UNIVERSITÀ
DEGLI STUDI
FIRENZE

Quantum Key Hunt VR

Mirko Bicchierai, Niccolò Marini

Course: Computer Graphics

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Introduction

Game Overview

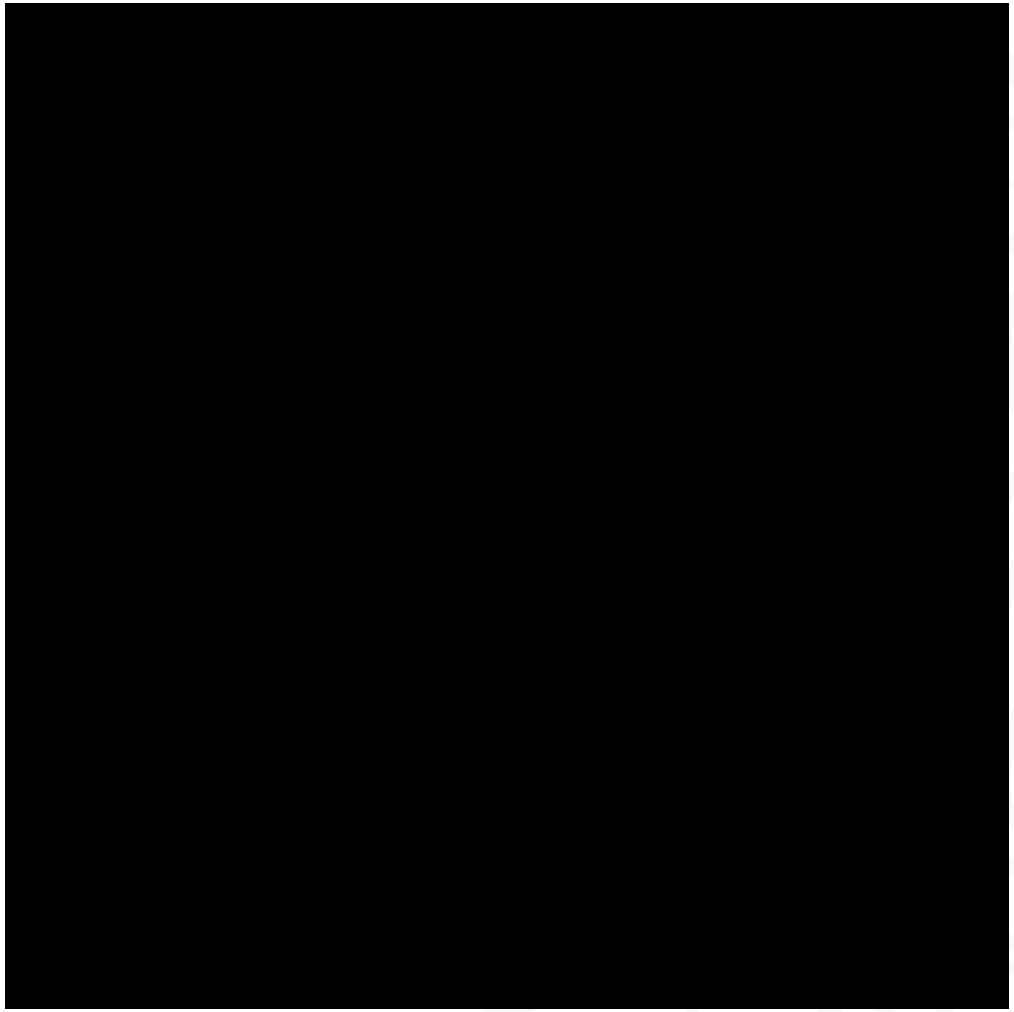
Quantum Key Hunt is an immersive VR game developed in Unity, where players are challenged to shoot moving targets to unlock a door and complete each level.

Tools and Technologies

- **Unity**: The game is built on Unity, leveraging its powerful engine for immersive VR experiences and robust gameplay mechanics.
- **XR Interaction Toolkit**: Utilizing Unity's XR Interaction Toolkit to translate VR user input into game input

UNIVERSITÀ DEGLI STUDI FIRENZE

Showcase Video



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Game Design

Assets pack and Level System

Assets

The main assets packs used during the project are:

- **Sci-Fi assets Pack** (Materials/Stairs/Doors)
- **Sci-Fi Guns** (Low Poly)
- **Attachment** (C4/Torch/Laser/Scope , Low Poly)

Conversion to URP: To achieve optimal performance and visual fidelity in our game, we converted these asset packs to the Universal Render Pipeline (URP). URP provides several advantages:

- **Enhanced Performance**:
- **Improved Graphics Quality**:
- **Consistency**

Universal Render Pipeline (URP)

- The Universal Render Pipeline (URP) is a **rendering solution** provided by Unity designed to offer high performance and graphical fidelity across a wide range of platforms.
- URP **optimizes** rendering processes to improve performance on both high-end and low-end devices.
- It features advanced rendering techniques, such as better lighting and shading effects, and ensures consistent visual quality across various hardware.
- URP is ideal for projects that need to **balance performance with visual quality**, making it a popular choice for both 2D and 3D games.

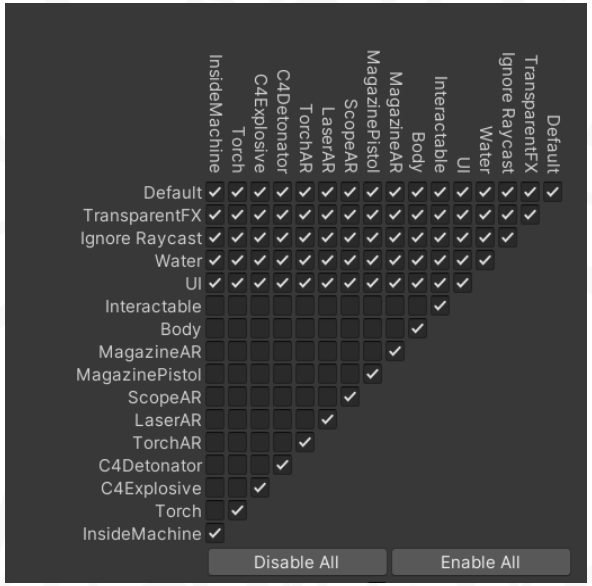
UNIVERSITÀ
DEGLI STUDI
FIRENZE

Game Physics

Layer Collision and Collision detection

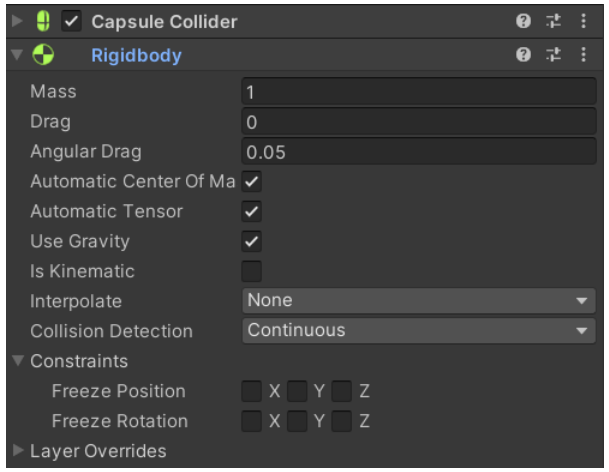
Layer Collision

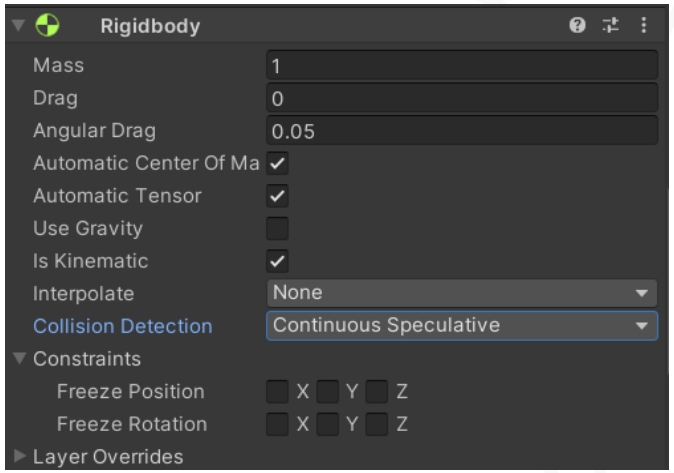
- To prevent issues, we disabled collisions between the various layers, particularly between the body and the layers.
- This is because, when collisions were enabled, objects would sometimes apply forces to others, causing issues, particularly with various sockets, as we will see later.
- To make everything more VR-friendly, collisions with the body must obviously be disabled.



