DESENVOLUPAMENT DE JOCS 3D

Práctica

Práctica 2 - https://youtu.be/MRsd0ptJxV8

Creando nivel – Prototyping
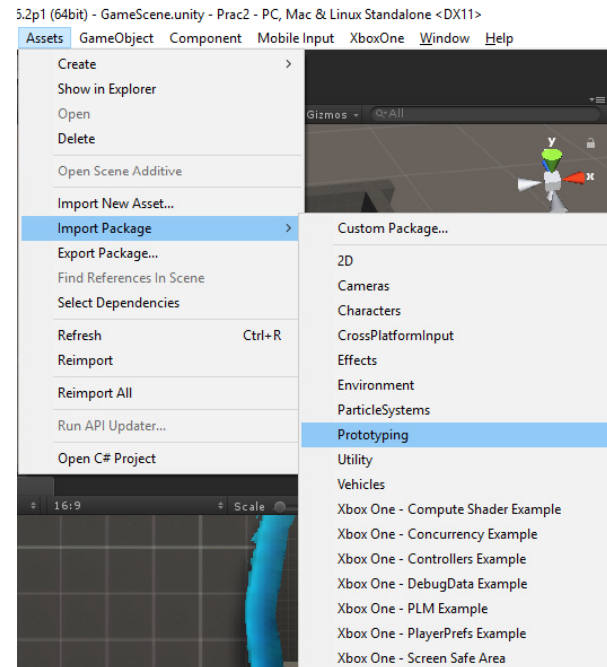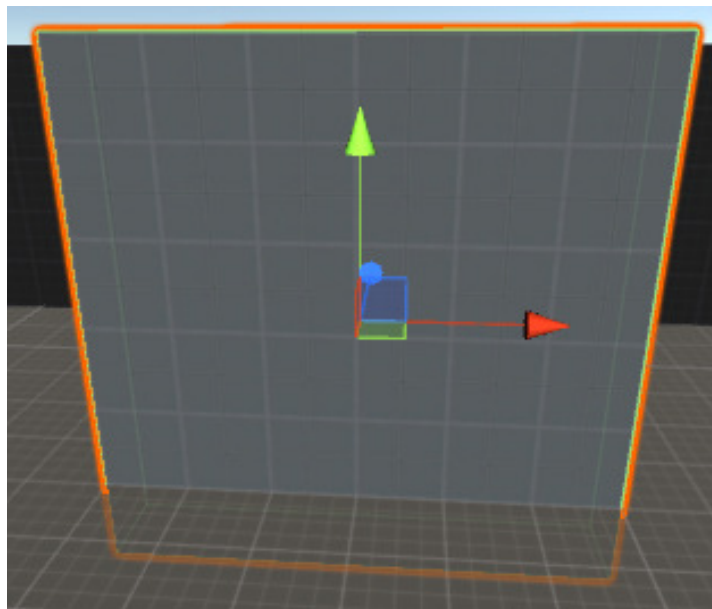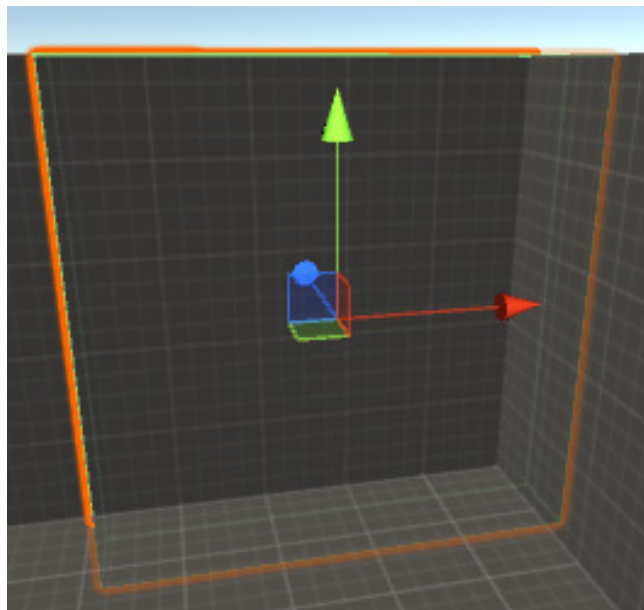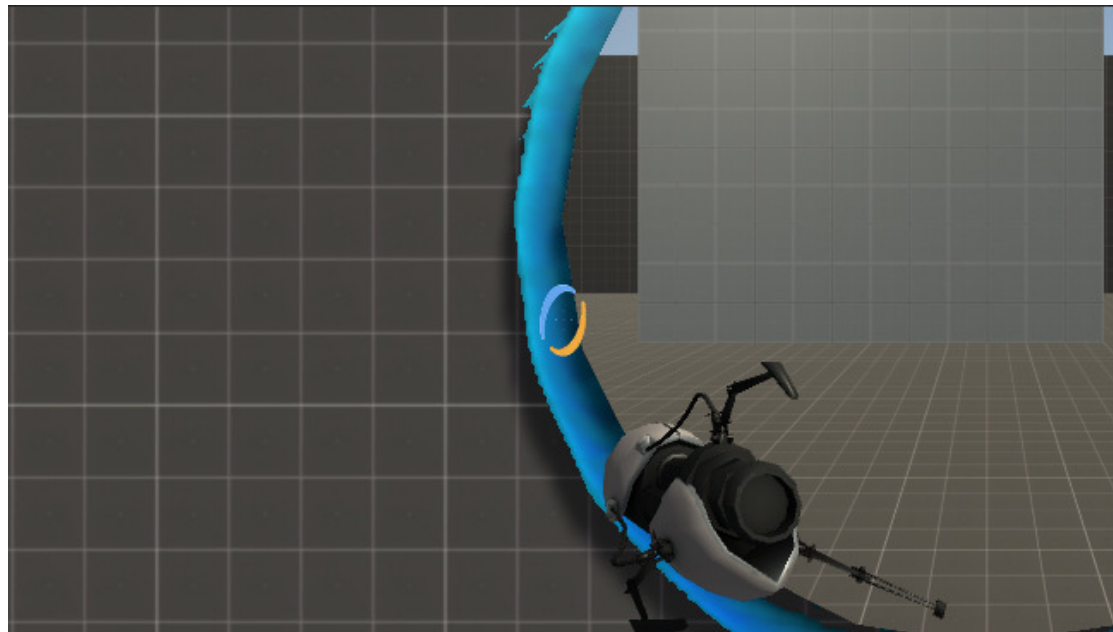
Creando nivel

