



7주차

멀티플레이 이식 (2)

강의 영상

<https://youtu.be/dp0UUu4AuQg>

코드

▼ Weapon.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Unity.Burst.CompilerServices;

public class Weapon : MonoBehaviour
```

```

{
    public int rpm = 700;
    private float fireInterval;
    private float fireTimer = 0f;

    public ParticleSystem muzzleFlash;
    public AudioClip fireSound;
    private AudioSource audioSource;

    public LayerMask layerMask;
    public GameObject bulletHolePrefab;

    public float defaultAccuracy = 0.2f;
    private float currentAccuracy;
    public float recoil = 0.1f;

    private Hud hud;

    public int ammoLeft = 30;
    public int maxAmmo = 30;
    private Animator animator;
    private bool isReloading = false;
    public Animator tpsAnimator;

    public ParticleSystem tpsMuzzleFlash;

    private PhotonView pv;
    private PlayerControl playerControl;

    private void Awake()
    {
        playerControl = GetComponentInParent<PlayerControl>();
        pv = GetComponent<PhotonView>();

        currentAccuracy = defaultAccuracy;
        fireInterval = 60f / rpm;
        audioSource = GetComponent<AudioSource>();
        hud = FindObjectOfType<Hud>();
        animator = GetComponent<Animator>();
    }

    private void Start()
    {
        if (!pv.IsMine)
        {
            //this.gameObject.SetActive(false);
            Renderer[] renderers = GetComponentsInChildren<Renderer>();
            foreach(Renderer r in renderers)
            {
                r.enabled = false;
            }
        }
    }

    private void Update()

```

```

{
    if (!pv.IsMine)
    {
        return;
    }

    if (playerControl.healthControl.isDead)
    {
        return;
    }

    fireTimer += Time.deltaTime;
    if (fireTimer >= fireInterval)
    {
        if (Input.GetKey(KeyCode.Mouse0) && !isReloading)
        {
            // 총알 발사 처리 로직
            fireTimer = 0f;
            currentAccuracy += recoil;
            ammoLeft--;

            RaycastTarget();
            FireEffect();
        }
    }

    currentAccuracy = Mathf.Lerp(currentAccuracy, defaultAccuracy,
        Time.deltaTime * 10f);

    hud.UpdateCrosshairs(currentAccuracy + 0.05f);
    hud.UpdateAmmoText(ammoLeft, maxAmmo);

    if (ammoLeft <= 0 || (Input.GetKeyDown(KeyCode.R) && ammoLeft < maxAmmo))
    {
        isReloading = true;
        animator.SetBool("isReloading", true);
    }

    playerControl.isReloading = isReloading;
}

private void FireEffect()
{
    muzzleFlash.Play();
    audioSource.PlayOneShot(fireSound);

    pv.RPC(nameof(RpcFireEffect), RpcTarget.Others);
}

private void RaycastTarget()
{
    Vector2 circle = Random.insideUnitCircle * currentAccuracy;
    Vector3 direction = Camera.main.transform.forward
        + Camera.main.transform.up * circle.y
        + Camera.main.transform.right * circle.x;

```

```

    Ray ray = new Ray(Camera.main.transform.position, direction);

    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, Mathf.Infinity, layerMask.value))
    {
        HealthControl hc = hit.collider.GetComponentInParent<HealthControl>();
        if (hc != null)
        {
            hc.OnHit(pv.OwnerActorNr, hit.point, ray.direction);
        }
        else
        {
            GameObject bh = Instantiate(bulletHolePrefab, hit.point, Quaternion.identity);
            Destroy(bh, 3f);

            pv.RPC(nameof(RpcOnWorldHit), RpcTarget.Others, hit.point);
        }
    }
    else
    {
    }
}

[PunRPC]
public void RpcFireEffect()
{
    tpsMuzzleFlash.Play();
    audioSource.PlayOneShot(fireSound);
}

[PunRPC]
public void RpcOnWorldHit(Vector3 hitpoint)
{
    GameObject bh = Instantiate(bulletHolePrefab, hitpoint, Quaternion.identity);
    Destroy(bh, 3f);
}

public void AnimationEvent(string eventName)
{
    if (eventName == "Weapon_Reload_Complete")
    {
        isReloading = false;
        ammoLeft = maxAmmo;
        animator.SetBool("isReloading", false);
    }
}
}

```

▼ HealthControl.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

public class HealthControl : MonoBehaviour, IPunObservable
{
    public float health = 100f;
    public bool isDead = false;

    public GameObject bloodPrefab;
    private Rigidbody[] rigidbodies;
    private Hud hud;

    public Animator tpsAnimator;
    private PhotonView pv;
    private CapsuleCollider capsuleCollider;

    private GameManager gm;

    private void Awake()
    {
        gm = FindObjectOfType<GameManager>();

        hud = FindObjectOfType<Hud>();
        pv = GetComponent<PhotonView>();

        capsuleCollider = GetComponent<CapsuleCollider>();
        rigidbodies = GetComponentsInChildren<Rigidbody>();
        foreach(Rigidbody rb in rigidbodies)
        {
            rb.isKinematic = true;
        }
    }

    private void Start()
    {
        if (!pv.IsMine)
        {
            if (isDead)
            {
                tpsAnimator.enabled = false;

                foreach (Rigidbody rb in rigidbodies)
                {
                    rb.isKinematic = false;
                }
            }
            else
            {
                capsuleCollider.enabled = true;
            }
        }
        else
    }

