2. VR 개발팀 진행 사항

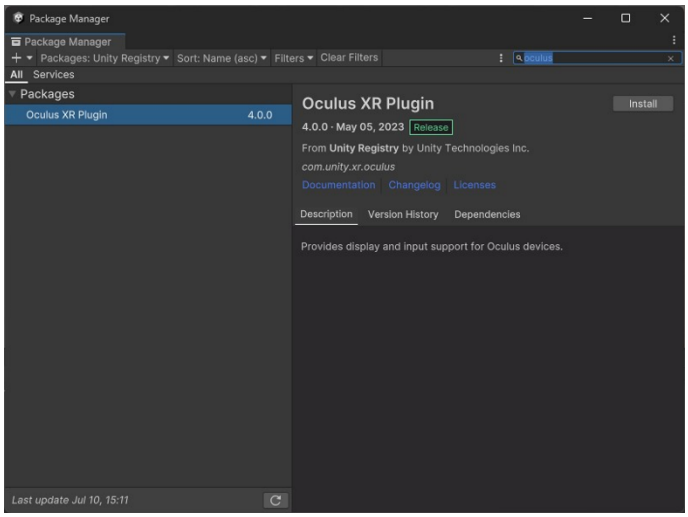
Unity 2023.1.3버전 사용 + 오큘러스 2, 오큘러스 Rift

사용한 Custom 에셋

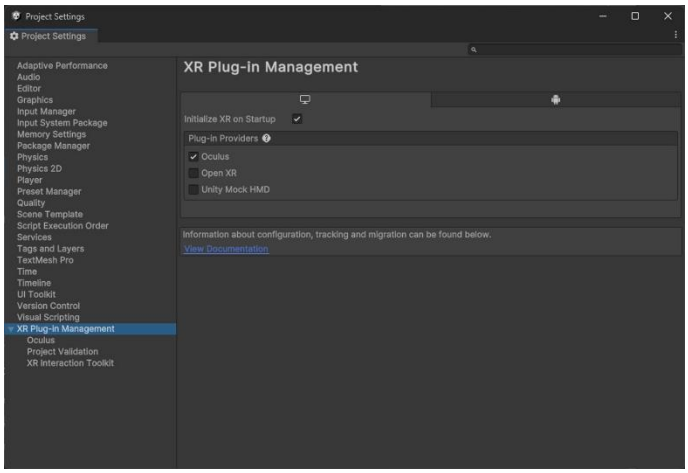
- Pdollar Point-Cloud GestureRecognizer
- SteamVR Plugin
- Oculus Integration
- VR Tunnelling Pro
- VRArmIK
- VR Screen Shake
- VR TK

미니 프로젝트 진행 <Pistol Whip>

1. 초기 세팅
   - Unity에 Oculus 세팅하기
     - 유니티 asset store에서 oculus integration 설치
       - https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022
     - 유니티 프로젝트 생성
     - [Window] - [Package Manager] - 다음 내용들 import
       - Oculus XR Plugin
       - XR Interaction Toolkit - Samples - Starter Assets
       - XR Plugint Management



■ [File] - [Build Settings] - [Player Settings] - XR Plug-in Management - Oculus 체크

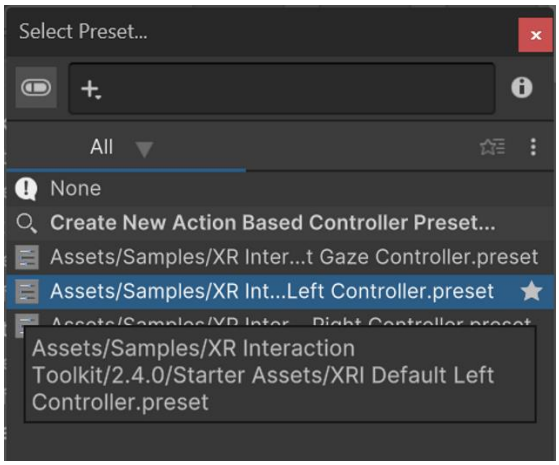


■ Oculus 앱 - [일반] - [기기] - 알 수 없는 출처 허용 - OpenXR Runtime 활성화

- XR Origin을 이용한 기본 카메라 및 컨트롤러 세팅

■ 컨트롤러 세팅

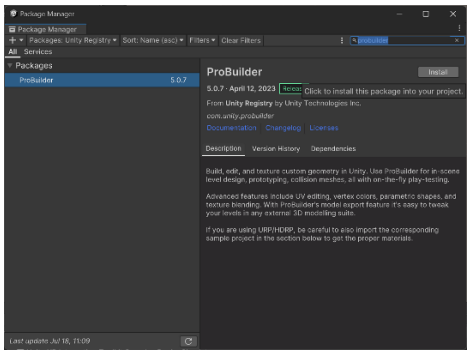
◆ 왼손 오른손에 따라 import한 asset 적용

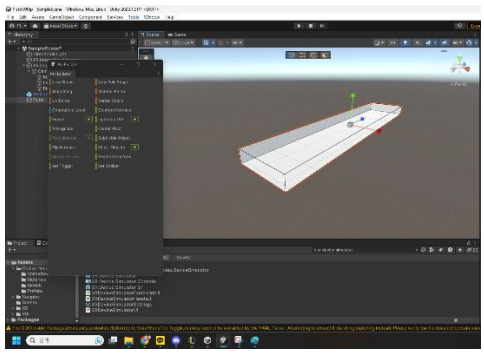


2. 사용한 assets 및 tools

- Unity Probuilder를 이용한 단순 모델링

  ■ ProBuilder를 사용하여 구조, 복잡한 터레인 요소, 차량, 무기의 프로토타입을 빠르게 제작하거나 커스텀 콜리전 지오메트리, 트리거 구역 또는 내비 메시(nav mesh)를 만듦