Creando nivel – Pared pintable
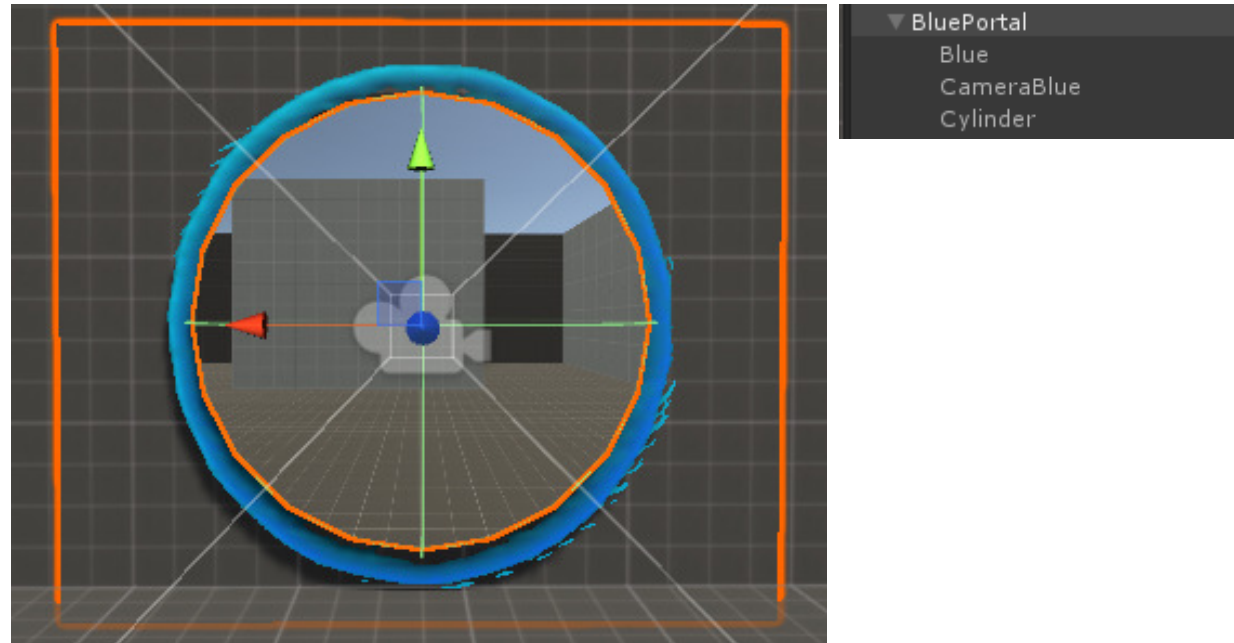
Creando nivel – Pared no pintable