```

```

        {
            capsuleCollider.enabled = false;
        }
    }

    private void Update()
    {
        if (pv.IsMine)
        {
            hud.UpdateHealthText(health);
        }
    }

    public void OnHit(int viewId, Vector3 hitPoint, Vector3 inDir)
    {
        pv.RPC(nameof(RpcOnHit),
            RpcTarget.Others, viewId, hitPoint, inDir);

        GameObject blood = Instantiate(bloodPrefab, hitPoint, Quaternion.identity);
        Destroy(blood, 5f);
    }

    [PunRPC]
    private void RpcOnHit(int viewId, Vector3 hitPoint, Vector3 inDir)
    {
        if (pv.IsMine)
        {
            if (!isDead)
            {
                health -= 20f;
                if (health <= 0f)
                {
                    health = 0f;

                    isDead = true;
                    OnDead(viewId, inDir);
                }

                hud.UpdateBloodScreen();
            }
        }

        GameObject blood = Instantiate(bloodPrefab, hitPoint, Quaternion.identity);
        Destroy(blood, 5f);
    }

    public void OnDead(int viewId, Vector3 inDir)
    {
        tpsAnimator.enabled = false;

        foreach (Rigidbody rb in rigidbodies)
        {
            rb.isKinematic = false;
        }
    }

```

```

        rigidbodies[0].AddForce(inDir * 300f, ForceMode.Impulse);

        hud.UpdateDeadScreen(true);

        Invoke(nameof(Respawn), 5f);

        pv.RPC(nameof(RpcOnDead), RpcTarget.Others, viewId, inDir);
    }

    [PunRPC]
    private void RpcOnDead(int viewId, Vector3 inDir)
    {
        if (viewId ==
            PhotonNetwork.LocalPlayer.ActorNumber)
        {
            hud.UpdateKillText();
        }

        capsuleCollider.enabled = false;
        tpsAnimator.enabled = false;

        foreach (Rigidbody rb in rigidbodies)
        {
            rb.isKinematic = false;
        }

        rigidbodies[0].AddForce(inDir * 300f, ForceMode.Impulse);
    }

    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            stream.SendNext(isDead);
        }
        else if (stream.IsReading)
        {
            isDead = (bool)stream.ReceiveNext();
        }
    }

    private void Respawn()
    {
        health = 100f;
        isDead = false;

        (Vector3 pos, Quaternion rot) =
            gm.GetRespawnPoint();

        this.transform.SetPositionAndRotation(pos, rot);
        hud.UpdateDeadScreen(false);

        pv.RPC(nameof(RpcOnRespawn), RpcTarget.Others,
            pos, rot);
    }

```

```

[PunRPC]
private void RpcOnRespawn(Vector3 pos, Quaternion rot)
{
    tpsAnimator.enabled = true;
    capsuleCollider.enabled = true;
    foreach (Rigidbody rb in rigidbodies)
    {
        rb.isKinematic = true;
    }

    this.transform.SetPositionAndRotation(pos, rot);
}
}

```

▼ Hud.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class Hud : MonoBehaviour
{
    public Transform[] crosshairs;
    public TextMeshProUGUI ammoLeftText;
    public Image bloodScreen;
    public GameObject deadScreen;
    public TextMeshProUGUI healthText;
    public GameObject killText;

    public void UpdateKillText()
    {
        killText.SetActive(true);

        CancelInvoke(nameof(DisableKillText));
        Invoke(nameof(DisableKillText), 2f);
    }

    private void DisableKillText()
    {
        killText.SetActive(false);
    }

    public void UpdateHealthText(float health)
    {
        healthText.text = "HP " + (int)health;
    }

    public void UpdateDeadScreen(bool active)
    {

```



```

        deadScreen.SetActive(active);
    }

    public void UpdateBloodScreen()
    {
        StopCoroutine(nameof(BloodScreenRoutine));
        StartCoroutine(nameof(BloodScreenRoutine));
    }

    private IEnumerator BloodScreenRoutine()
    {
        float alpha = 0.3f;
        while(alpha >= 0f)
        {
            Color color = bloodScreen.color;
            color.a = alpha;
            bloodScreen.color = color;

            alpha -= Time.deltaTime;
            yield return null;
        }
    }

    public void UpdateAmmoText(int ammoLeft, int maxAmmo)
    {
        ammoLeftText.text = ammoLeft + "/" + maxAmmo;
    }

    public void UpdateCrosshairs(float dist)
    {
        Vector3 upPosition = Camera.main.transform.position +
            Camera.main.transform.forward * dist;
        Vector3 downPosition = Camera.main.transform.position +
            Camera.main.transform.forward - Camera.main.transform.up * dist;
        Vector3 rightPosition = Camera.main.transform.position +
            Camera.main.transform.forward + Camera.main.transform.right * dist;
        Vector3 leftPosition = Camera.main.transform.position +
            Camera.main.transform.forward - Camera.main.transform.right * dist;

        crosshairs[0].position = Camera.main.WorldToScreenPoint(upPosition);
        crosshairs[1].position = Camera.main.WorldToScreenPoint(downPosition);
        crosshairs[2].position = Camera.main.WorldToScreenPoint(rightPosition);
        crosshairs[3].position = Camera.main.WorldToScreenPoint(leftPosition);
    }
}

```

▼ GameManager.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

```

```

public class GameManager : MonoBehaviour
{
    public List<Transform> respawnPoints;

    private void Start()
    {
        (Vector3 pos, Quaternion rot) = GetRespawnPoint();
        PhotonNetwork.Instantiate("Player", pos, rot);
    }

    public (Vector3, Quaternion) GetRespawnPoint()
    {
        // Random.Range(a, b);
        // a ~ (b - 1);
        int index = Random.Range(0, respawnPoints.Count);
        return
            (respawnPoints[index].position,
             respawnPoints[index].rotation
            );
    }
}

```