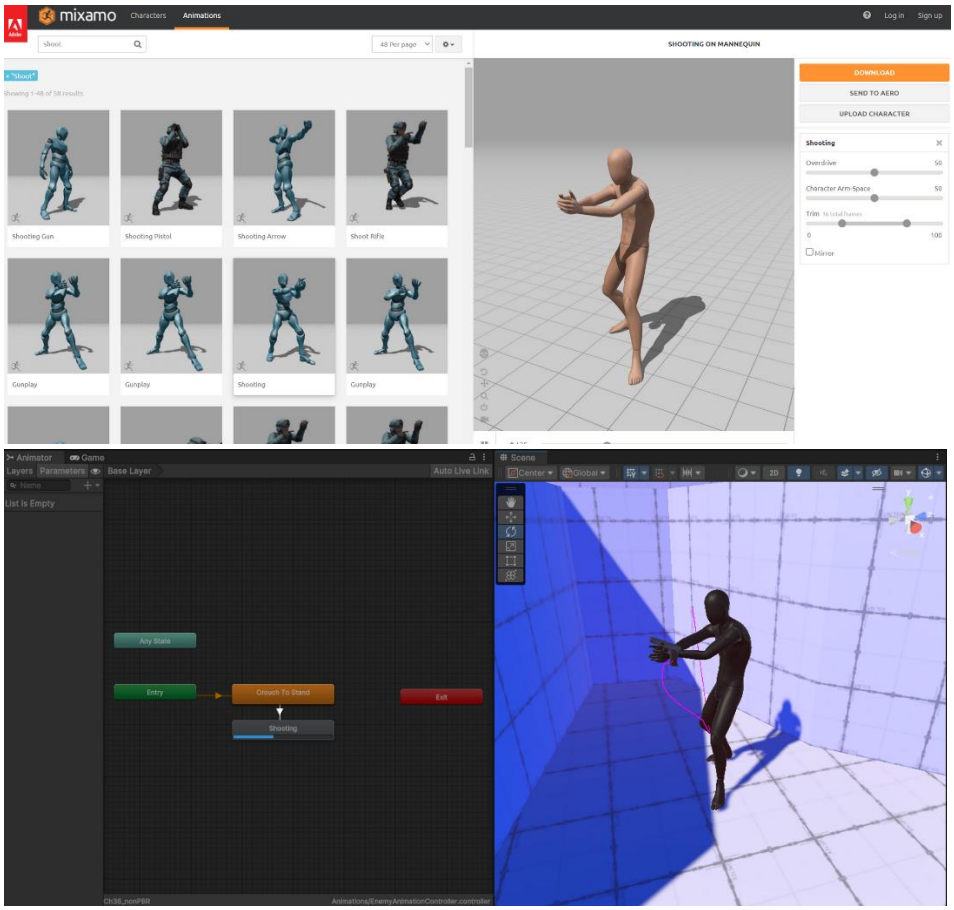




- Mixamo 및 애니메이션

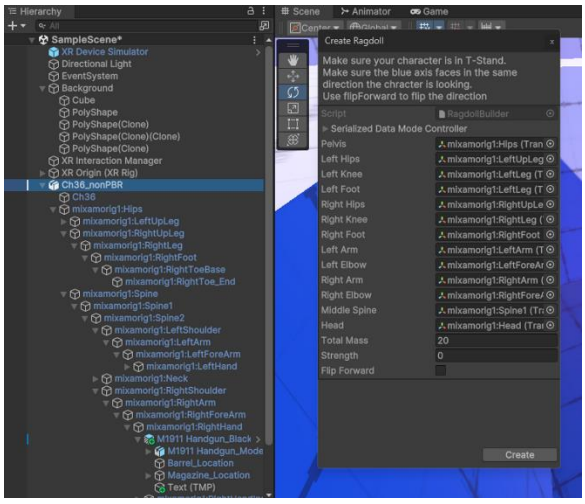
  ■ https://www.mixamo.com/#/

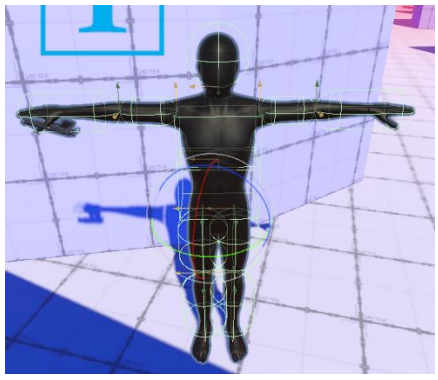
  ■ 3d 캐릭터 모델 및 애니메이션 asset 무료 제공 사이트



- Ragdoll

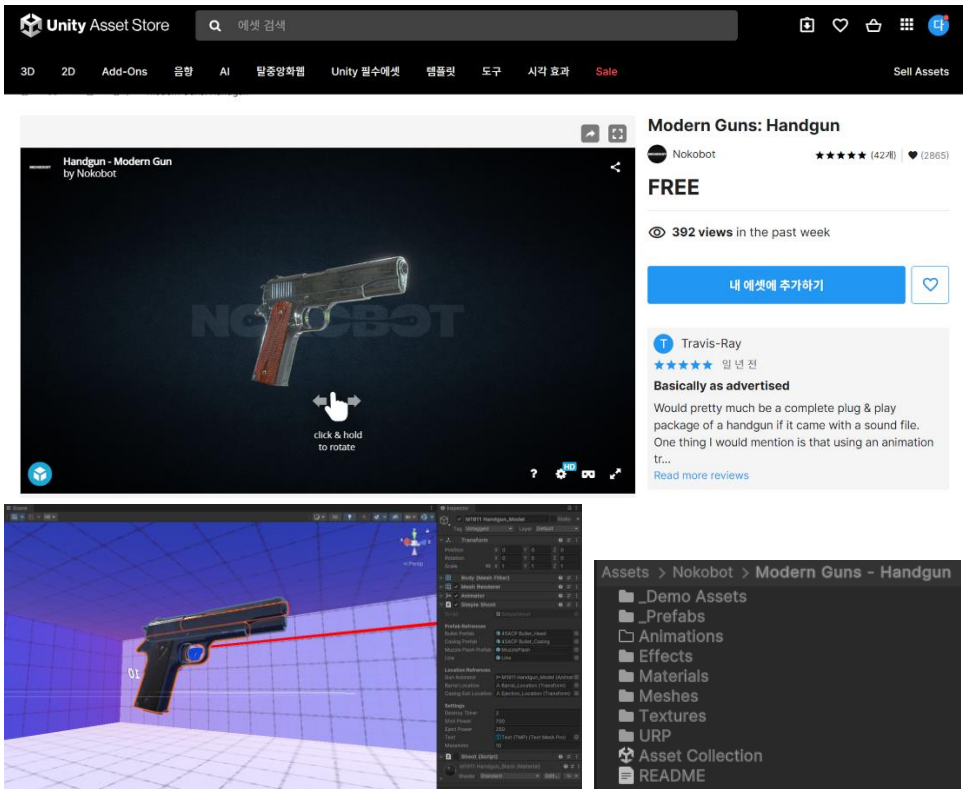
  ■ 캐릭터 피격 시 타격감을 주는 기능 활성화





- Pistol asset

  ■ Unity Asset Store에서 무료 제공하는 Modern Guns : Handgun import



3. Scripts

- SimpleShoot.cs

■ 플레이어가 총을 쏘기 위한 함수들을 작성하기 위한 변수 설정

```
public class SimpleShoot : MonoBehaviour
{
    // 프리팹 참조를 위한 공개 변수들
    public GameObject bulletPrefab;         // 발사되는 총알 프리팹
    public GameObject casingPrefab;         // 총에서 발사되는 탄피 프리팹
    public GameObject muzzleFlashPrefab;    // 총구 화염 효과 프리팹
    public GameObject line;                 // 발사 이펙트를 위한 라인 오브젝트 (스크립트에서 사용하지 않음)

    // 위치 참조를 위한 공개 변수들
    [SerializeField] private Animator gunAnimator;            // 총기를 제어하는 애니메이터 컴포넌트의 참조
    [SerializeField] private Transform barrelLocation;        // 총구 위치를 나타내는 Transform
    [SerializeField] private Transform casingExitLocation;    // 탄피가 배출되는 위치를 나타내는 Transform

    // 설정을 위한 공개 변수들
    [Tooltip("탄피 오브젝트를 파괴할 시간을 지정합니다.")]
    [SerializeField] private float destroyTimer = 2f;         // 배출된 탄피 파괴까지의 시간(초)
    [Tooltip("총알 속도")]
    [SerializeField] private float shotPower = 500f;          // 총알 발사 시 가해지는 힘
    [Tooltip("탄피 배출 속도")]
    [SerializeField] private float ejectPower = 150f;         // 탄피 배출 시 가해지는 힘

    public TMPro.TextMeshPro text;      // 탄약 수를 표시하는 TextMeshPro 컴포넌트를 위한 참조
    public int maxammo = 10;            // 최대 탄약 용량
    private int currentammo = 10;       // 현재 탄약 수
    public float shotPower = 100f;      // 총알이 주는 데미지
```

■ 초기 설정 및 버튼 눌림 감지를 통한 함수 실행

```
void Start()
{
    // 만약 총구 위치가 지정되어 있지 않으면 스크립트를 추가한 오브젝트의 위치를 총구 위치로 설정
    if (barrelLocation == null)
        barrelLocation = transform;

    // 만약 gunAnimator가 지정되어 있지 않으면 자식 오브젝트에서 애니메이터 컴포넌트를 찾아서 할당
    if (gunAnimator == null)
        gunAnimator = GetComponentInChildren<Animator>();
}

void Update()
{
    // "Fire1" 버튼이 눌렸을 때
    if (Input.GetButtonDown("Fire1"))
    {
        // 탄약이 남아있으면 발사 애니메이션 실행
        if (currentammo > 0)
            gunAnimator.SetTrigger("Fire");

        // 캐릭터가 정면을 바라보지 않았고, 탄약이 최대 용량보다 작을 경우 재장전 함수 호출
        if (Vector3.Angle(transform.up, Vector3.up) > 100 && currentammo < maxammo)
            Reload();

        // UI에 현재 탄약 수 표시
        text.text = currentammo.ToString();
    }
}
```

■ 재장전, 총알 발사, 탄피 배출 함수

```
// 재장전 함수
void Reload()
{
    currentammo = maxammo; // 현재 탄약 수를 최대 탄약 수로 설정
}

// 발사 함수
public void Shoot()
{
    currentammo--; // 탄약 소비

    // 총구 화염 효과 프리팹이 있으면 인스턴스화하여 임시 변수 tempFlash에 할당하고 일정 시간 후에 파괴
    if (muzzleFlashPrefab)
    {
        GameObject tempFlash;
        tempFlash = Instantiate(muzzleFlashPrefab, barrelLocation.position, barrelLocation.rotation);

        Destroy(tempFlash, destroyTimer);
    }

    // 총알 프리팹이 없으면 함수 종료
    if (!bulletPrefab)
    { return; }

    // 총알 프리팹을 총구 위치에 인스턴스화하고 총알에 힘을 가해 발사
    Instantiate(bulletPrefab, barrelLocation.position, barrelLocation.rotation).GetComponent<Rigidbody>().AddForce(barrelLocation.forward * shotPower);

    // 총구 방향으로 100 단위로 Raycast를 실행하여 충돌 정보를 hitInfo에 저장
    RaycastHit hitInfo;
    bool hasHit = Physics.Raycast(barrelLocation.position, barrelLocation.forward, out hitInfo, 100);

    // 충돌한 오브젝트에 'Dead' 메세지를 전달하여 피격 처리
    if (hasHit)
        hitInfo.collider.SendMessageUpwards("Dead", hitInfo.point, SendMessageOptions.DontRequireReceiver);
}

// 탄피 배출 함수
void CasingRelease()
{
    // 탄피 배출 위치나 프리팹이 없으면 함수 종료
    if (!casingExitLocation || !casingPrefab)
    { return; }

    // 탄피 프리팹을 배출 위치에 인스턴스화하고 랜덤한 힘과 회전력을 가해 탄피를 발사한 후 일정 시간 후에 파괴
    GameObject tempCasing;
    tempCasing = Instantiate(casingPrefab, casingExitLocation.position, casingExitLocation.rotation) as GameObject;
    tempCasing.GetComponent<Rigidbody>().AddExplosionForce(Random.Range(ejectPower * 0.7f, ejectPower), (casingExitLocation.position - casingExitLocation.right * 0.3f - casingExitLocation.up * 0.6f), 1f);
    tempCasing.GetComponent<Rigidbody>().AddTorque(new Vector3(0, Random.Range(100f, 500f), Random.Range(100f, 1000f)), ForceMode.Impulse);

    Destroy(tempCasing, destroyTimer);
}
```