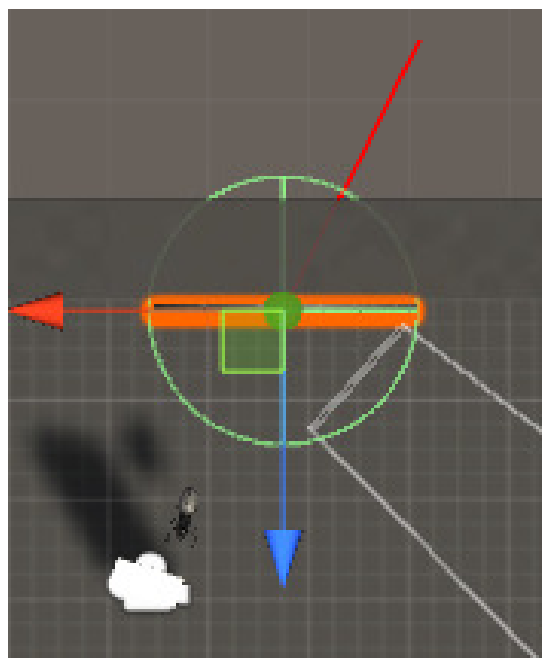# FPSPlayerController

Portales

Portales
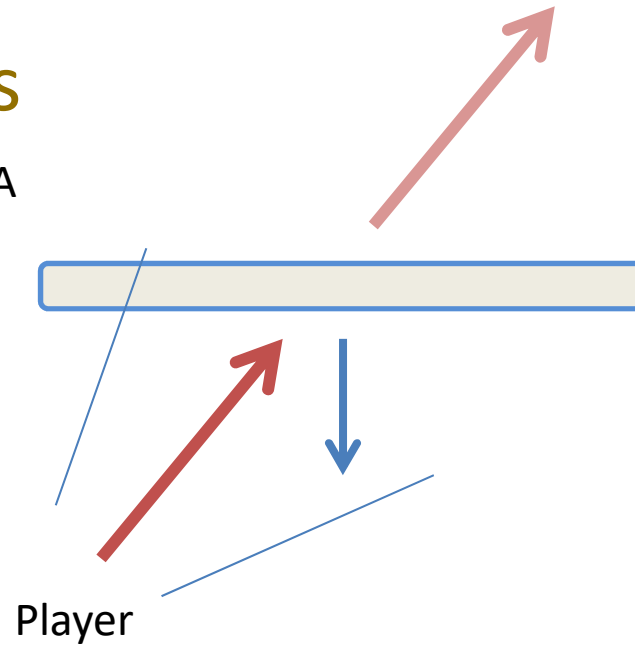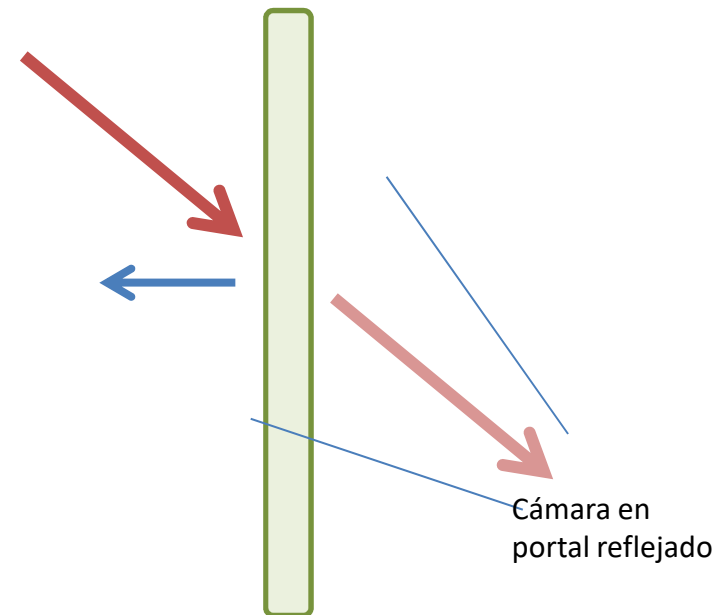
Portales

Portal A

Portal B

Player

Cámara en
portal reflejado

Portales – Implementación

```
public Transform m_PlayerCamera;
public Portal m_MirrorPortal;
public Camera m_PortalCamera;
public float m_MinFOV=8.0f;
public float m_MaxFOVDistance=20.0f;
public float m_NearClipOffset=0.5f;
```

Portales – Implementación

```
void Update ()
{
    Vector3 l_ReflectedVector=m_PlayerCamera.position-transform.position;
    l_ReflectedVector=transform.position-l_ReflectedVector;
    Debug.DrawLine(transform.position, l_ReflectedVector, Color.red);

    l_ReflectedVector=transform.InverseTransformPoint(l_ReflectedVector);
    m_MirrorPortal.m_PortalCamera.transform.position = m_MirrorPortal.transform.TransformPoint(l_ReflectedVector);
    m_PortalCamera.transform.forward=(transform.position-m_PortalCamera.transform.position).normalized;
    m_PortalCamera.nearClipPlane = Vector3.Distance(m_PortalCamera.transform.position ,
    this.transform.position)+m_NearClipOffset;
}
```
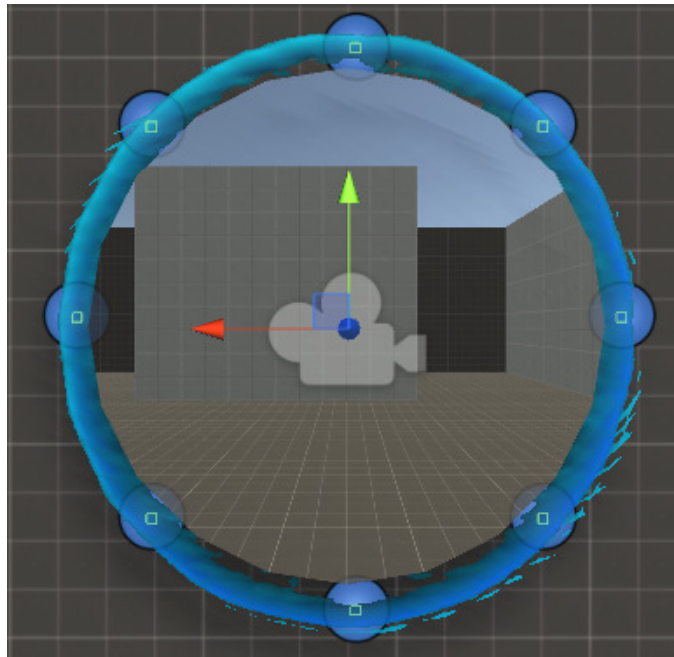
Portales – Implementación

```
void Update ()
{
    //FOV
    Vector3 l_PlayerToPortal=transform.position-m_PlayerCamera.position;
    float l_Distance=l_PlayerToPortal.magnitude;
    float l_Pct=1.0f-Mathf.Min(l_Distance/m_MaxFOVDistance, 1.0f);
    m_MirrorPortal.m_PortalCamera.fieldOfView=Mathf.Lerp(m_MinFOV, 60.0f, l_Pct);
}
```

Portales – IsValidPosition

Portales – IsValidPosition – Implementación

```
public List<Transform> m_ValidPoints;

public bool IsValidPosition()
{
    Vector3 l_Normal=Vector3.zero;
    for(int i=0; i<m_ValidPoints.Count; ++i)
    {
        Transform l_ValidPoint=m_ValidPoints[i];
        Ray l_Ray=new Ray(m_PlayerCamera.position, l_ValidPoint.position-m_PlayerCamera.position);

        RaycastHit l_RaycastHit;
        if(Physics.Raycast(l_Ray, out l_RaycastHit))
        {
            //Por tag, normal y distancia coincide en todos los puntos
        }
    }
}
```