Collision Target Detection

- The collisions registered for the fired bullets rigidbody are set to "**continuous**" to improve accuracy in detecting collisions.






```
private void OnCollisionEnter(Collision other) {
    if (!_isDisabled && other.gameObject.CompareTag("Bullet")){
        Destroy(other.gameObject);
        TargetDestroyEffect();
        ToggleTarget();
        Registry.TargetDestroy();
    }
}

1 reference
private void TargetDestroyEffect() {
    var random = UnityEngine.Random.Range(0.8f, 1.2f);
    _audioSource.pitch = random;
    _audioSource.Play();
    _particleSystem.Play();
}

1 reference
private void ToggleTarget() {
    _meshRenderer.enabled = _isDisabled;
    _boxCollider.enabled = _isDisabled;
    _isDisabled = !_isDisabled;
}
```

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Player Movement

Different Type

UNIVERSITÀ DEGLI STUDI FIRENZE

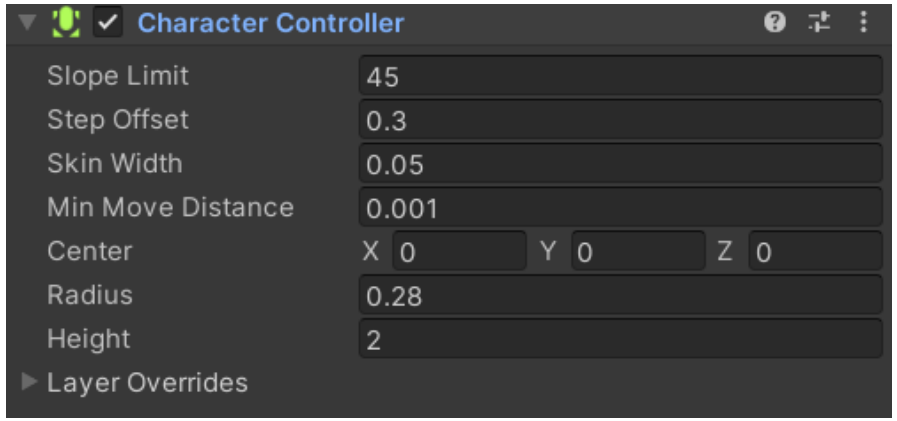
Different types of player movement

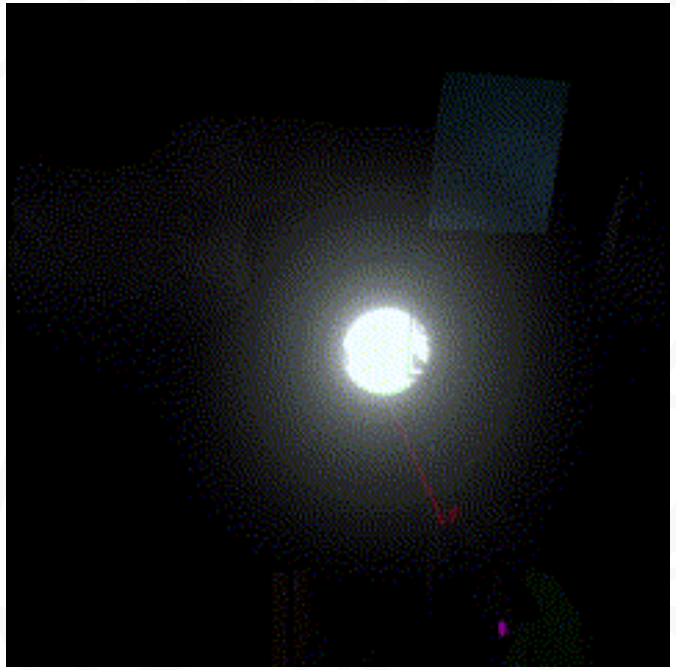
- There are mainly three ways to implement player movement in VR:
  1. **Snap Turn**: Rotate the view by *n* degrees with each input.
  2. **Continuos Turn**: Continuously rotate the view smoothly according to the player's input.
  3. **Teleport**: Allow the player to choose the point to teleport to and the direction to face.

- We chose the second option because, based on our testing, it proved to be the best for our type of game and caused less nausea when using the headset.

Stairs climb system

- In the **CharacterController**, we set a slope limit of 45 degrees and a maximum step offset. This allows our player to climb over objects in the scene that are up to 0.3 units high.





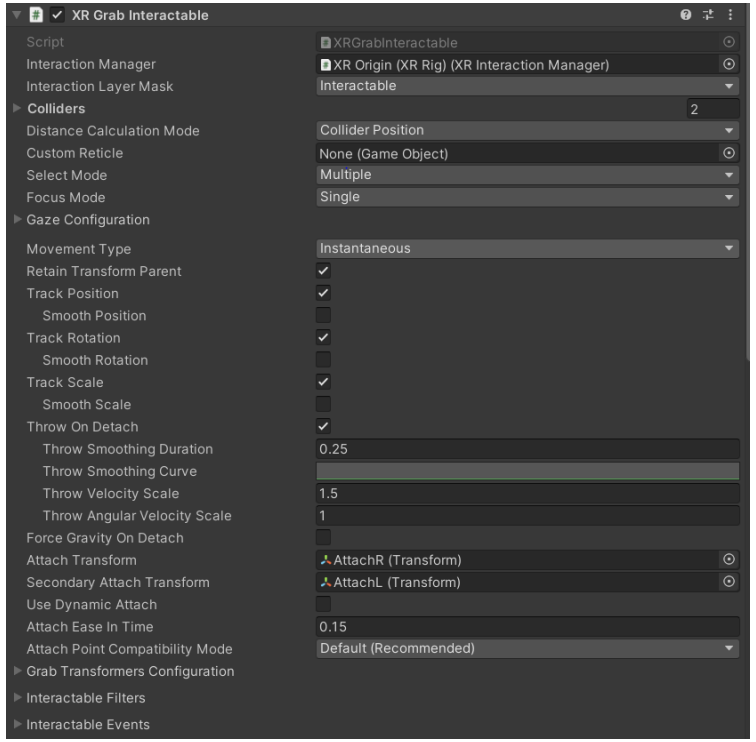
UNIVERSITÀ
DEGLI STUDI
FIRENZE

Grab Interactor

Torch and the Guns

Grab Interactor

- The Grab Interactor This is used for interacting with objects at close range by directly grabbing them with hand controllers.