- Move.cs

■ 플레이어가 자동으로 앞으로 이동하게 해주는 스크립트

```
public class Move : MonoBehaviour
{
    void Update()
    {
        //매 프레임마다 현재 위치를 z축 방향으로 2만큼 이동시킴
        //Time.deltaTime은 이전 프레임과 현재 프레임 사이의 시간 간격을 나타냄
        //Time.deltaTime을 곱하여 초당 이동 거리를 보정하여 부드러운 이동을 만듦
        //new Vector3(0, 0, 2)는 이동 방향을 나타냄 (x:0, y:0, z:2)
        transform.position += Time.deltaTime * new Vector3(0, 0, 2);
    }
}
```

- Enemy.cs

  ■ Ragdoll 및 카메라 설정

```
public class Enemy : MonoBehaviour
{
    public Shoot shooter;  // 적이 사용할 Shoot 스크립트의 참조 변수

    void Start()
    {
        SetupRagdoll(true);  // 레그돌 설정을 초기화하고 시작
    }

    // 레그돌 설정을 변경하는 함수
    void SetupRagdoll(bool value)
    {
        // 자식들 중 모든 Rigidbody 컴포넌트들의 isKinematic 값을 변경하여 물리적 영향을 받을 지 여부를 결정
        foreach (var item in GetComponentsInChildren<Rigidbody>())
        {
            item.isKinematic = value;
        }
    }

    void Update()
    {
        // 적이 항상 메인 카메라를 향하도록 설정함
        transform.forward = Vector3.ProjectOnPlane((Camera.main.transform.position - transform.position), Vector3.up).normalized;
    }
```

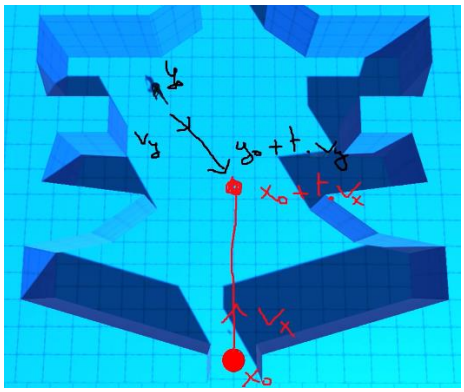
  ■ 적이 사망할 때 호출되는 함수

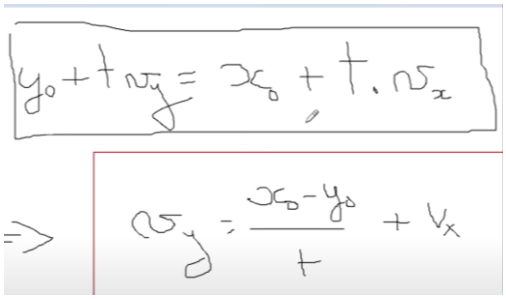
```
void Dead(Vector3 hitpoint)
{
    GetComponent<Animator>().enabled = false; // 애니메이터 비활성화
    SetupRagdoll(false); // 레그돌 설정을 활성화하여 물리적 영향을 받도록 함

    // 주변의 오브젝트들 중 hitpoint 근처에 있는 Rigidbody 컴포넌트들에 폭발력을 가함
    foreach (var item in Physics.OverlapSphere(hitpoint, 0.5f))
    {
        Rigidbody rb = item.GetComponent<Rigidbody>();
        if (rb)
            rb.AddExplosionForce(1000, hitpoint, 0.5f);
    }

    this.enabled = false; // 스크립트 비활성화
}
```

  ■ 적이 총을 쏠 때와 쏘는 대상의 위치를 반환하는 함수

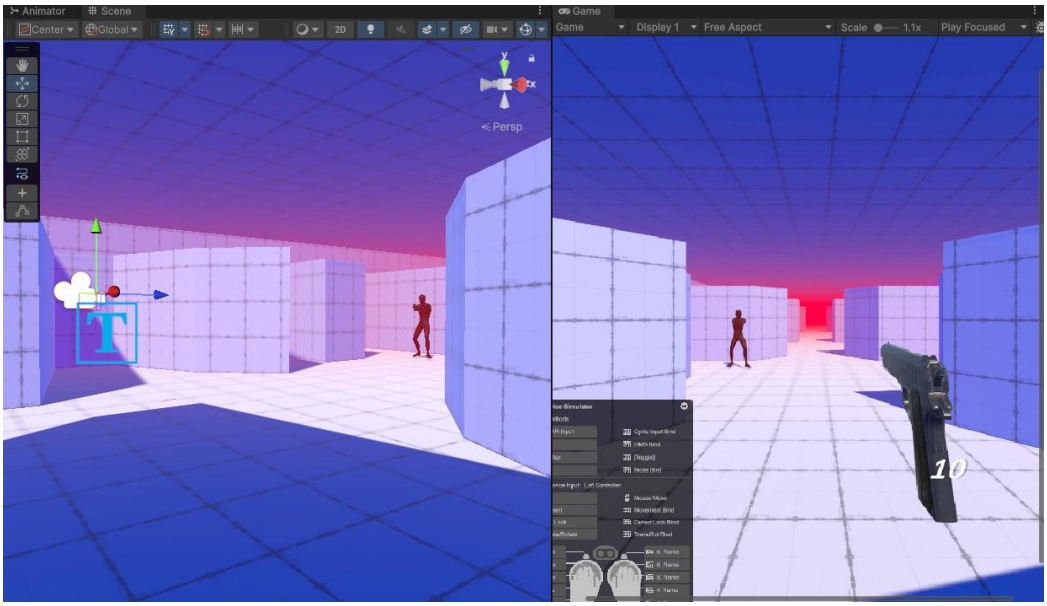
```
// 적이 총을 쏠 때 호출되는 함수
void Shoot()
{
    shooter.barrelLocation.forward = GetTarget().normalized;  // 적의 총구 방향을 GetTarget() 함수의 반환 값으로 설정
    shooter.shotPower = GetTarget().magnitude;  // 총알에 가해질 힘을 GetTarget() 함수의 반환 값의 크기로 설정
    shooter.Shoot();  // 슈터 스크립트의 Shoot() 함수 호출
}

// 적이 쏘는 대상의 위치를 반환하는 함수
Vector3 GetTarget()
{
    return Vector3.forward;  // 대상 위치를 전방으로 고정하여 반환 (실제로는 적이 원하는 대상 위치를 반환해야 함)
}
```

■ 플레이어가 자동으로 앞으로 이동하는 것을 고려해 미리 적이 총을 발사해 플레플레이 도달할 때 총알이 플레이어에게 도달하도록 계산





4. 실행 결과

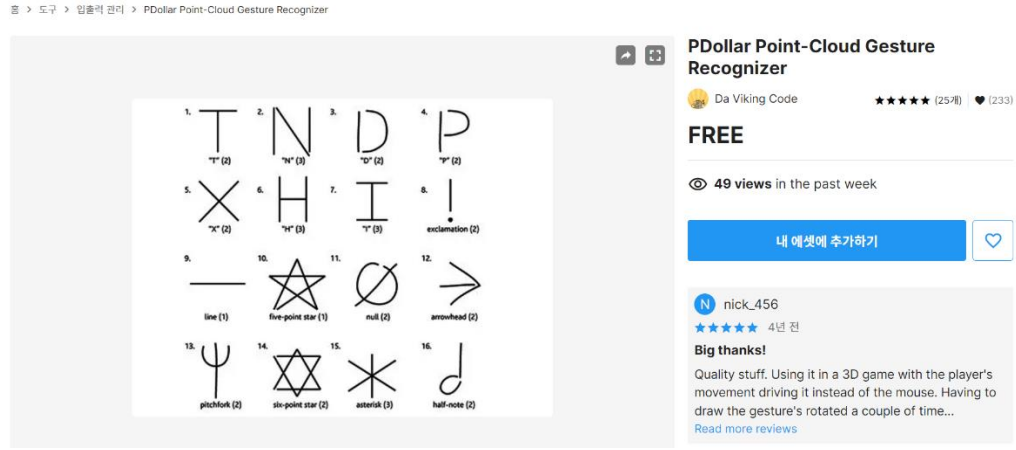


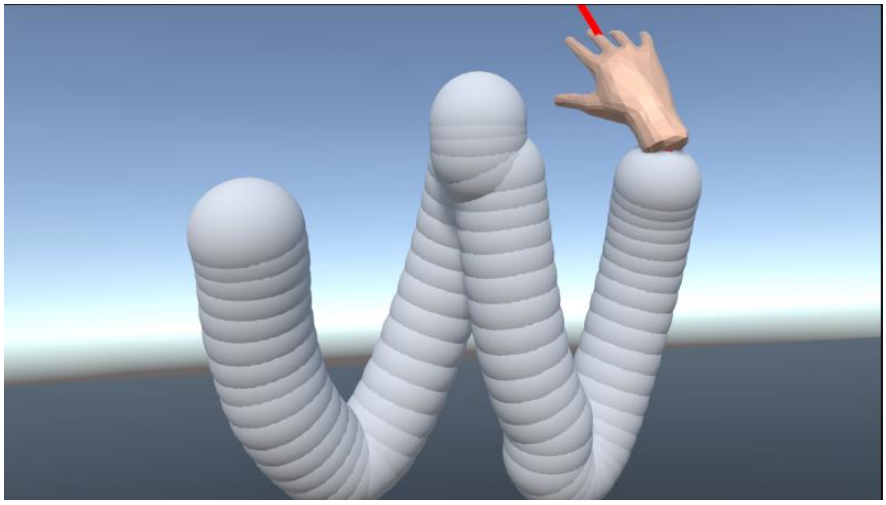
프로젝트 외 추가 기능 구현

1. 모션 트래킹:

- 현재 컨트롤러의 위치 정보를 기록하고 저장합니다.