Portales – Teleport

```
public void Teleport(Portal _Portal)
{
    Vector3 l_Position=_Portal.transform.InverseTransformPoint(transform.position);
    transform.position=_Portal.m_MirrorPortal.transform.TransformPoint(l_Position);
    Vector3 l_Direction=_Portal.transform.InverseTransformDirection(-transform.forward);
    transform.forward=_Portal.m_MirrorPortal.transform.TransformDirection(l_Direction);

    m_Yaw=transform.rotation.eulerAngles.y;
}
```
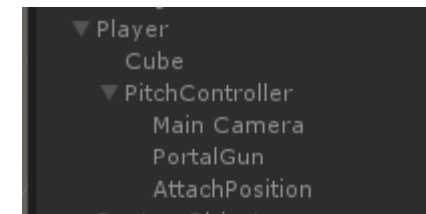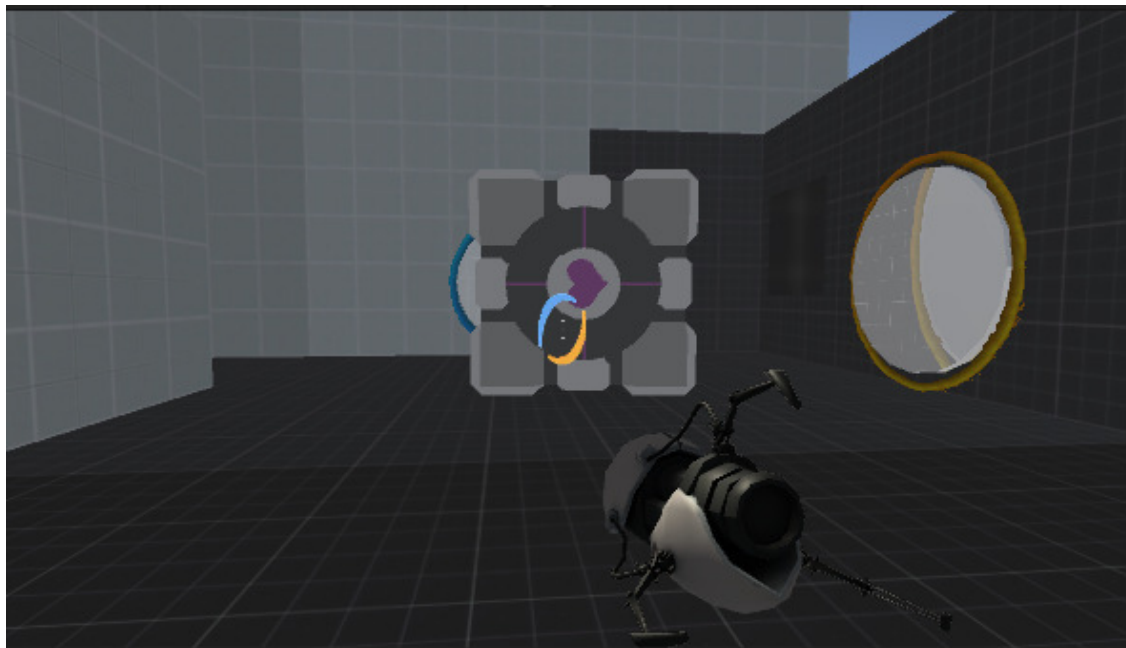
Portales – Implementación

```
void SetPortal(Portal _Portal, Vector3 Position, Vector3 Normal, float Scale)
{
    _Portal.transform.position=Position+Normal*m_SetPortalOffset;
    _Portal.transform.forward=Normal;
    _Portal.transform.localScale=Vector3.one*Scale;
}
```