Grab Interactor

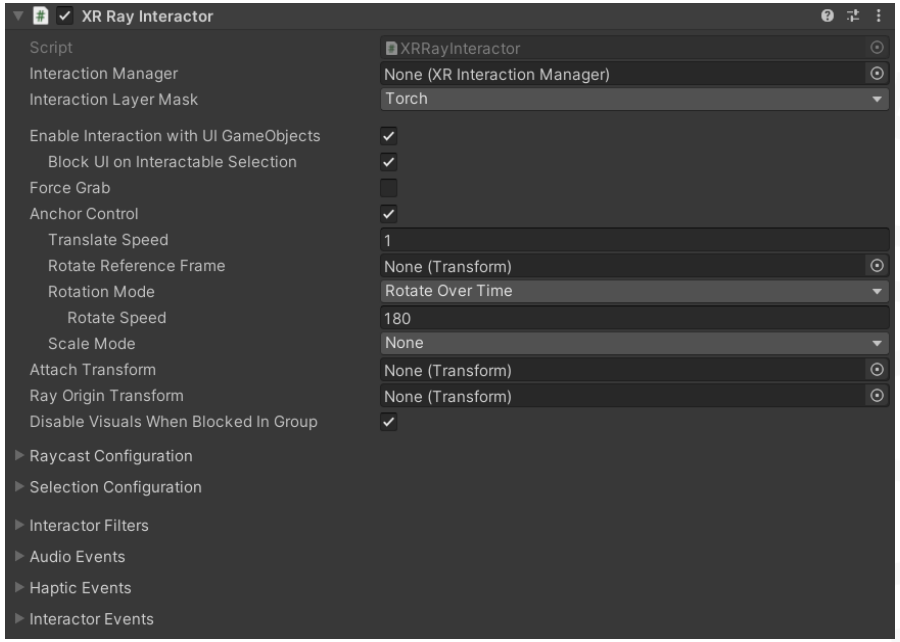
```
private void OnSelectEntered(SelectEnterEventArgs args){
    if (args.interactorObject is XRBaseInteractor interactor){
        if (interactor.CompareTag("LeftHandFake")){
            isLeftHandAttached = true;
            leftHandInteractor = interactor;
        }
        else if (interactor.CompareTag("RightHand")){
            isRightHandAttached = true;
            rightHandInteractor = interactor;
        }
    }
}

private void OnSelectExited(SelectExitEventArgs args){
    if (args.interactorObject is XRBaseInteractor interactor){
        if (interactor == leftHandInteractor){
            isLeftHandAttached = false;
            leftHandInteractor = null;
        }
        else if (interactor == rightHandInteractor){
            isRightHandAttached = false;
            rightHandInteractor = null;
        }
    }
}
```

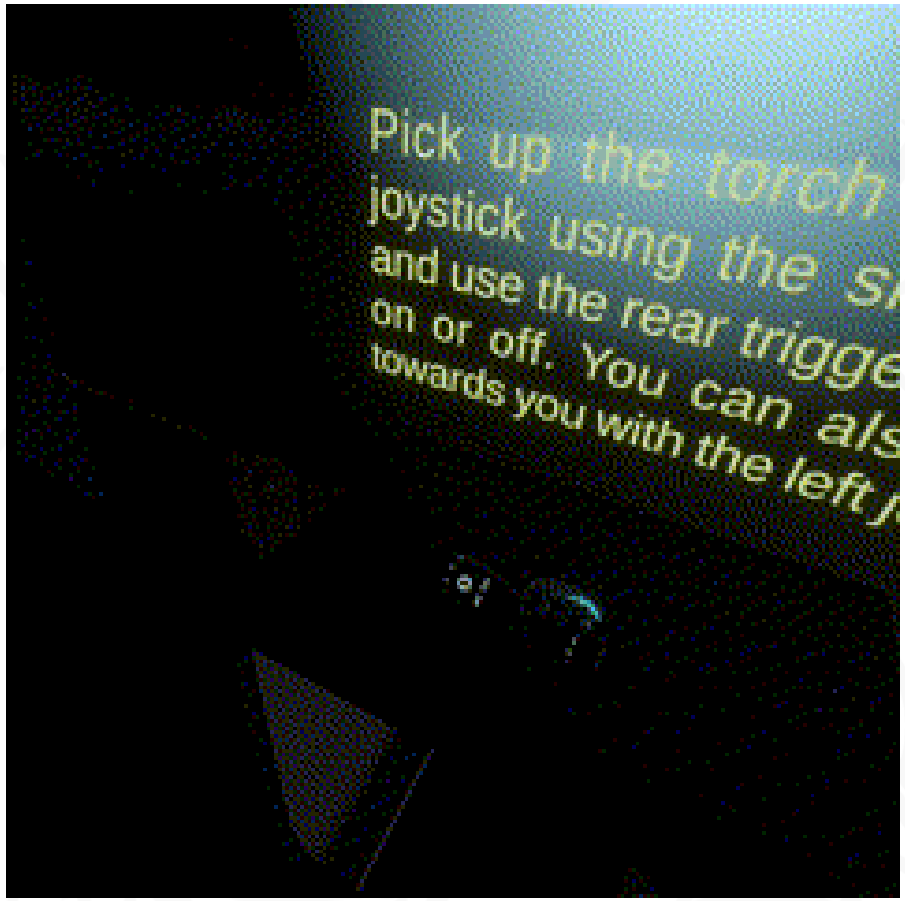
```
void Update(){
    Debug.Log(InputDevices.GetDeviceAtXRNode(XRNode.LeftHand));
    timeElapsedX += Time.deltaTime;
    if (timeElapsedX >= delayX)
        wasXButtonPressed = false;

    timeElapsedY += Time.deltaTime;

    if (timeElapsedY >= delayY)
        wasYButtonPressed = false;
    bool xButtonPressed = false;
    if (laser != null && isLeftHandAttached && isRightHandAttached &&
        InputDevices.GetDeviceAtXRNode(XRNode.LeftHand).TryGetFeatureValue(CommonUsages.primaryButton, out xButtonPressed) &&
        xButtonPressed && !wasXButtonPressed){
        Debug.Log("Toggle Laser");
        laser.le.Toggle();
        wasXButtonPressed = xButtonPressed;
        timeElapsedX = 0f;
    }

    bool yButtonPressed = false;
    if (torch != null && isLeftHandAttached && isRightHandAttached &&
        InputDevices.GetDeviceAtXRNode(XRNode.LeftHand).TryGetFeatureValue(CommonUsages.secondaryButton, out yButtonPressed) &&
        yButtonPressed && !wasYButtonPressed){
        Debug.Log("Toggle Torch");
        torch.te.Toggle();
        wasYButtonPressed = yButtonPressed;
        timeElapsedY = 0f;
    }
}
```

- This weapon, in particular, features a different grab interactor from the standard ones, configured for a **two-handed grip**, with two Attach Transformers, one for each hand.

Ray interactor

- The XR Ray interactor allows for interactions at a distance by projecting a ray from the controller. It’s ideal for selecting or interacting with objects without needing to be physically close.