- 저장된 정보와 적합성을 검사하기 위해 PDollar Point Asset을 활용합니다.





```
void endMovement()
{
    isMoving = false;

    // Convert recorded positions to screen coordinates and create a Gesture object.
    Point[] pointArray = new Point[positionList.Count];
    for (int i = 0; i < positionList.Count; i++)
    {
        Vector2 screenPoint = Camera.main.WorldToScreenPoint(positionList[i]);
        pointArray[i] = new Point(screenPoint.x, screenPoint.y, 0);
    }
    Gesture newGesture = new Gesture(pointArray);

    // If in creation mode, save the gesture to a file.
    if (creationMode)
    {
        newGesture.Name = newGestureName;
        testingSet.Add(newGesture);
        Debug.Log("Add Complete");

        string fileName = Application.persistentDataPath + "/" + newGestureName + ".xml";
        GestureIO.WriteGesture(pointArray, newGestureName, fileName);
    }
    // If not in creation mode, recognize the gesture and invoke the corresponding event if recognized.
    else
    {
        Result result = PointCloudRecognizer.Classify(newGesture, testingSet.ToArray());
        Debug.Log(result.GestureClass + " / " + result.Score);

        if (result.Score > recognitionGesture)
        {
            OnRecognized.Invoke(result.GestureClass);
        }
    }
}

// Function to update the movement by adding new positions to the recorded list.
void updateMovement()
{
    // Check if the movement has exceeded the 'newPositionDistance' threshold.
    Vector3 lastPosition = positionList[positionList.Count - 1];
    if (Vector3.Distance(movementSource.position, lastPosition) < newPositionDistance)
    {
        positionList.Add(movementSource.position);
        // Instantiate and display an object during movement (if provided).
        if (displayObject != null)
        {
            Destroy(Instantiate(displayObject, movementSource.position, Quaternion.identity), 3);
        }
    }
```

2. 수영:

- 컨트롤러의 움직임에 반대되는 방향으로 물리력을 작용하여 수영 동작을 구현합니다.

- InputActionReference 함수를 활용하여 물리적인 움직임을 구현합니다.

Class InputActionReference

References a specific InputAction in an InputActionMap stored inside an InputActionAsset.

Inheritance

↳ System.Object
  ↳ InputActionReference

Namespace: UnityEngine.Experimental.Input

Syntax

```
public class InputActionReference : ScriptableObject
```

Remarks

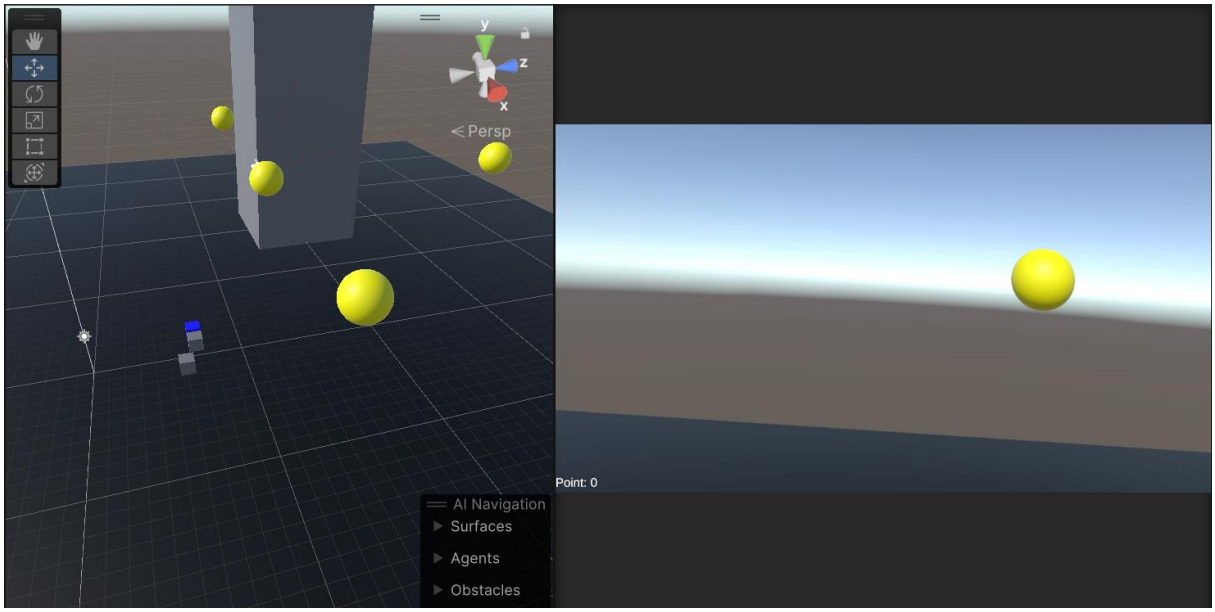
The difference to a plain reference directly to an InputAction object is that an InputActionReference can be serialized without causing the referenced InputAction to be serialized as well. The reference will remain intact even if the action or the map that contains the action is renamed.

References can be set up graphically in the editor by dropping individual actions from the project browser onto a reference field.

Properties

action

The action that the reference resolves to.

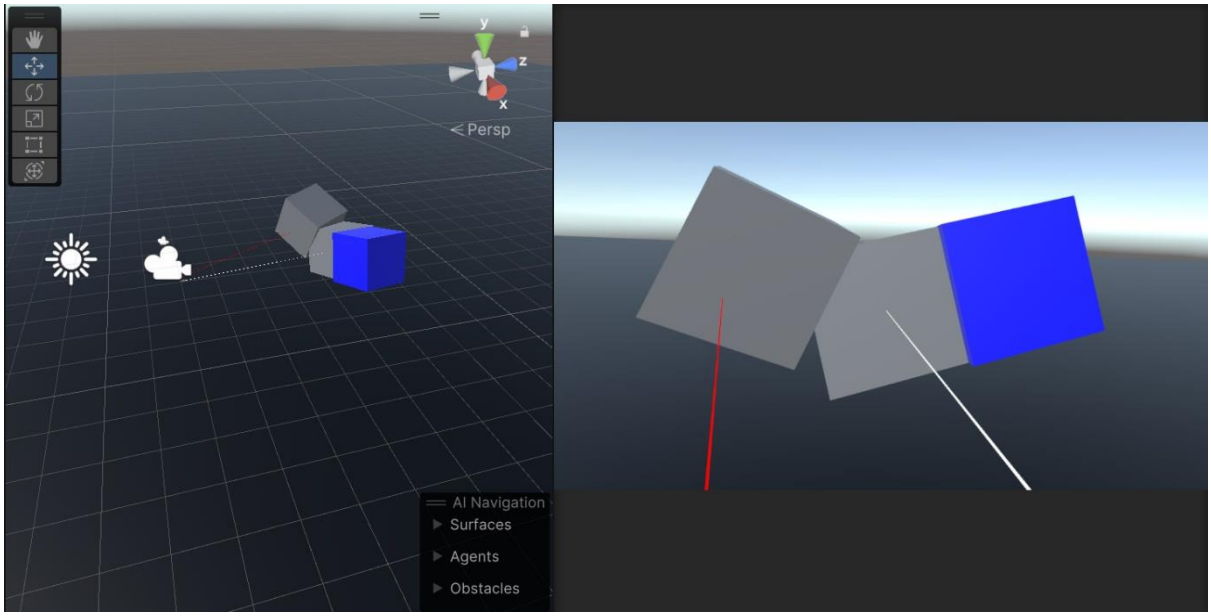


```
public class SwimControll : MonoBehaviour
{
    // Variables to control swim behavior
    [Header("Value")]
    [SerializeField] float swimForce = 2f;  // The force applied when swimming
    [SerializeField] float dragForce = 1f;  // The force applied to slow down the swimmer
    [SerializeField] float minForce;        // The minimum force required to swim
    [SerializeField] float minTimeBetweenStrokes; // The minimum time between strokes

    // References to input actions and a tracking reference
    [Header("Reference")]
    [SerializeField] InputActionReference leftControllerSwimReference;   // Input action for the left controller swim command
    [SerializeField] InputActionReference leftControllerVelocity;       // Input action to get the velocity of the left controller
    [SerializeField] InputActionReference rightControllerSwimReference;  // Input action for the right controller swim command
    [SerializeField] InputActionReference rightControllerVelocity;       // Input action to get the velocity of the right controller
    [SerializeField] Transform trackingReference;                        // A reference transform used to convert local velocity to world velocity

    Rigidbody _rigidbody;   // Reference to the Rigidbody component of the swimmer
    float _cooldownTimer;   // Timer to control the time between strokes

    void Awake()
    {
        // Get the Rigidbody component and set initial constraints
        _rigidbody = GetComponent<Rigidbody>();
        _rigidbody.useGravity = false;
        _rigidbody.constraints = RigidbodyConstraints.FreezeRotation;
    }

    void FixedUpdate()
    {
        // Update the cooldown timer with the time since the last fixed update
        _cooldownTimer = Time.fixedDeltaTime;

        // Check if both left and right swim buttons are pressed and enough time has passed since the last stroke
        if (_cooldownTimer > minTimeBetweenStrokes
            && leftControllerSwimReference.action.IsPressed()
            && rightControllerSwimReference.action.IsPressed())
        {
            // Calculate the combined local velocity of both hands and invert it
            var leftHandVelocity = leftControllerVelocity.action.ReadValue<Vector3>();
            var rightHandVelocity = rightControllerVelocity.action.ReadValue<Vector3>();
            Vector3 localVelocity = leftHandVelocity + rightHandVelocity;
            localVelocity *= -1;

            // Check if the magnitude of the local velocity is greater than the minimum force required to swim
            if (localVelocity.sqrMagnitude > minForce * minForce)
            {
                // Convert the local velocity to world velocity using the tracking reference
                Vector3 worldVelocity = trackingReference.TransformDirection(localVelocity);

                // Apply swim force to the swimmer in the direction of the world velocity
```