Portales – AttachObject

AttachObject – Implementación

```
void UpdateAttachedObject()
{
        Vector3 l_EulerAngles=m_AttachingPosition.rotation.eulerAngles;
        if(!m_AttachedObject)
        {
                Vector3 l_Direction=m_AttachingPosition.transform.position-m_ObjectAttached.transform.position;
                float l_Distance=l_Direction.magnitude;
                float l_Movement=m_AttachingObjectSpeed*Time.deltaTime;
                if(l_Movement>=l_Distance)
                {
                        m_AttachedObject=true;
                        m_ObjectAttached.MovePosition(m_AttachingPosition.position);
                        m_ObjectAttached.MoveRotation(Quaternion.Euler(0.0f, l_EulerAngles.y, l_EulerAngles.z));
                }
                else
                {
                        l_Direction/=l_Distance;
                        m_ObjectAttached.MovePosition(m_ObjectAttached.transform.position+l_Direction*l_Movement);
                        m_ObjectAttached.MoveRotation(Quaternion.Lerp(m_AttachingObjectStartRotation, Quaternion.Euler(0.0f, l_EulerAngles.y, l_EulerAngles.z), 1.0f-
Mathf.Min(l_Distance/1.5f, 1.0f)));
                }
        }
        else
        {
                m_ObjectAttached.MoveRotation(Quaternion.Euler(0.0f, l_EulerAngles.y, l_EulerAngles.z));
                m_ObjectAttached.MovePosition(m_AttachingPosition.position);
        }
}
```
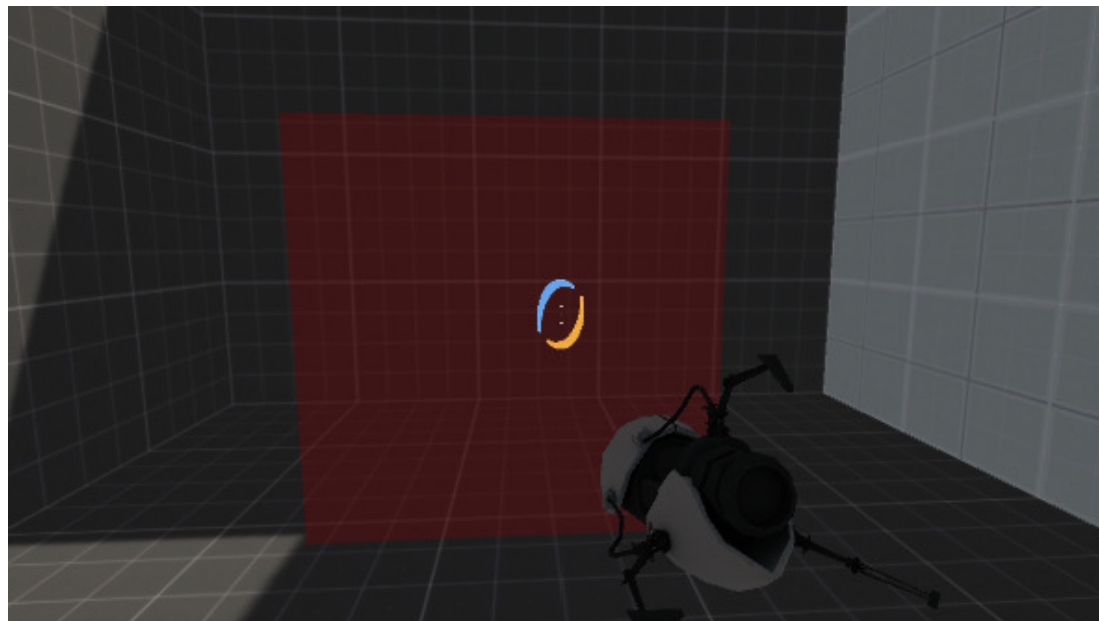
AttachObject – Implementación

```
void DetachObject(float Force)
{
    m_AttachedObject=false;
    m_AttachingObject=false;
    m_ObjectAttached.isKinematic=false;
    m_ObjectAttached.GetComponent<Companion>().SetTeleport(true);
    m_ObjectAttached.AddForce(m_AttachingPosition.forward*Force);
}
```
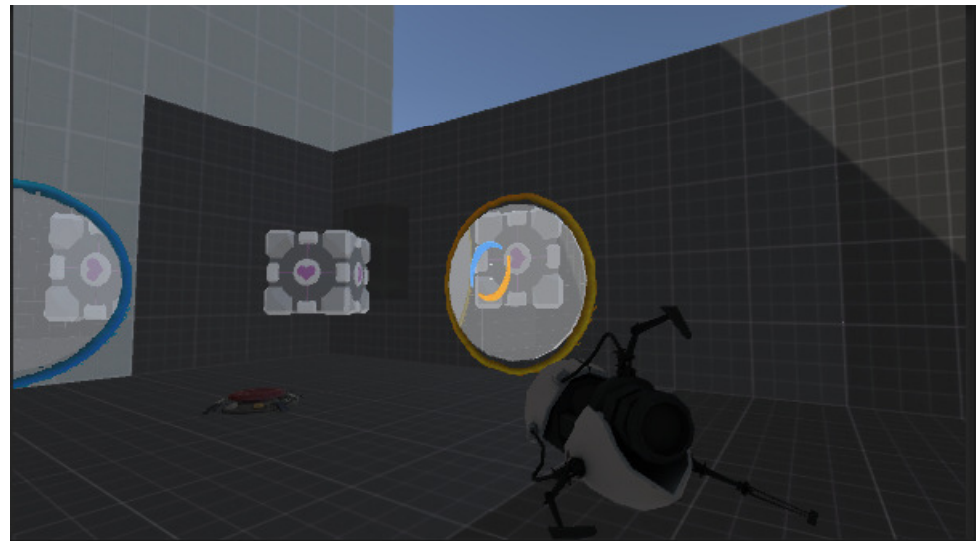
CompanionSpawner

CompanionSpawner – Implementación

```
public class CompanionSpawner : MonoBehaviour
{
    public Transform m_SpawnPosition;
    public GameObject m_CompanionPrefab;

    public void Spawn()
    {
        GameObject.Instantiate(m_CompanionPrefab, m_SpawnPosition.position,
m_SpawnPosition.rotation, null);
    }
}
```