Using Direct and Ray interactor together

- In *Quantum Key Hunt*, we implemented a grab system similar to *Half-Life: Alyx* on the left hand, allowing players to grab objects both directly and from a distance. Using Unity's **XRRayInteractor**, players can pull distant objects toward them with a physics-based throw interaction.

```
private void Update()
{
    if (isSelected && firstInteractorSelecting is XRRayInteractor)
    {
        Vector3 velocity = (rayInteractor.transform.position - previousPos) / Time.deltaTime;
        previousPos = rayInteractor.transform.position;
        if (velocity.magnitude > velocityThreshold)
        {
            Drop();
            interactableRigidbody.velocity = ComputeVelocity();
            canJump = false;
        }
    }
}
```

Using Direct and Ray interactor together

- The system calculates the velocity of the grabbed object, determining when to "drop" and send the object flying based on the player's hand movement. The interaction combines **Direct Interactor** for close objects and **Ray Interactor** for distant grabs.

```
public Vector3 ComputeVelocity()
{
    Vector3 diff = rayInteractor.transform.position - transform.position;
    Vector3 diffXZ = new Vector3(diff.x, 0, diff.z);
    float diffXZLength = diffXZ.magnitude;
    float diffYLength = diff.y;

    float angleInRadian = Mathf.Clamp(diff.normalized.y * 90, jumpAngleDegree, 90) * Mathf.Deg2Rad;

    float jumpSpeed = Mathf.Sqrt(-Physics.gravity.y * Mathf.Pow(diffXZLength, 2)/
     (2 * Mathf.Cos(angleInRadian)*Mathf.Cos(angleInRadian)*(diffXZ.magnitude * Mathf.Tan(angleInRadian) - diffYLength)));

    // jumpSpeed = jumpSpeed * 0.5f;

    Vector3 jumpVelocityVector = diffXZ.normalized * Mathf.Cos(angleInRadian) * jumpSpeed + Vector3.up * Mathf.Sin(angleInRadian) * jumpSpeed;

    return jumpVelocityVector;
}
```