3. 오브젝트 합치기:

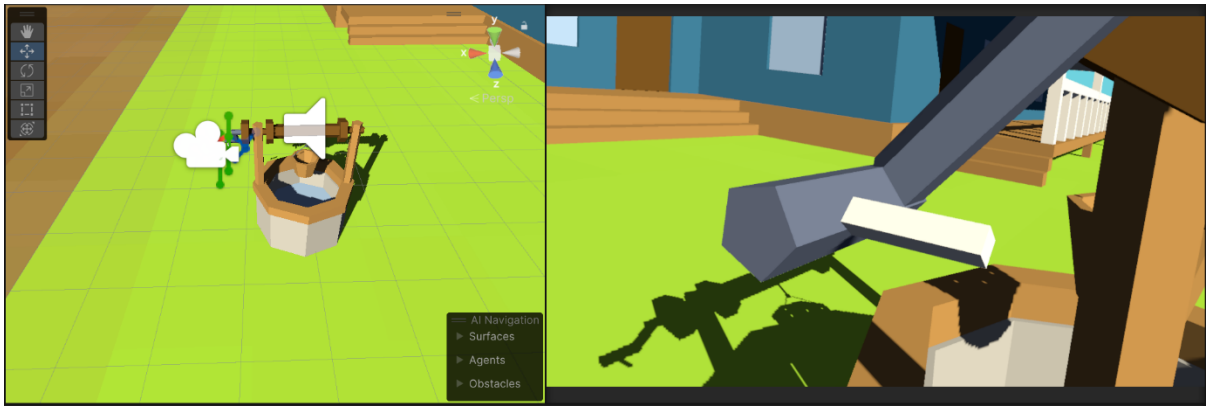
- 하나의 오브젝트를 다른 오브젝트에 부착하는 기능을 구현합니다.

- 두 오브젝트의 상대적인 위치 데이터를 기반으로 특정 위치에 오브젝트를 붙일 수 있는 기능을 제공합니다.



4. 오브젝트 상호작용:

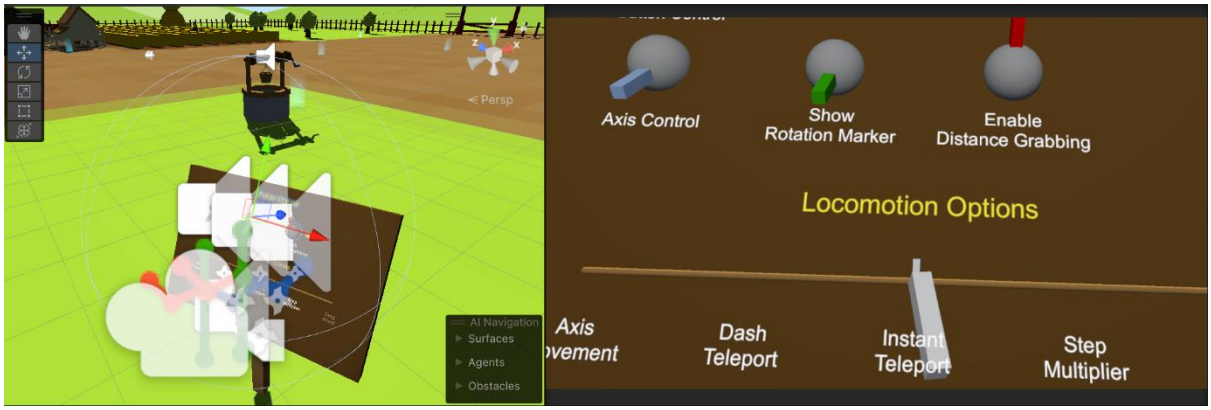
- 회전 가능한 원형 오브젝트를 돌리는 기능을 구현합니다.



5. 설정 UI:

- 게임 옵션을 변경할 수 있는 UI를 왼손에 구현합니다.

- 사용자가 게임 설정을 조정할 수 있는 인터페이스를 제공합니다.

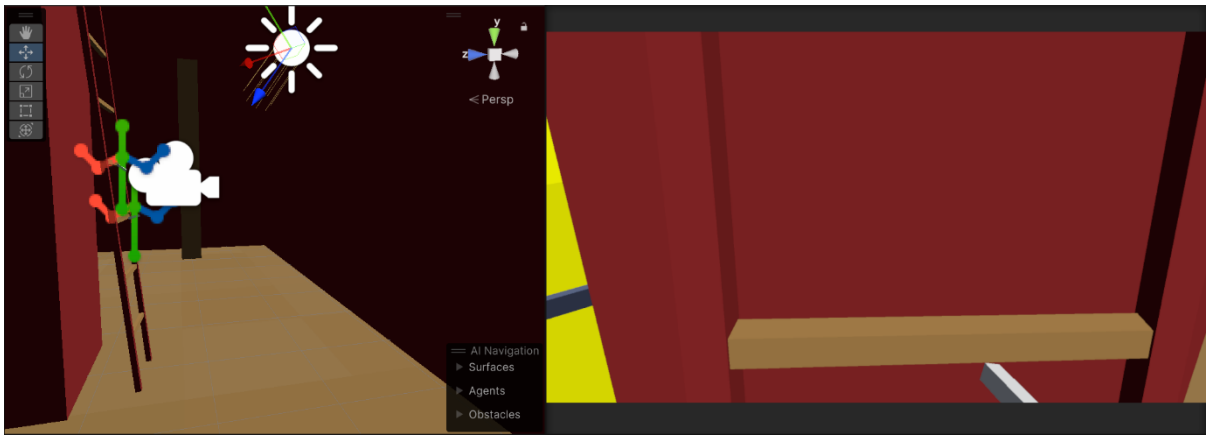


```
public virtual void SetLocation()
{
    // Check if LocationTarget or LocationDirection is null; if so, return early
    if (LocationTarget == null || LocationDirection == null)
    {
        return;
    }

    // Get the positions of the LocationDirection and LocationTarget objects
    Vector3 locationDirectionPosition = LocationDirection.transform.position;
    Vector3 locationTargetPosition = LocationTarget.transform.position;

    // Set the position of the current object (assuming this is attached to a script component)
    // The object's X and Z positions will match the LocationDirection object's position,
    // while its Y position will match the LocationTarget object's position.
    transform.position = new Vector3(locationDirectionPosition.x, locationTargetPosition.y, locationDirectionPosition.z);

    // Move the "ControlStation" (presumably another object) to a position offset from the LocationDirection object's forward direction
    // The amount of offset is determined by the LocationOffset variable.
    ControlStation.transform.localPosition = LocationDirection.transform.forward * LocationOffset;

    // Scale the ControlStation's local position by the sum of Vector3.right and Vector3.forward.
    // However, this operation does not assign the result back to the ControlStation's local position,
    // so it won't have any effect on the object's position.
    Vector3.Scale(ControlStation.transform.localPosition, Vector3.right + Vector3.forward);

    // Make the ControlStation object look at a specific target position, which is determined by the locationDirectionPosition and locationTargetPosition.
    // The ControlStation's rotation will change to point towards the targetPosition.
    ControlStation.transform.LookAt(locationDirectionPosition);

    // Rotate the ControlStation object around the Y-axis by 180 degrees, adding a half-turn rotation.
    // This is done by modifying the localEulerAngles of the ControlStation.
    ControlStation.transform.localEulerAngles = Vector3.up * (ControlStation.transform.localEulerAngles.y + 180f);
}

protected virtual void OnEnable()
{
    // Call the method that configures toggle visibility when the object is enabled
    ConfigureToggleVisibility();
}

/// <summary>
/// Links the provided ActivationAction action to the internal LinkedInternalAction.
/// </summary>
protected virtual void ConfigureToggleVisibility()
{
    // Clear any existing sources from the LinkedInternalAction when it's active and enabled.
    // Then add the provided ActivationAction as a source to the LinkedInternalAction.
    LinkedInternalAction.RunWhenActiveAndEnabled(() => LinkedInternalAction.ClearSources());
    LinkedInternalAction.RunWhenActiveAndEnabled(() => LinkedInternalAction.AddSource(ActivationAction));
}
```

6. 사다리 오르기:

- 손잡이를 잡고 오를 수 있는 사다리를 구현합니다.