Companion – Teleport
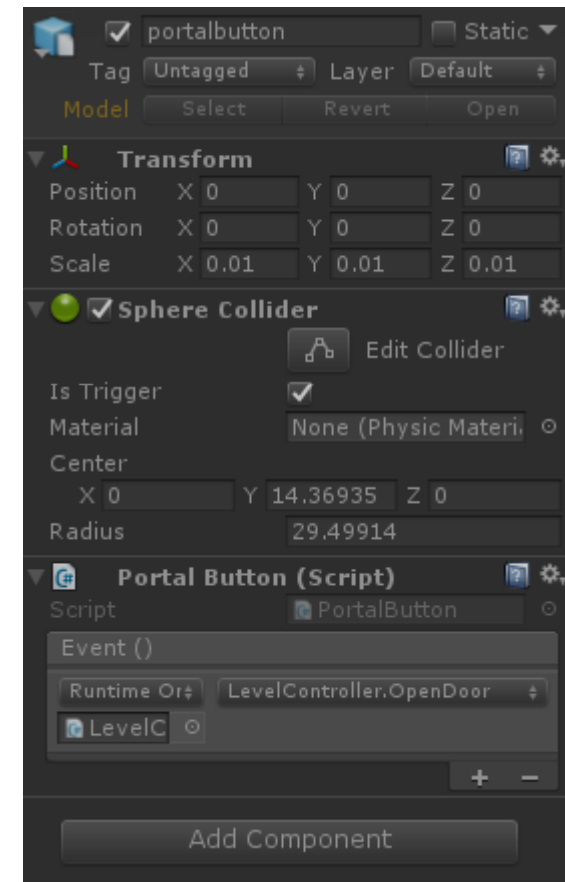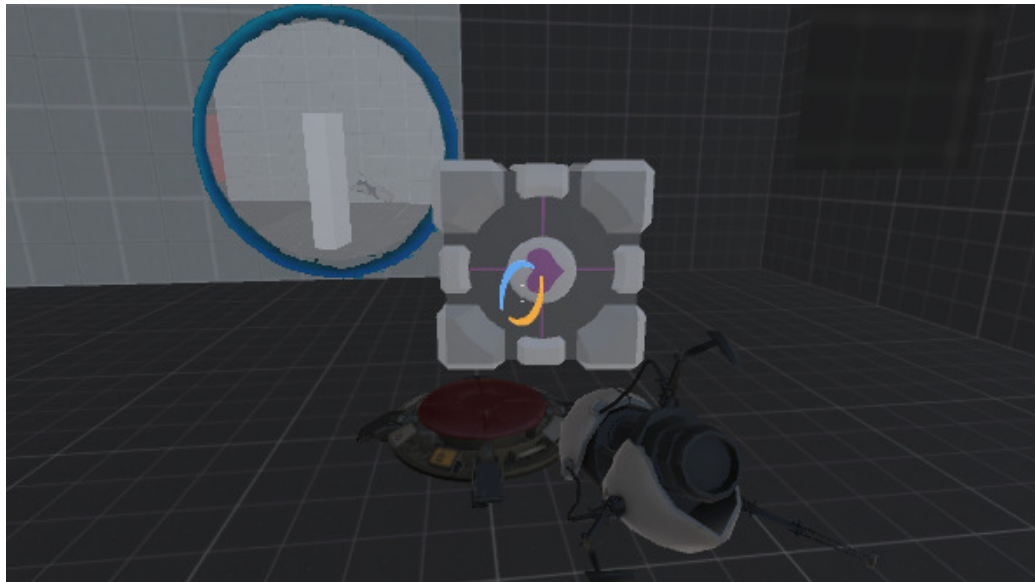
Companion – Implementación

```
public void Teleport(Portal _Portal)
{
    Rigidbody l_Rigidbody=GetComponent<Rigidbody>();
    Vector3 l_Position=_Portal.transform.InverseTransformPoint(transform.position);
    transform.position=_Portal.m_MirrorPortal.transform.TransformPoint(l_Position);
    Vector3 l_Direction=_Portal.transform.InverseTransformDirection(-transform.forward);
    transform.forward=_Portal.m_MirrorPortal.transform.TransformDirection(l_Direction);

    Vector3 l_Velocity=_Portal.transform.InverseTransformDirection(-l_Rigidbody.velocity);
    l_Rigidbody.velocity=_Portal.m_MirrorPortal.transform.TransformDirection(l_Velocity);
    transform.localScale*=(_Portal.m_MirrorPortal.transform.localScale.x/_Portal.transform.loca
    lScale.x);
}
```
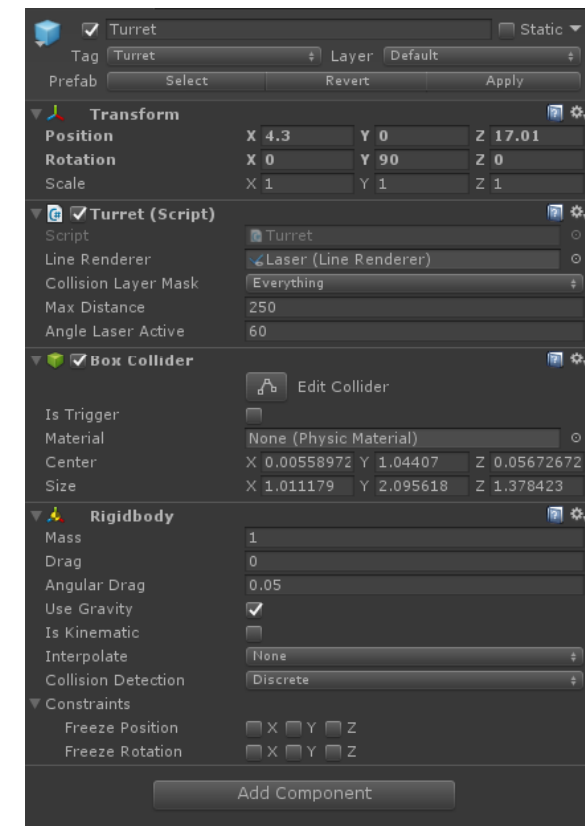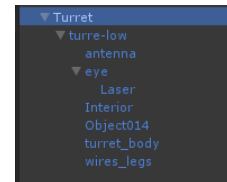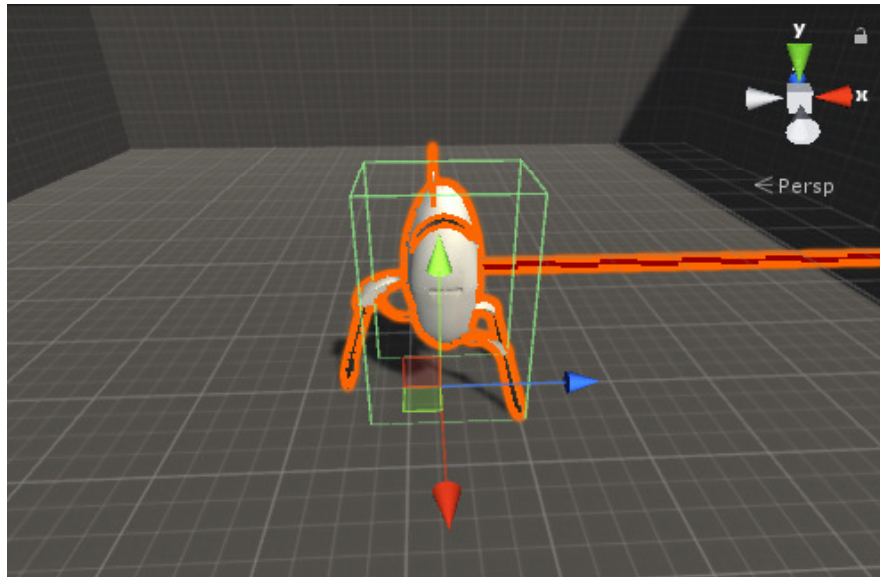
PortalButton