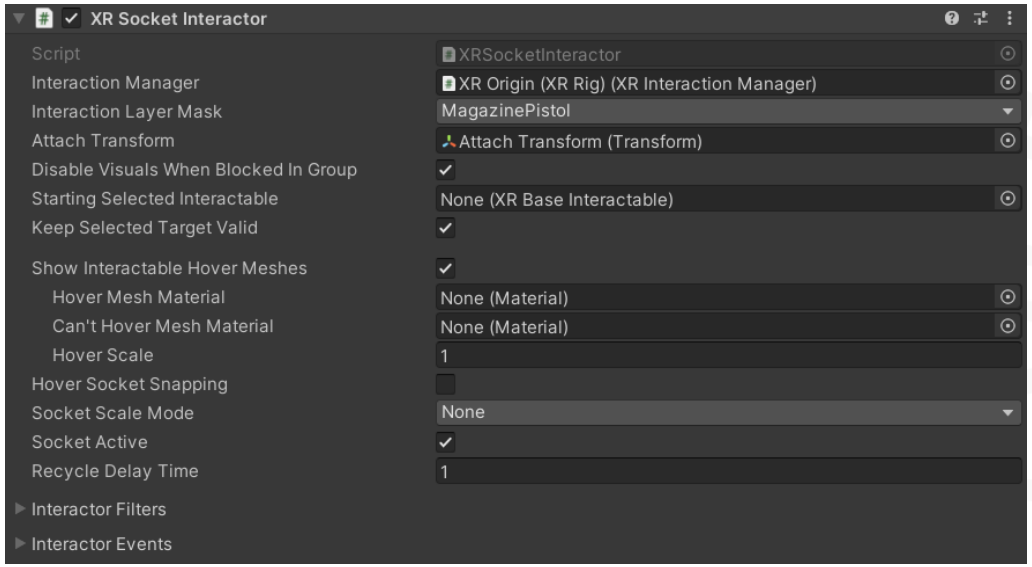
UNIVERSITÀ
DEGLI STUDI
FIRENZE

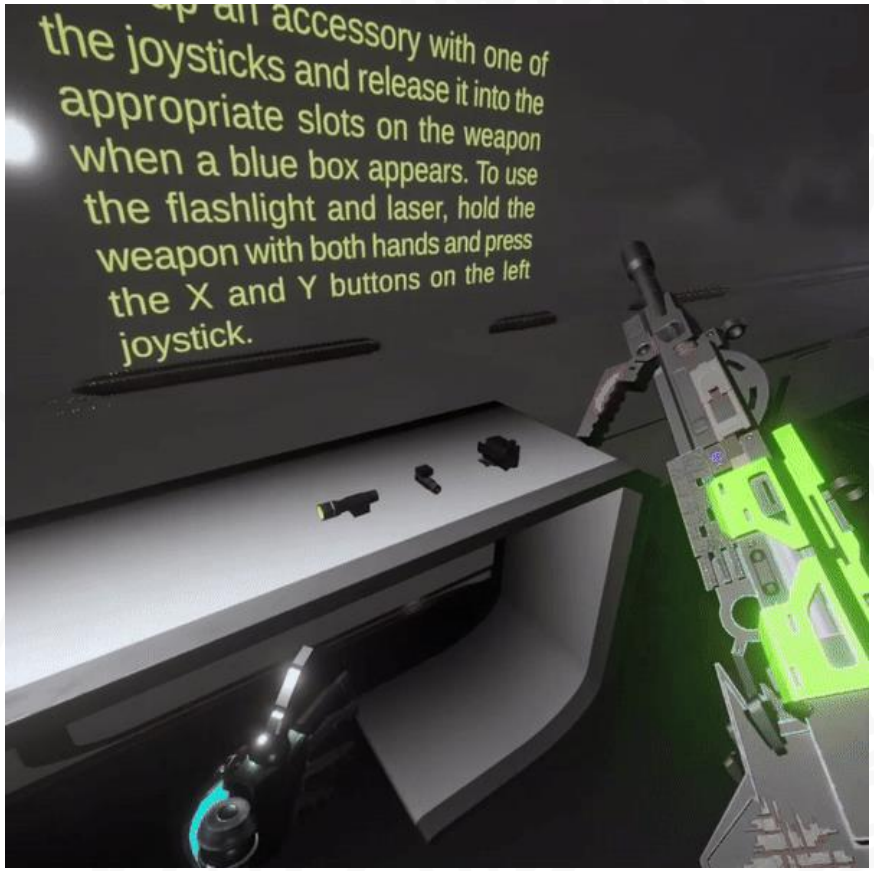
Scoket Interactor

Magazine and attachments

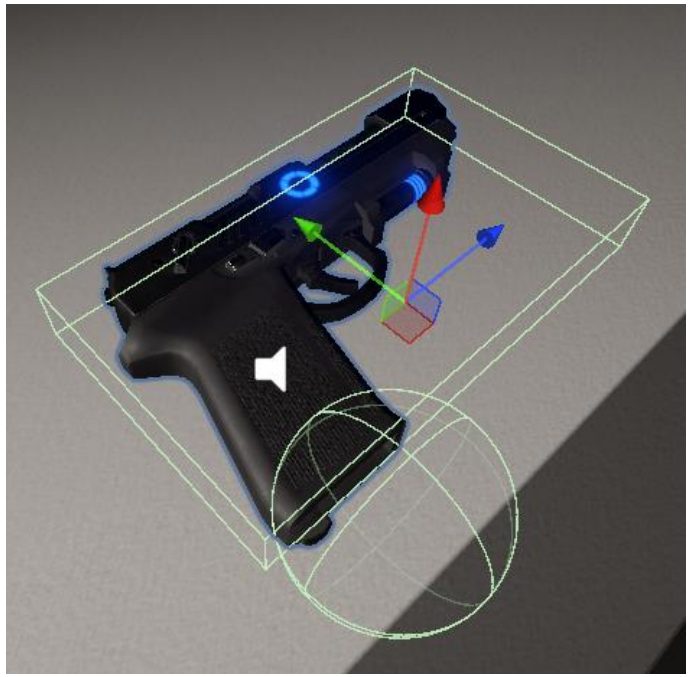
Socket Interactor

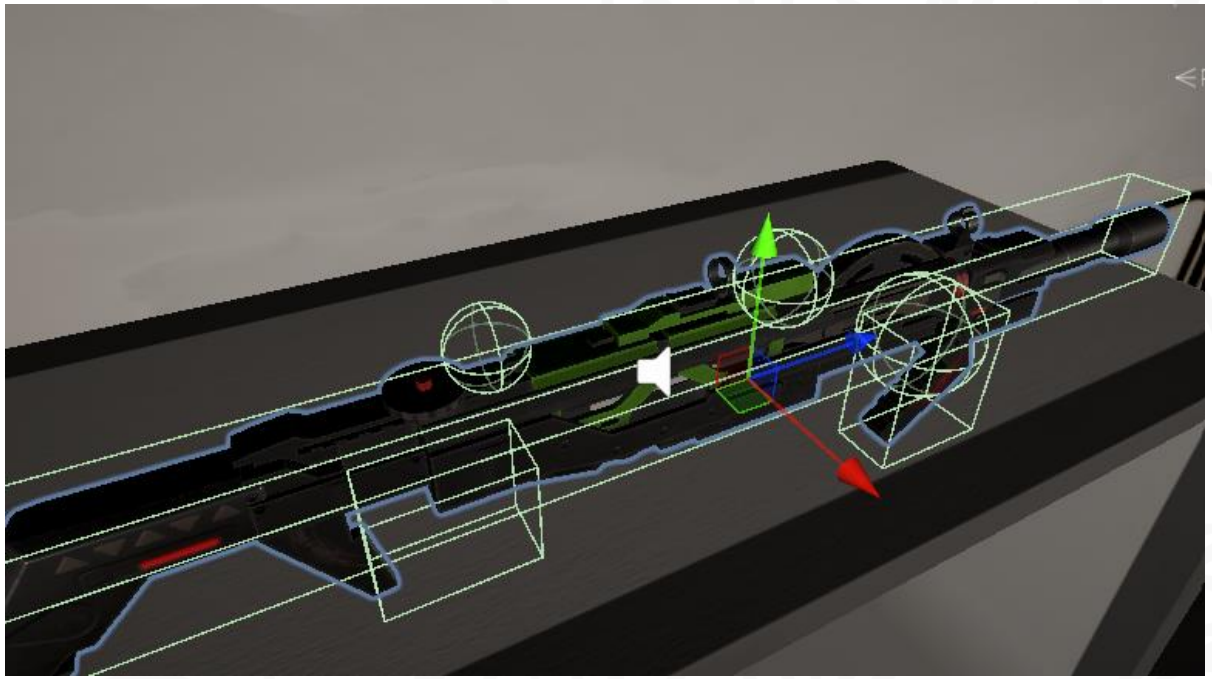
- The XR Socket Interactor enables snapping objects into predefined slots, allowing for object placement or assembly mechanics in a 3D space.





Socket Colliders model

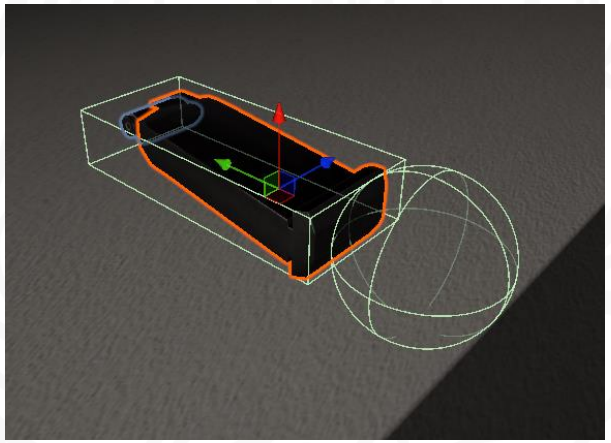




Socket Interactor

- We implemented **XRSocketInteractors** to allow players to seamlessly attach and detach magazines and attachments to guns. The socket mechanism ensures that objects like magazines are snapped into place, enabling smooth interactions during reloading or modifying the weapon.
- The **MagazineSocket** script listens for objects entering or exiting the socket, attaching the magazine to the weapon and enabling it for use once it's placed in the correct slot.

```
private void OnMagazineAttached(SelectEnterEventArgs args)
{
    Magazine magazine = args.interactableObject.transform.GetComponent<Magazine>();
    if (magazine != null)
    {
        magazine.SetCollider(true);
        weapon.AttachMagazine(magazine);
    }
}

private void OnMagazineDetached(SelectExitEventArgs args)
{
    Magazine magazine = args.interactableObject.transform.GetComponent<Magazine>();
    if (magazine != null)
    {
        magazine.SetCollider(false);
        weapon.DetachMagazine();
    }
}
```