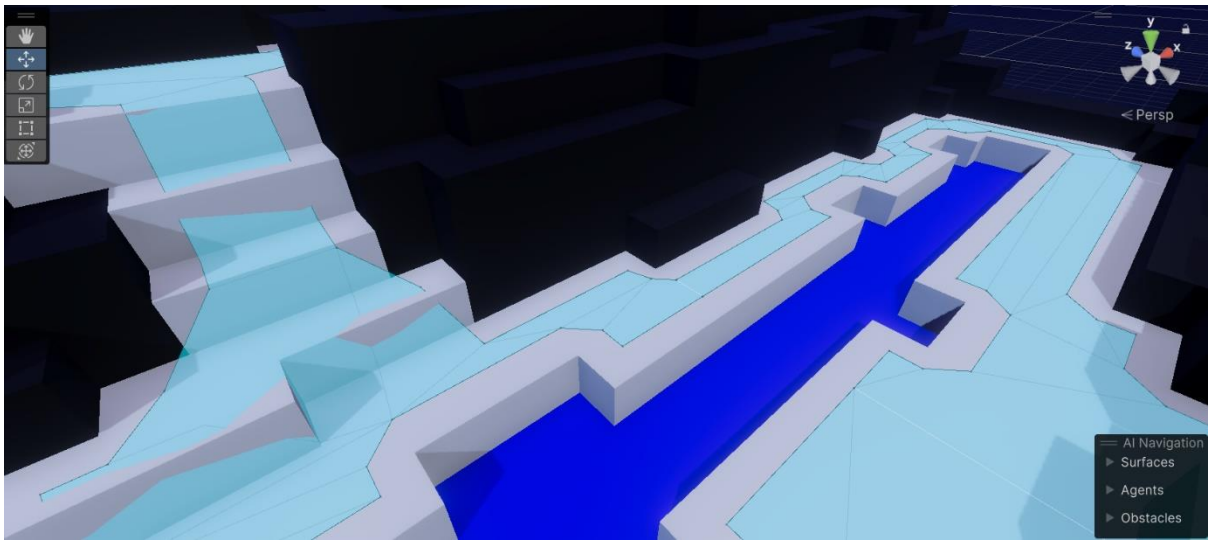
- 플레이어가 사다리를 오를 수 있는 기능을 제공합니다.

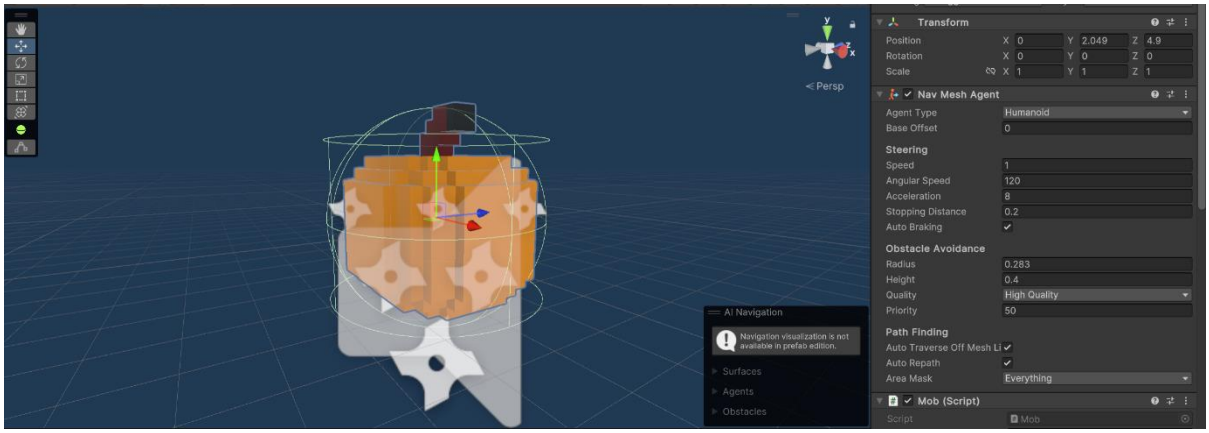


7. 추적 및 루트 생성:

- NavMeshAgent 기능을 활용하여 몬스터가 이동할 수 있는 공간을 설정하고, 특정 오브젝트를 향해 이동하도록 합니다.

- 추적을 위해 루트를 생성하고, 몬스터가 따라가는 경로를 설정합니다.

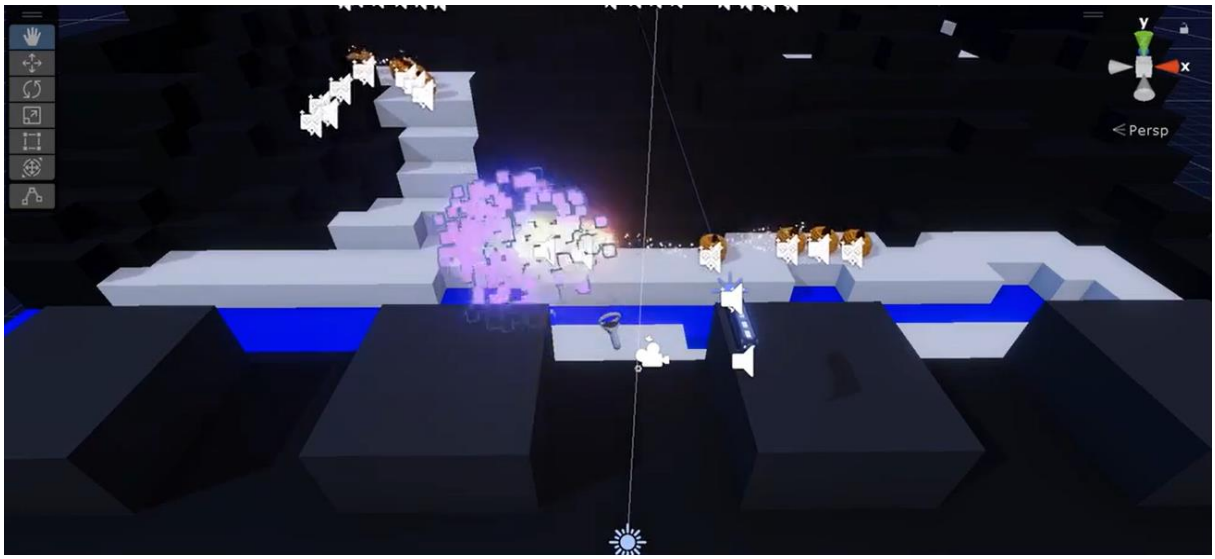




8. 이펙트 및 소리 구현:

- 화염 이펙트와 폭발 이펙트를 구현합니다.

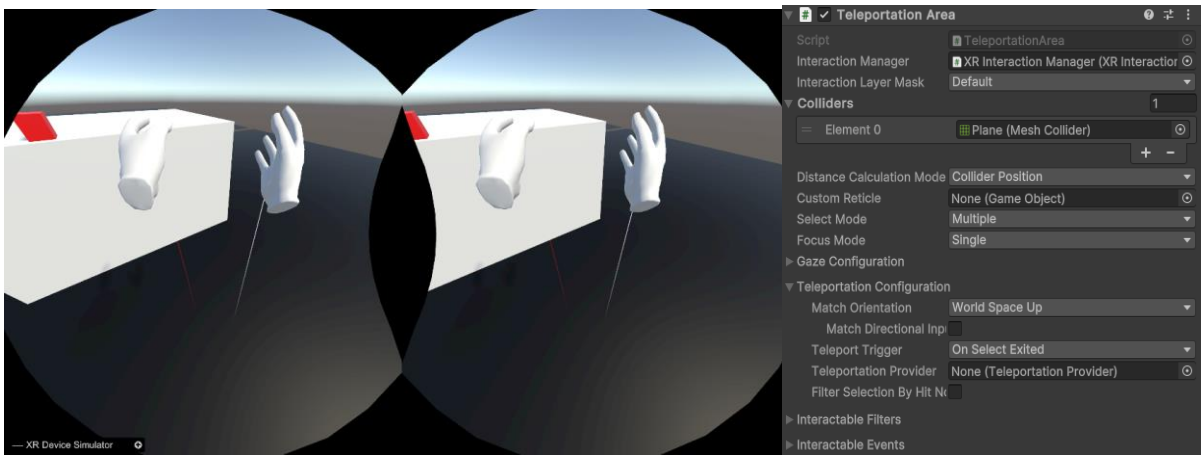
- 시각적인 효과와 함께 음향적인 효과를 제공합니다.




```
using UnityEngine.Events;
using UnityEngine.XR.Interaction.Toolkit;

public void Throw()
{
    // Get the XRGrabInteractable component attached to the current object
    var interactable = GetComponent<XRGrabInteractable>();

    // Cancel the selection of the interactable using the interaction manager
    interactable.interactionManager.CancelInteractableSelection((IXRSelectInteractable)interactable);

    // Get the Rigidbody component attached to the current object
    var rb = GetComponent<Rigidbody>();

    // Apply a relative force to the Rigidbody, making the object move
    rb.AddRelativeForce(new Vector3(0f, 150f, 300f));
}

private void OnTriggerEnter(Collider other)
{
    // If the state is idle, return without performing any further actions
    if (state == State.Idle)
        return;

    // Call the Explosion method
    Explosion();
}

private void Explosion()
{
    // Find all colliders within a sphere around the object's position using physics overlaps
    var overlaps = Physics.OverlapSphere(transform.position, explosionRadius, explosionHittableMask, QueryTriggerInteraction.Collide);

    // Iterate through each overlapped collider
    foreach (var overlap in overlaps)
    {
        // Get the Hittable component from the overlapped collider
        var hitObject = overlap.GetComponent<Hittable>();

        // If a Hittable component is found, call its Hit method
        hitObject?.Hit();
    }

    // Invoke the OnExplosion event, if any listeners are attached
    OnExplosion?.Invoke();

    // Call the Recycle method after a delay specified by recycleDelay
    Invoke(nameof(Recycle), recycleDelay);
}
```