PortalButton – Implementación

```
public class PortalButton : MonoBehaviour
{
    public UnityEvent m_Event;

    void OnTriggerEnter(Collider _Collider)
    {
        if(_Collider.tag=="Companion")
            m_Event.Invoke();
    }
}
```
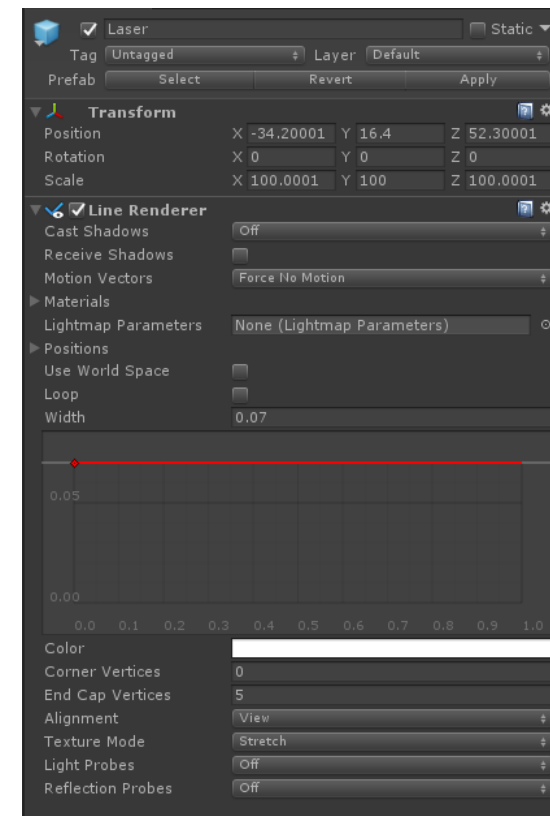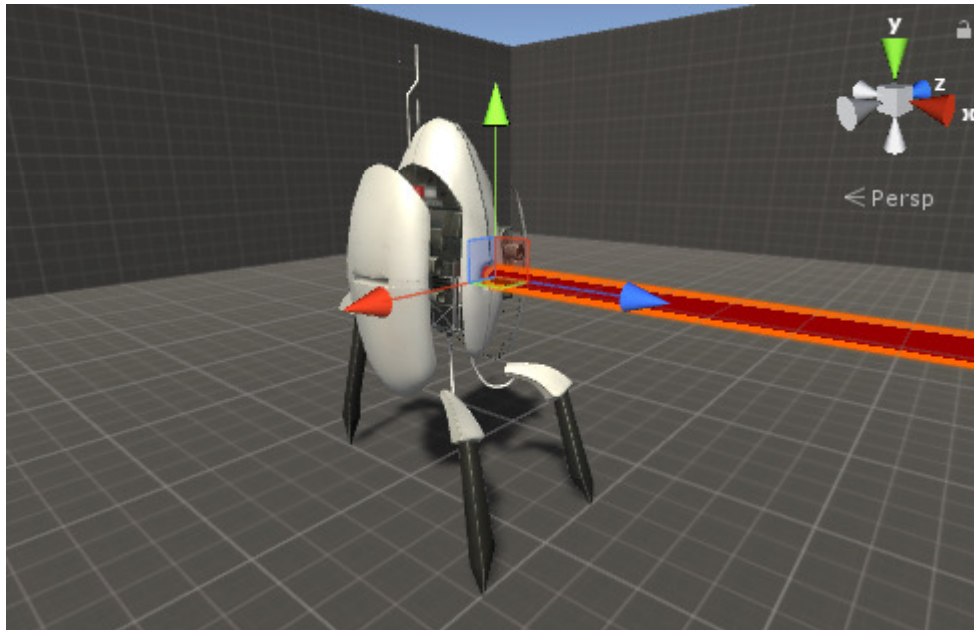
Turret

Turret - Laser

Turret – Implementación

```
public LineRenderer m_LineRenderer;
public LayerMask m_CollisionLayerMask;
public float m_MaxDistance=250.0f;
public float m_AngleLaserActive=60.0f;
```