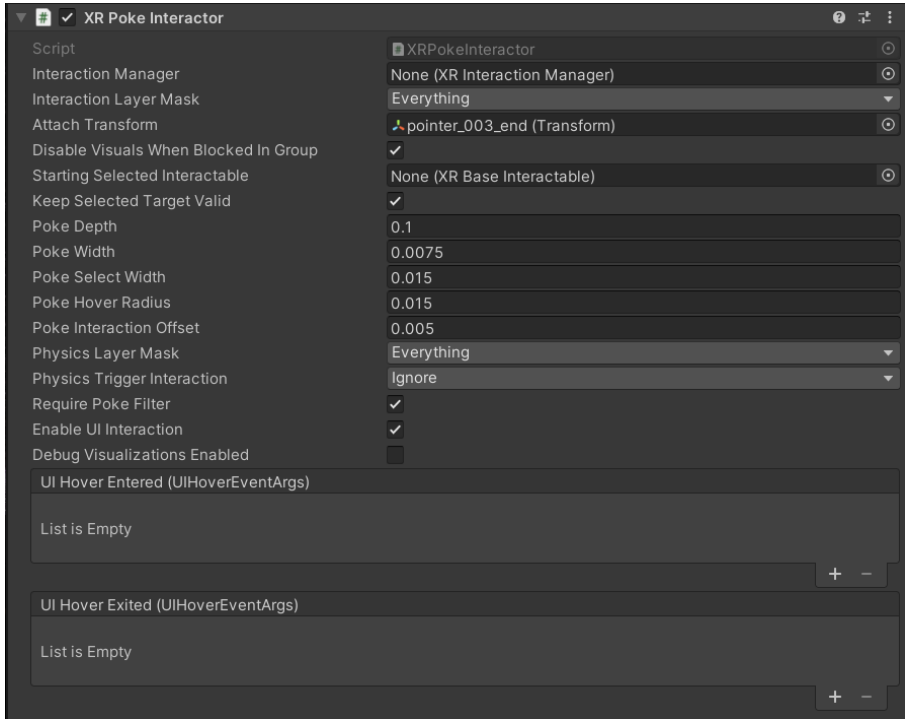
UNIVERSITÀ
DEGLI STUDI
FIRENZE

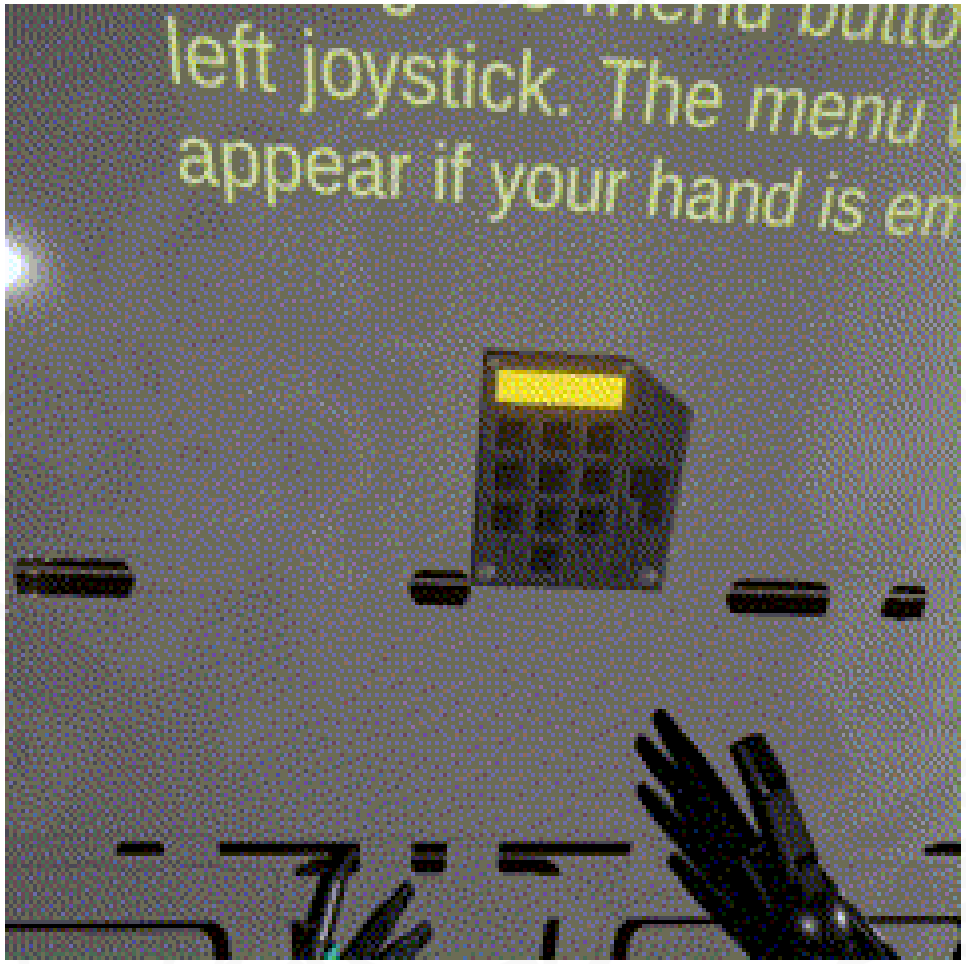
Poke Interactor

Key-Pad

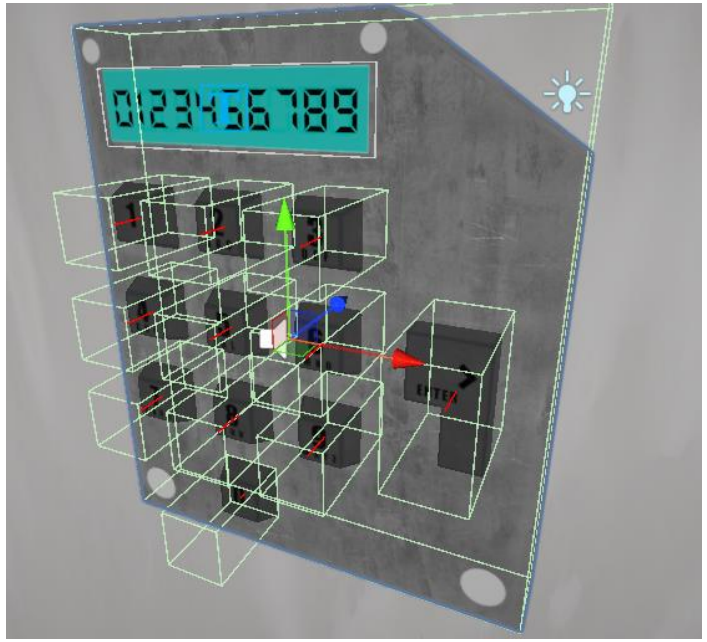
Poke Interactor

- The XR Poke Interactor is designed for touch-based interactions, it lets players "poke" objects, simulating real-world tapping or button-press actions.





PinPad Model



- For the pinpad, we modeled **each key with its own BoxCollider** to allow it to be pressed by the player's right index finger.
- This can sometimes lead to missed collisions between the finger and the key. However, adjusting the poke interactor settings can improve the situation.