9. 텔레포트

- 지정한 버튼을 클릭해서 순간이동 가능한 구역의 바닥을 클릭하면 텔레포트

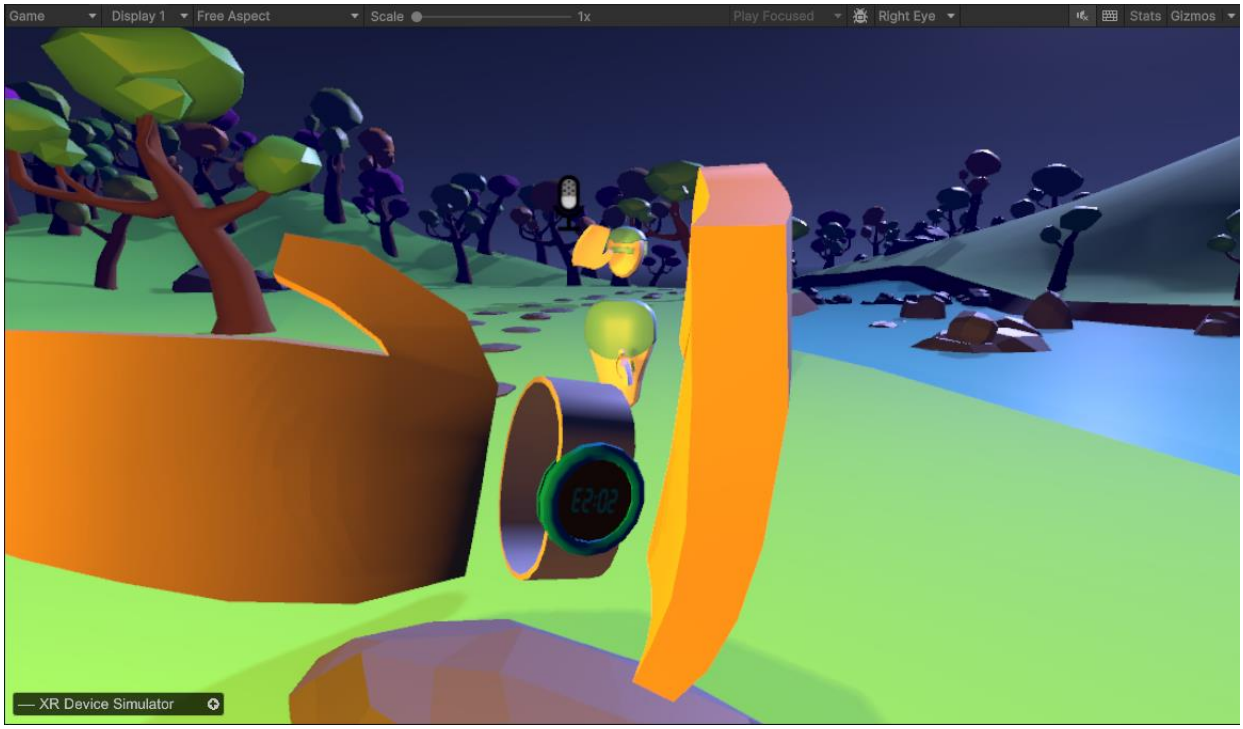


3. 서버 개발팀 진행 사항

Multiplayer Virtual Reality (VR) Development With Unity

Unity와 Photon 연동을 위해 Unity 학습을 진행함.

3.1 서버 개발팀 결과물



3.2 소스 코드 및 설명

```
namespace Oculus.Platform.Samples.VrVoiceChat
{
    using UnityEngine;
    using System;
    using System.Collections.Generic;
    using Oculus.Platform;
    using Oculus.Platform.Models;

    public class RoomManager
    {
        // the ID of the Room that I'm in
        private ulong m_roomID;

        // the other User in the Room
        private User m_remoteUser;

        // how often I should poll for invites
        private static readonly float INVITE_POLL_FREQ_SECONDS = 5.0f;

        // the next time I should poll Oculus Platform for valid Room Invite requests
        private float m_nextPollTime;

        public struct Invite
        {
            public readonly ulong RoomID;
            public readonly string OwnerID;

            public Invite(ulong roomID, string owner)
            {
                this.RoomID = roomID;
                this.OwnerID = owner;
            }
        }

        // cached list of rooms that I've been invited to and I'm waiting
        // for more information about
        private HashSet<ulong> m_pendingRoomRequests;

        // accumulation list of room invites and the room owner
        private List<Invite> m_invites;

        public RoomManager()
        {
            Rooms.SetRoomInviteAcceptedNotificationCallback(LaunchedFromAcceptingInviteCallback);
            Rooms.SetUpdateNotificationCallback(RoomUpdateCallback);
        }

        public ulong RemoteUserID
```

```
{
    get
    {
        return m_remoteUser != null ? m_remoteUser.ID : 0;
    }
}

public String RemoteOculusID
{
    get
    {
        return m_remoteUser != null ? m_remoteUser.OculusID : String.Empty;
    }
}

#region Launched Application from Accepting Invite

// Callback to check whether the User accepted the invite as
// a notification which caused the Application to launch.  If so, then
// we know we need to try to join that room.
void LaunchedFromAcceptingInviteCallback(Message<string> msg)
{
    if (msg.IsError)
    {
        PlatformManager.TerminateWithError(msg);
        return;
    }

    Debug.Log("Launched Invite to join Room: " + msg.Data);

    m_roomID = Convert.ToUInt64(msg.GetString());
}

// Check to see if the App was launched by accepting the Notication from the main Oculus app.
// If so, we can directly join that room.  (If it's still available.)
public bool CheckForLaunchInvite()
{
    if (m_roomID != 0)
    {
        JoinExistingRoom(m_roomID);
        return true;
    }
    else
    {
        return false;
    }
}

#endregion

#region Create a Room and Invite Friend(s) from the Oculus Universal Menu

public void CreateRoomAndLaunchInviteMenu()
{
    Rooms.CreateAndJoinPrivate(RoomJoinPolicy.InvitedUsers, 2, true)
         .OnComplete(CreateAndJoinPrivateRoomCallback);
}

void CreateAndJoinPrivateRoomCallback(Message<Room> msg)
{
    if (msg.IsError)
    {
        PlatformManager.TerminateWithError(msg);
        return;
    }

    m_roomID = msg.Data.ID;
    m_remoteUser = null;
```

```
        PlatformManager.TransitionToState(PlatformManager.State.WAITING_FOR_ANSWER);

        // launch the Room Invite workflow in the Oculus Univeral Menu
        Rooms.LaunchInvitableUserFlow(m_roomID).OnComplete(OnLaunchInviteWorkflowComplete);
    }

    void OnLaunchInviteWorkflowComplete(Message msg)
    {
        if (msg.IsError)
        {
            PlatformManager.TerminateWithError(msg);
            return;
        }
    }

    #endregion

    #region Polling for Invites

    public bool ShouldPollInviteList
    {
        get
        {
            return m_pendingRoomRequests == null && Time.time >= m_nextPollTime;
        }
    }

    public void UpdateActiveInvitesList()
    {
        m_nextPollTime = Time.time + INVITE_POLL_FREQ_SECONDS;
        m_pendingRoomRequests = new HashSet<ulong>();
        m_invites = new List<Invite>();
        Notifications.GetRoomInviteNotifications().OnComplete(GetRoomInviteNotificationsCallback);
    }

    // task 13572454: add the type to callback definition
    void GetRoomInviteNotificationsCallback(Message msg_untyped)
    {
        Message<RoomInviteNotificationList> msg = (Message<RoomInviteNotificationList>)msg_untyped;

        if (msg.IsError)
        {
            PlatformManager.TerminateWithError(msg);
            return;
        }

        // loop over all the rooms we're invited to and request more info
        foreach (RoomInviteNotification invite in msg.Data)
        {
            m_pendingRoomRequests.Add(invite.RoomID);
            Rooms.Get(invite.RoomID).OnComplete(GetRoomInfoCallback);
        }

        if (msg.Data.Count == 0)
        {
            m_pendingRoomRequests = null;
            PlatformManager.SetActiveInvites(m_invites);
        }
    }

    void GetRoomInfoCallback(Message<Room> msg)
    {
        if (msg.IsError)
        {
            PlatformManager.TerminateWithError(msg);
            return;
        }
```

```
        if (msg.Data.OwnerOptional != null)
        {
            Invite invite = new Invite(msg.Data.ID, msg.Data.OwnerOptional.OculusID);
            m_pendingRoomRequests.Remove(invite.RoomID);

            // make sure the room still looks usable
            // (e.g. they aren't currently talking to someone)
            if (msg.Data.UsersOptional != null && msg.Data.UsersOptional.Count == 1)
            {
                m_invites.Add(invite);
            }
        }

        // once we're received all the room info, let the platform update
        // its display
        if (m_pendingRoomRequests.Count == 0)
        {
            m_pendingRoomRequests = null;
            PlatformManager.SetActiveInvites(m_invites);
        }
    }

    #endregion

    #region Accept Invite

    public void JoinExistingRoom(ulong roomID)
    {
        Rooms.Join(roomID, true).OnComplete(JoinRoomCallback);
    }

    void JoinRoomCallback(Message<Room> msg)
    {
        if (msg.IsError)
        {
            // is reasonable if caller called more than 1 person, and I didn't answer first
            return;
        }

        string oculusOwnerID = msg.Data.OwnerOptional != null ? msg.Data.OwnerOptional.OculusID : "";
        int numUsers = msg.Data.UsersOptional != null ? msg.Data.UsersOptional.Count : 0;

        Debug.LogFormat("Joined room: {0} owner: {1} count: ",
            msg.Data.ID, oculusOwnerID, numUsers);

        m_roomID = msg.Data.ID;

        // if the caller left while I was in the process of joining, just hangup
        if (msg.Data.UsersOptional == null || msg.Data.UsersOptional.Count != 2)
        {
            PlatformManager.TransitionToState(PlatformManager.State.HANGUP);
        }
        else
        {
            foreach (User user in msg.Data.UsersOptional)
            {
                if (user.ID != PlatformManager.MyID)
                {
                    m_remoteUser = user;
                }
            }

            PlatformManager.TransitionToState(PlatformManager.State.CONNECTED_IN_A_ROOM);
        }

        // update the invite list sooner
        m_nextPollTime = Time.time;
    }
```

```
        #endregion

        #region Room Updates

        void RoomUpdateCallback(Message<Room> msg)
        {
            if (msg.IsError)
            {
                PlatformManager.TerminateWithError(msg);
                return;
            }

            string oculusOwnerID = msg.Data.OwnerOptional != null ? msg.Data.OwnerOptional.OculusID : "";
            int numUsers = msg.Data.UsersOptional != null ? msg.Data.UsersOptional.Count : 0;

            Debug.LogFormat("Room {0} Update: {1} owner: {2} count: ",
                msg.Data.ID, oculusOwnerID, numUsers);

            // if the Room count is not 2 then the other party has left.
            // We'll just hangup the connection here.
            // If the other User created then room, ownership would switch to me.
            if (msg.Data.UsersOptional == null || msg.Data.UsersOptional.Count != 2)
            {
                PlatformManager.TransitionToState(PlatformManager.State.HANGUP);
            }
            else
            {
                foreach (User user in msg.Data.UsersOptional)
                {
                    if (user.ID != PlatformManager.MyID)
                    {
                        m_remoteUser = user;
                    }
                }

                PlatformManager.TransitionToState(PlatformManager.State.CONNECTED_IN_A_ROOM);
            }
        }

        #endregion

        #region Room Exit

        public void LeaveCurrentRoom()
        {
            if (m_roomID != 0)
            {
                Rooms.Leave(m_roomID);
                m_roomID = 0;
                m_remoteUser = null;
            }
            PlatformManager.TransitionToState(PlatformManager.State.WAITING_TO_CALL_OR_ANSWER);
        }

        #endregion
    }
}
```

