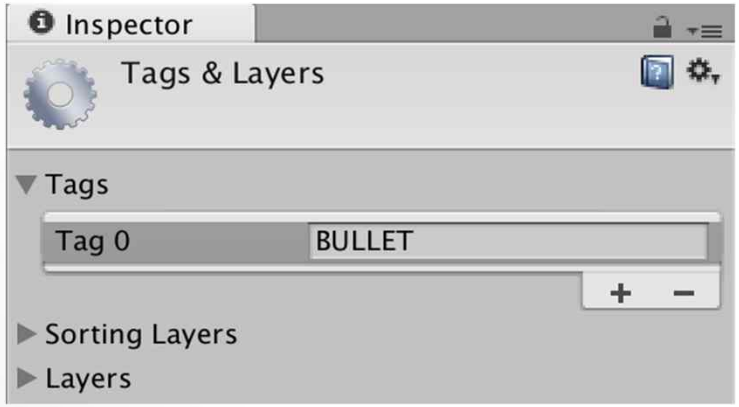
- 모든 Collider컴포넌트에는 IsTrigger 속성이 있음
  - On : 충돌 체크는 하지만 물리적 현상은 일어나지 않음 (OnTrigger~로 시작)
  - Off : 충돌 체크도 하면서 물리적 현상이 일어남(OnCollision~로 시작)

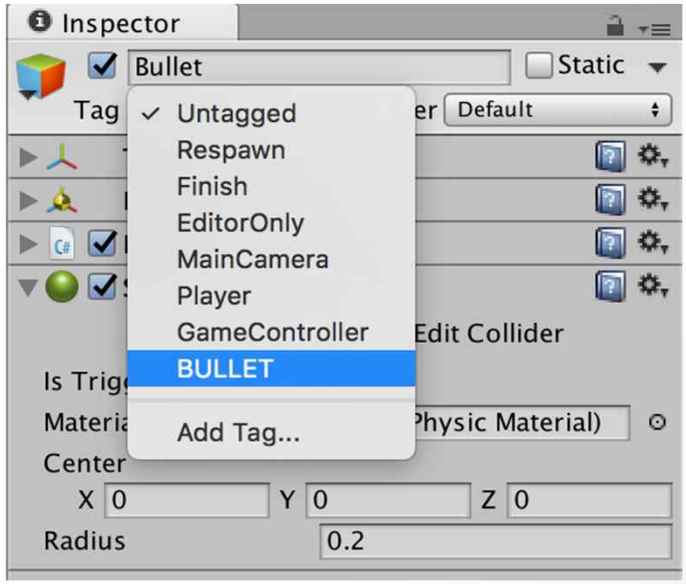




포톤네트워크를 사용한 FPS 개발

- 벽에 총알이 날라 올 경우 어떤 Collider컴포넌트를 가진 오브젝트가 날라오는지를 판별해야 함
  - 판별의 식별자로 태그를 활용할 수 있음 (C#의 Hash Table 기능)
  - Edit > Project Setting < Tags & Layers





포톤네트워크를 사용한 FPS 개발

- 플레이어 컨트롤 수정하기
  - 플레이어의 총히트, 죽음, 리스폰 처리하기

```
public class PlayerCtl : MonoBehaviour {

    public float speed = 5.0f;
    public float rotSpeed = 120.0f;

    private Transform tr;
    private PhotonView pv;

    public Material[] _material;

    private Vector3 currPos;
    private Quaternion currRot;

    public Transform firePos;
    public GameObject bullet;

    private bool isDie = false;
    private int hp = 100;
    private float respawnTime = 3.0f;
```

포톤네트워크를 사용한 FPS 개발

- 총알의 충돌 체크
  - 생명력이 0이하면 사망처리

```
//총알의 충돌체크
void OnTriggerEnter(Collider coll)
{
    Debug.Log("HIT!:"+ coll.gameObject.tag);
    if (coll.gameObject.tag == "BULLET")
    {
        Debug.Log("HIT!");
        Destroy(coll.gameObject);
        hp -= 10;

        if(hp <= 0)
        {
            StartCoroutine(this.RespawnPlayer(respawnTime));
        }
    }
}
```

포톤네트워크를 사용한 FPS 개발

- 사망처리
  - 죽으면 일정 시간 동안 투명화 되었다가 다시 처리되도록 하기

```
// 사망처리 및 Respawn 처리
IEnumerator RespawnPlayer(float waitTime)
{
    Debug.Log("Died!");
    isDie = true;
    StartCoroutine(this.PlayerVisible(false, 0.0f));
    yield return new WaitForSeconds(waitTime);

   // 리스폰 후 플레이어의 위치를 무작위로 선출
    tr.position = new Vector3(Random.Range(-20.0f, 20.0f), 1.0f, Random.Range(-20.0f, 20.0f));

    //생명치를 초깃값으로 재설정
    hp = 100;
    isDie = false;
    StartCoroutine(this.PlayerVisible(true, 0.5f));
}
```

포톤네트워크를 사용한 FPS 개발

- 리스폰시 활성/비활성화 하기
  - 죽었을때 플레이어를 안보이게 하는 방법으로 해결

```
// 플레이어의 보이는 것의 활성/활성처리
IEnumerator PlayerVisible(bool visibled, float dealyTime)
{
    yield return new WaitForSeconds(dealyTime);
    GetComponentInChildren<MeshRenderer>().enabled = visibled;
    transform.Find("Gun").gameObject.GetComponentInChildren<MeshRenderer>().enabled = visibled


}
```

포톤네트워크를 사용한 FPS 개발

- 팀과 1:1 전투를 해보기!
  - APP_ID를 공개하여 동기화 하면 같은 방에 접속할수 있음
  - 3c4c4fc9-1aae-4d23-8b39-c3bb4b903437