UNIVERSITÀ
DEGLI STUDI
FIRENZE

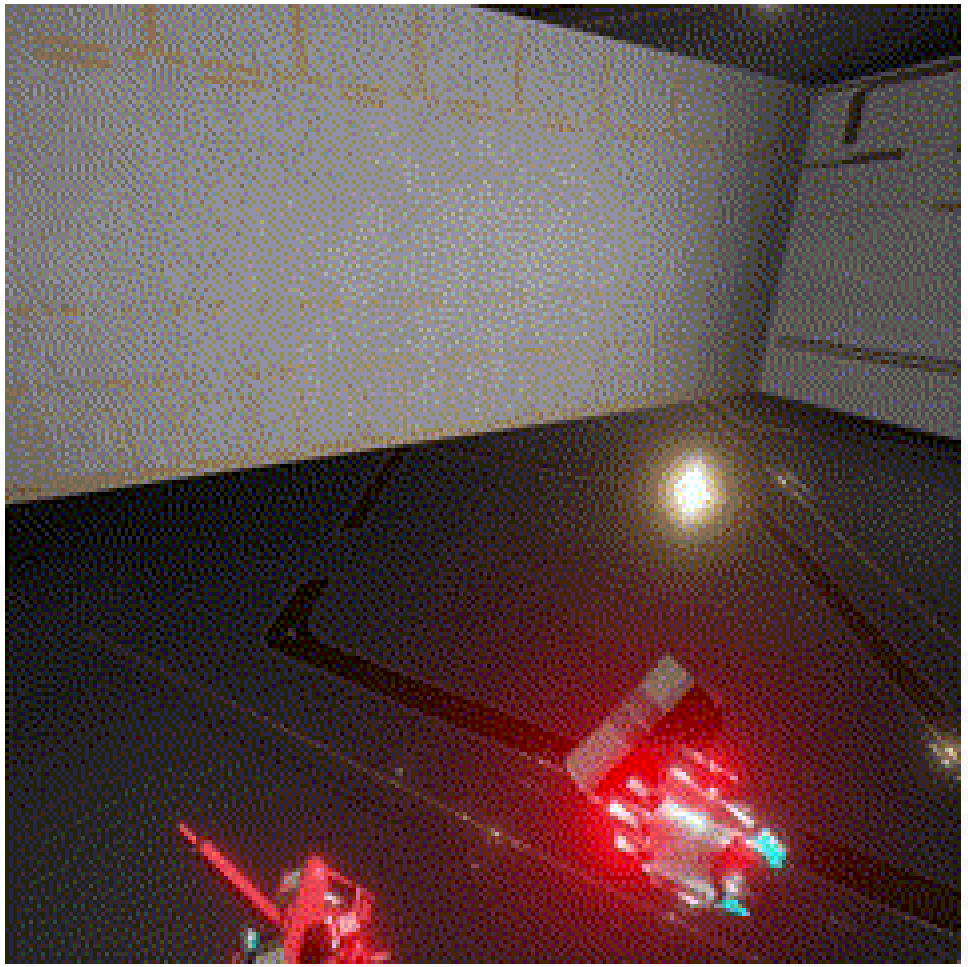
Destroyable Wall

C4 system

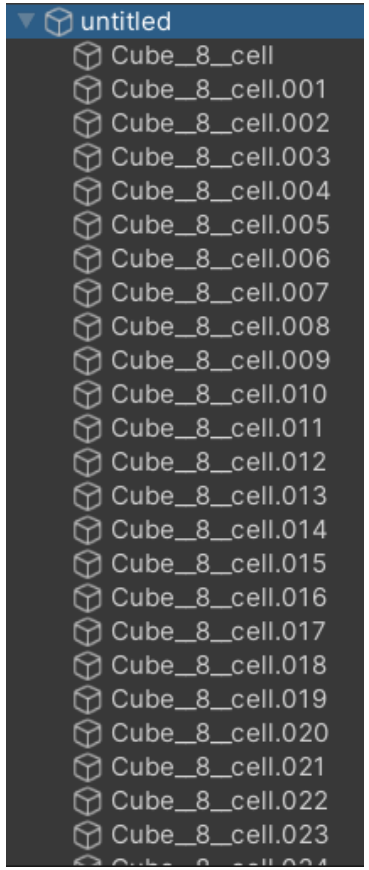
UNIVERSITÀ DEGLI STUDI FIRENZE

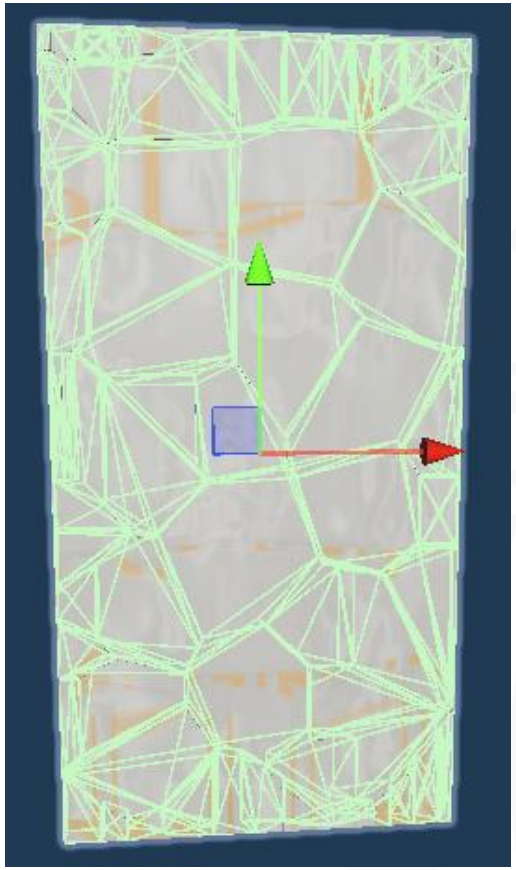
Destroyable Wall: Introduction

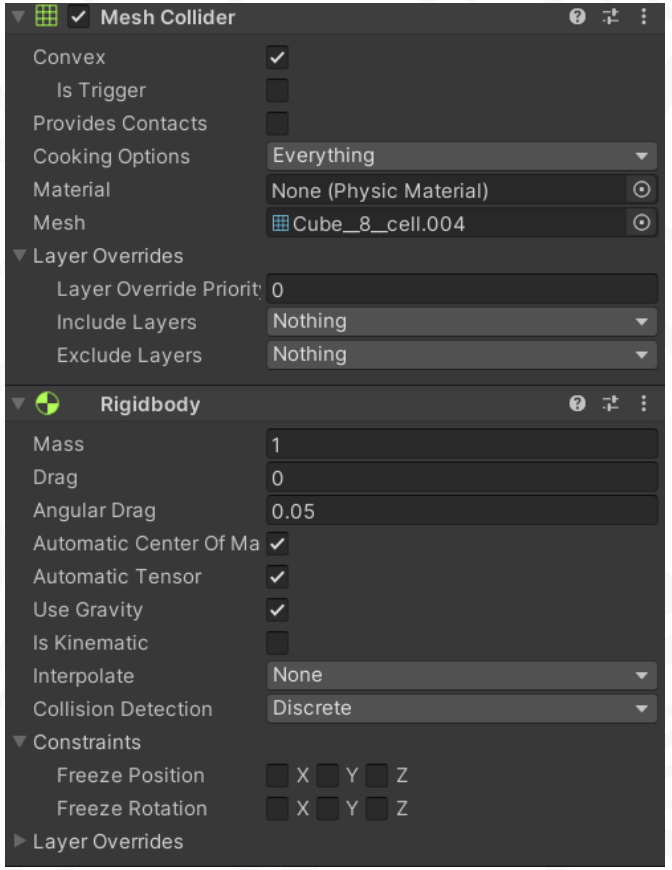
- In our game, there is an explosion mechanic involving C4 and a detonator.
- Only certain walls within the scene are destructible.



Destroyable Wall: Brick System







Destroyable Wall: Explosion System

```
public void Explode()
{
    Collider[] colliders = Physics.OverlapSphere(transform.position,radius);
    foreach (Collider near in colliders){
        ExplodeWall explodeWall = near.GetComponent<ExplodeWall>();
        if (explodeWall != null)
            explodeWall.BreakWall(explosionForce,transform,radius);
    }
    Instantiate(explosionEffect,transform.position,transform.rotation);
    Destroy(gameObject);
}
```

```
public void BreakWall(float explosionForce, Transform x, float radius){
    GameObject fract = Instantiate(fractured,transform.position,transform.rotation);
    foreach(Rigidbody rb in fract.GetComponentsInChildren<Rigidbody>()){
        rb.AddExplosionForce(explosionForce,x.position, radius, 1f, ForceMode.Impulse);
    }
    Destroy(gameObject);
}
```

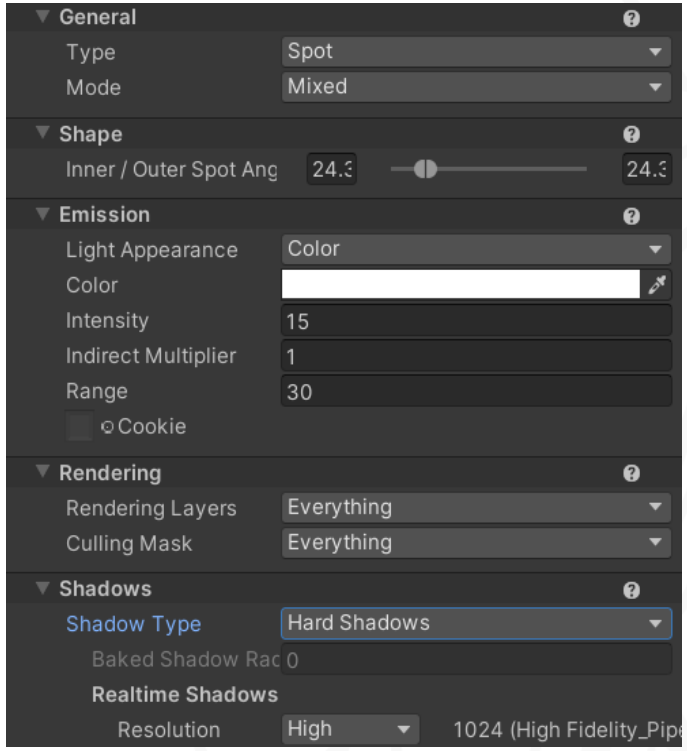
- The first function pertains to the C4 and is called by the detonator when activated.
- The second function is specific to the destructible wall.