RoomManager Class:

사용자가 공유 경험을 만드는 데 도움이 되는 방을 생성, 가입 및 초대하는 것을 관리하는 도우미 클래스

- `LaunchedFromAcceptingInviteCallback()` : 사용자가 방 초대를 수락하면 호출되는 메서드. 이 메서드는 사 용자가 참여한 방의 ID를 가져옴.
- `CheckForLaunchInvite()` : 사용자가 Oculus 앱의 알림을 수락하여 애플리케이션이 시작되었는지 확인하는 메서드. 해당 방에 직접 참여할 수 있음.
- `CreateRoomAndLaunchInviteMenu()` : 새로운 방을 만들고 Oculus Univeral 메뉴에서 방 초대 워크플로를 시작하는 메서드.
- `GetRoomInfoCallback()` : RoomManager가 사용자가 초대받은 방에 대한 자세한 정보를 수신하면 호출되는 메서드. RoomManager가 추적하는 초대 목록을 업데이트함.
- `JoinExistingRoom()`: 기존 방에 참여하는 메서드.
- `RoomUpdateCallback()` : 사용자가 있는 방에 업데이트가 있을 때 호출되는 메서드. 방의 사용자 목록을 업데이트함.
- `LeaveCurrentRoom()`: 현재 방을 떠나는 메서드

```
using System;
using System.Collections;

class Example
{
    public static void Main()
    {
        // Create a new hash table.
        //
        Hashtable openWith = new Hashtable();

        // Add some elements to the hash table. There are no
        // duplicate keys, but some of the values are duplicates.
        openWith.Add("txt", "notepad.exe");
        openWith.Add("bmp", "paint.exe");
        openWith.Add("dib", "paint.exe");
        openWith.Add("rtf", "wordpad.exe");

        // The Add method throws an exception if the new key is
        // already in the hash table.
        try
        {
            openWith.Add("txt", "winword.exe");
        }
        catch
        {
            Console.WriteLine("An element with Key = \"txt\" already exists.");
        }

        // The Item property is the default property, so you
        // can omit its name when accessing elements.
        Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

        // The default Item property can be used to change the value
        // associated with a key.
        openWith["rtf"] = "winword.exe";
        Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

        // If a key does not exist, setting the default Item property
        // for that key adds a new key/value pair.
```

```
        openWith["doc"] = "winword.exe";

        // ContainsKey can be used to test keys before inserting
        // them.
        if (!openWith.ContainsKey("ht"))
        {
            openWith.Add("ht", "hypertrm.exe");
            Console.WriteLine("Value added for key = \"ht\": {0}", openWith["ht"]);
        }

        // When you use foreach to enumerate hash table elements,
        // the elements are retrieved as KeyValuePair objects.
        Console.WriteLine();
        foreach (DictionaryEntry de in openWith)
        {
            Console.WriteLine("Key = {0}, Value = {1}", de.Key, de.Value);
        }

        // To get the values alone, use the Values property.
        ICollection valueColl = openWith.Values;

        // The elements of the ValueCollection are strongly typed
        // with the type that was specified for hash table values.
        Console.WriteLine();
        foreach (string s in valueColl)
        {
            Console.WriteLine("Value = {0}", s);
        }

        // To get the keys alone, use the Keys property.
        ICollection keyColl = openWith.Keys;

        // The elements of the KeyCollection are strongly typed
        // with the type that was specified for hash table keys.
        Console.WriteLine();
        foreach (string s in keyColl)
        {
            Console.WriteLine("Key = {0}", s);
        }

        // Use the Remove method to remove a key/value pair.
        Console.WriteLine("\nRemove(\"doc\")");
        openWith.Remove("doc");

        if (!openWith.ContainsKey("doc"))
        {
            Console.WriteLine("Key \"doc\" is not found.");
        }
    }
}

/* This code example produces the following output:

An element with Key = "txt" already exists.
For key = "rtf", value = wordpad.exe.
For key = "rtf", value = winword.exe.
Value added for key = "ht": hypertrm.exe

Key = dib, Value = paint.exe
Key = txt, Value = notepad.exe
Key = ht, Value = hypertrm.exe
Key = bmp, Value = paint.exe
Key = rtf, Value = winword.exe
Key = doc, Value = winword.exe

Value = paint.exe
Value = notepad.exe
```

- txt -> winword.exe 요소를 해시 테이블에 추가하려고 시도. 이 요청은 키 txt 가 이미 해시 테이블에 존재하기 때문에 실패.
- 키 rtf 와 관련된 값을 가져옴. 값은 wordpad.exe .
- 키 rtf 와 관련된 값을 winword.exe 로 설정.
- doc -> winword.exe 요소를 해시 테이블에 추가.
- 키 ht 가 해시 테이블에 존재하는지 확인 키가 존재하지 않으므로 코드는 ht -> hypertrm.exe 요소를 해시 테이블에 추가.
- foreach 루프를 사용하여 해시 테이블의 요소를 반복. 요소는 KeyValuePair 개체로 가져옴.
- 해시 테이블의 모든 요소의 값을 가져옴.
- 해시 테이블의 모든 요소의 키를 가져옴.
- doc 요소를 해시 테이블에서 제거.
- 키 doc 가 여전히 해시 테이블에 존재하는지 확인. 키가 존재하지 않으므로 코드는 "Key "doc" is not found." 메시지를 출력.

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

public class SpawnManager : MonoBehaviour
{

    [SerializeField]
    GameObject GenericVRPlayerPrefab;

    public Vector3 spawnPosition;

    // Start is called before the first frame update
    void Start()
    {
        if (PhotonNetwork.IsConnectedAndReady)
        {
            PhotonNetwork.Instantiate(GenericVRPlayerPrefab.name, spawnPosition, Quaternion.identity);
        }
    }
```

SpawnManager 클래스의 두 개의 공개 변수.

- GenericVRPlayerPrefab 는 새로운 플레이어가 게임에 참여할 때 인스턴스화될 프리뱁을 나타내는 GameObject.
- spawnPosition 은 새로운 플레이어가 스폰될 위치를 나타내는 Vector3.

SpawnManager 클래스의 두 가지 메서드.

- Start() 는 게임이 시작될 때 호출. 이 메서드에서 SpawnManager 는 플레이어가 네트워크에 연결되어 플레이할 준비가 되었는지 확인. 플레이어가 연결되어 준비가 되어 있으면 SpawnManager 는 spawnPosition 에 지정된 위치에 GenericVRPlayerPrefab 유형의 새로운 GameObject를 인스턴스화.
- Update() 는 매 프레임마다 호출됨.