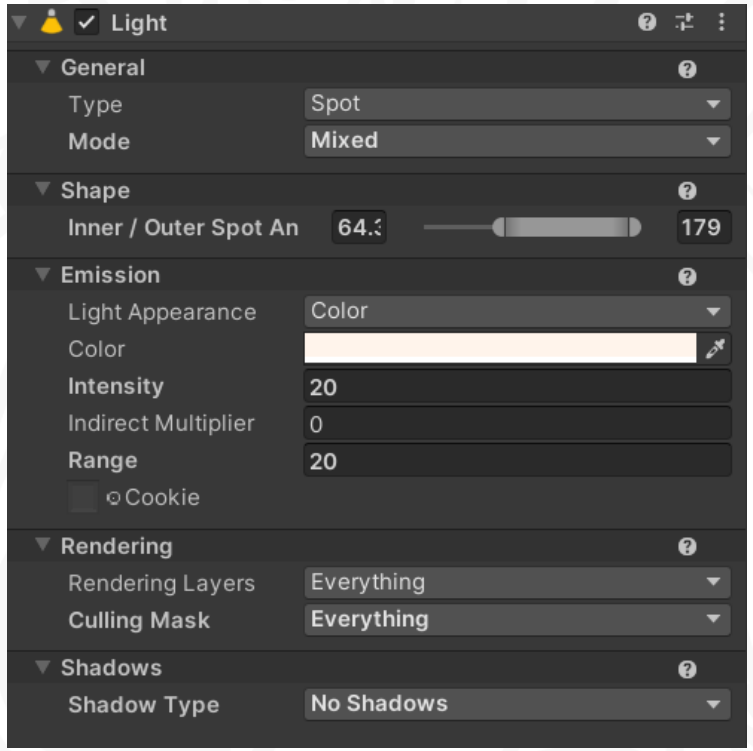
UNIVERSITÀ DEGLI STUDI FIRENZE

Light System

Light System

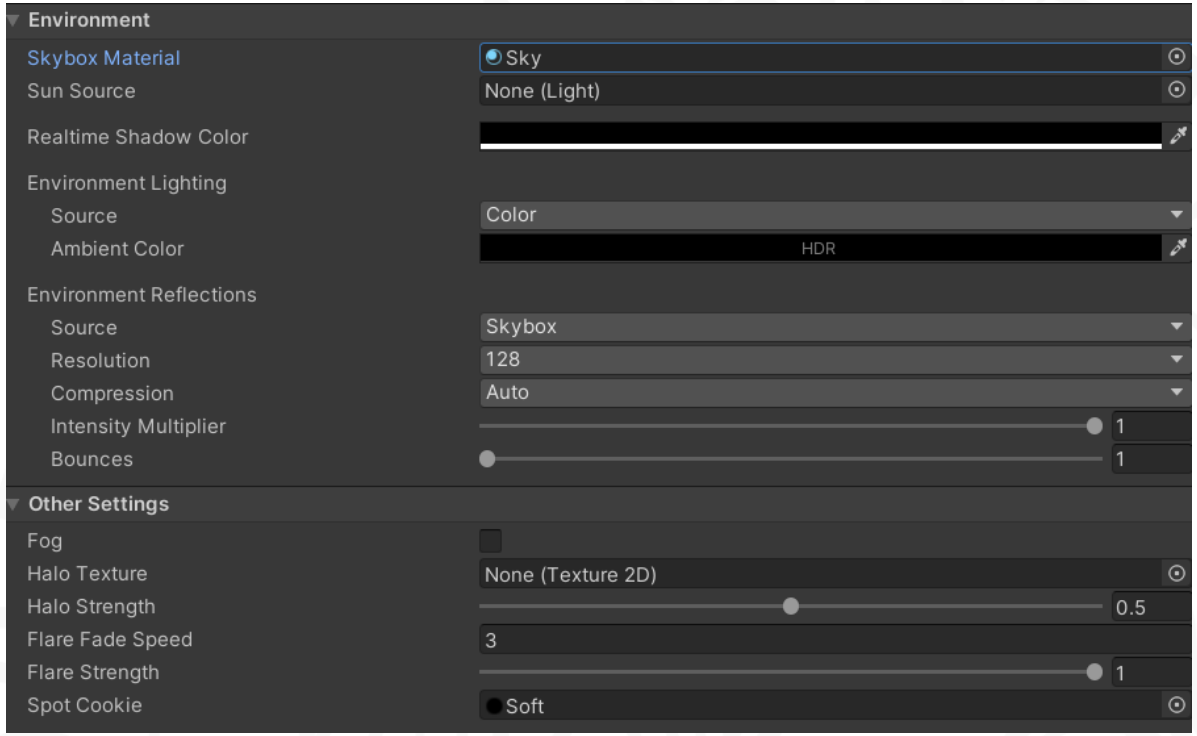
- In the game, we have two types of lighting: **static lights** within the scene and **real-time lights**, such as torches, which can move.
- Torch lights are the only ones casting **shadows**, due to performance considerations.





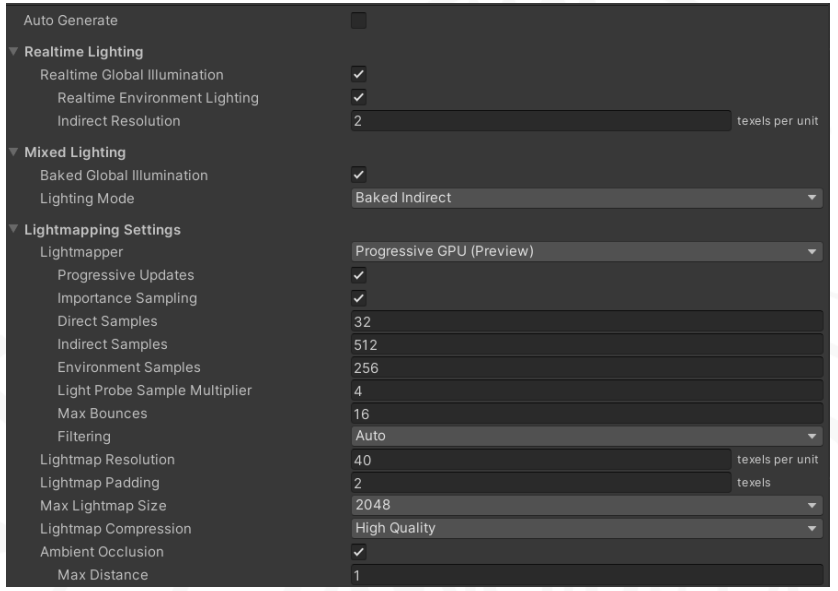
Environment Settings

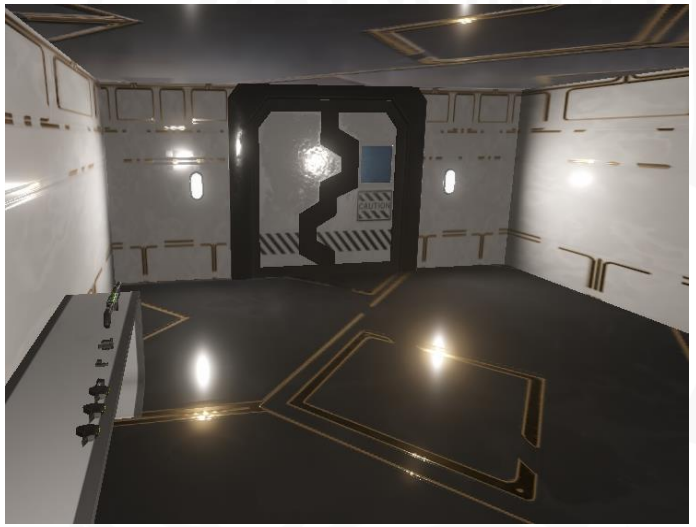
- For our project, since we are always rendering **indoor rooms**, we set the shadow color to black and the ambient color to black as well, making it effectively inconsequential.
- Additionally, we **disabled all external lights**, such as the sun.



Light, global settings

- We used these settings for all the scenes.
- We set the maximum number of **light bounces to 16** because, experimentally, this resulted in more aesthetically pleasing shadows and colors.
- We also increased the samples for indirect lighting.





UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusions

UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusions

- In conclusion, we aimed to create a game that is as comprehensive as possible, covering all aspects and incorporating the standard features expected in modern VR games.
- Unlike a traditional 3D game, VR introduces unique challenges when dealing with elements that aren't **VR-friendly**. For instance, an HUD with text (like a canvas that follows you) can be problematic, or even the placement of objects in the scene, which may sometimes be physically out of reach.
- Additionally, in VR, every small **graphical detail**, such as lighting and shadows, becomes much more noticeable. Any imperfections in these elements will be amplified by the immersive nature of VR, making flaws more apparent to the player.
- Another crucial aspect we've focused on is **performance**. In VR, dropping below 60 FPS can lead to motion sickness, so maintaining a high frame rate is essential to ensure a comfortable experience for the player.

UNIVERSITÀ DEGLI STUDI FIRENZE

Thank you for your attention!

Quantum Key Hunt VR

Mirko Bicchierai and Niccolò Marini

Course: Computer Graphics.