Turret – Laser Implementación

```
Vector3 l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
RaycastHit l_RaycastHit;
if(Physics.Raycast(new Ray(m_LineRenderer.transform.position,
    m_LineRenderer.transform.forward), out l_RaycastHit, m_MaxDistance,
    m_CollisionLayerMask.value))
{
    l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
}
m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
```
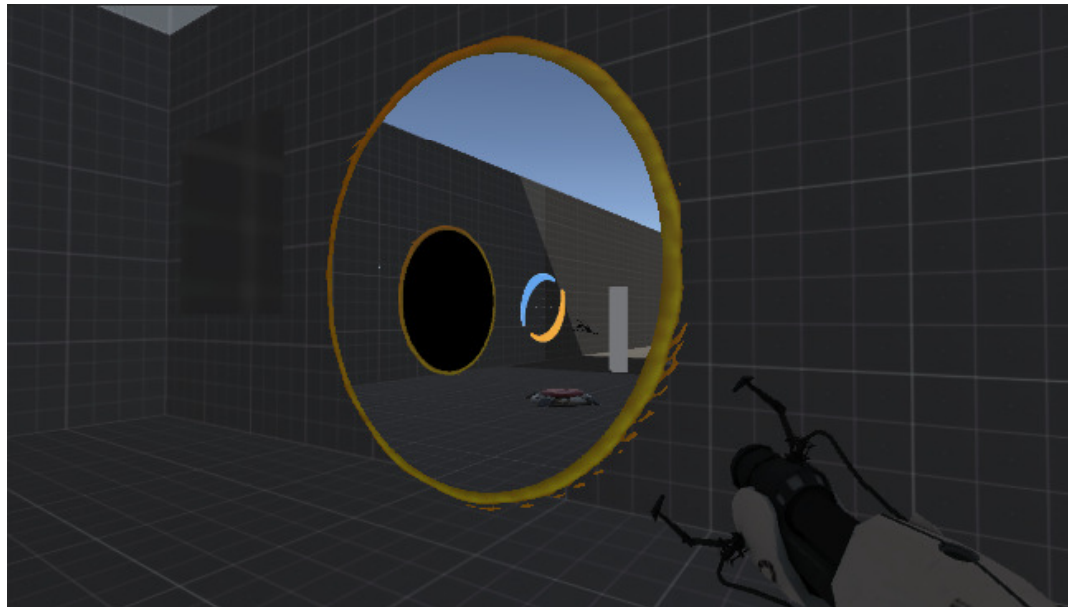
Turret – Laser activo Implementación

```
float l_DotAngleLaserActive=Mathf.Cos(m_AngleLaserActive*Mathf.Deg2Rad*0.5f);
bool l_RayActive=Vector3.Dot(transform.up, Vector3.up)>l_DotAngleLaserActive;
```
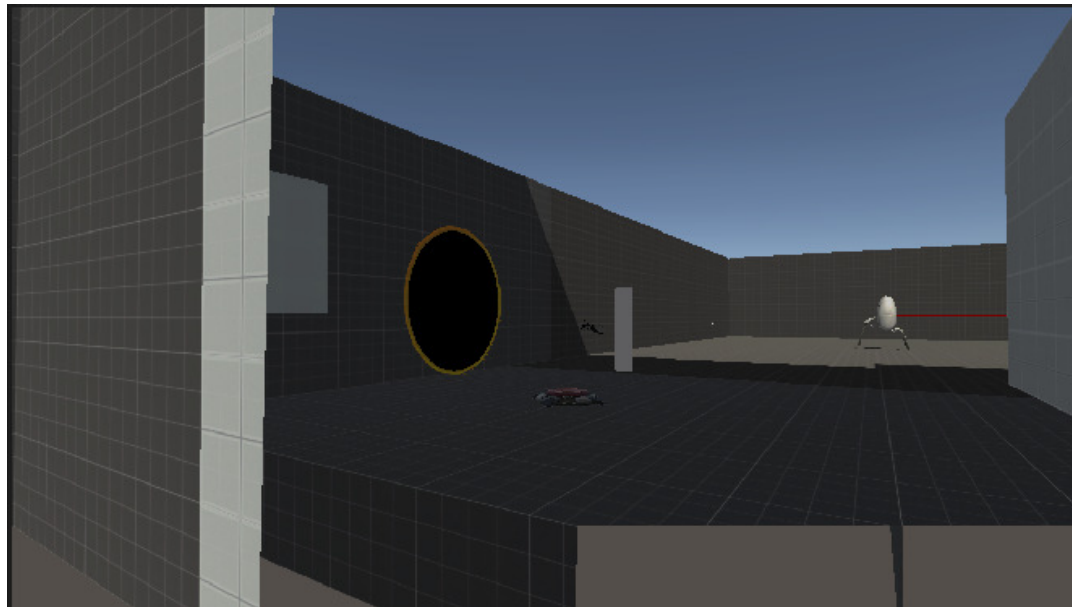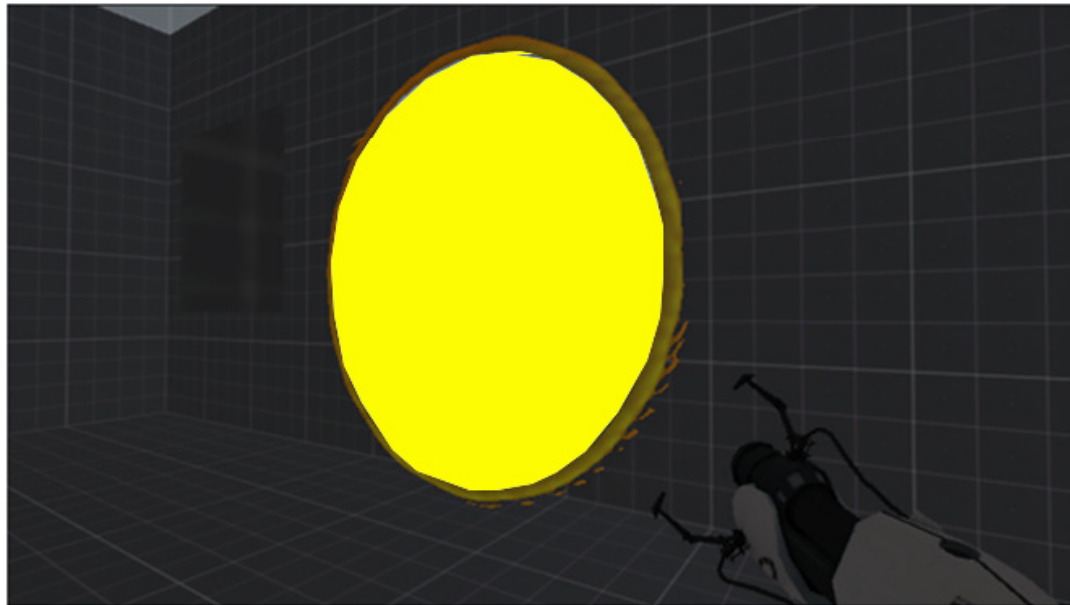
Portal - Cámara

Portal - RenderTarget

Portal - Shader

Portal – Shader RenderTarget
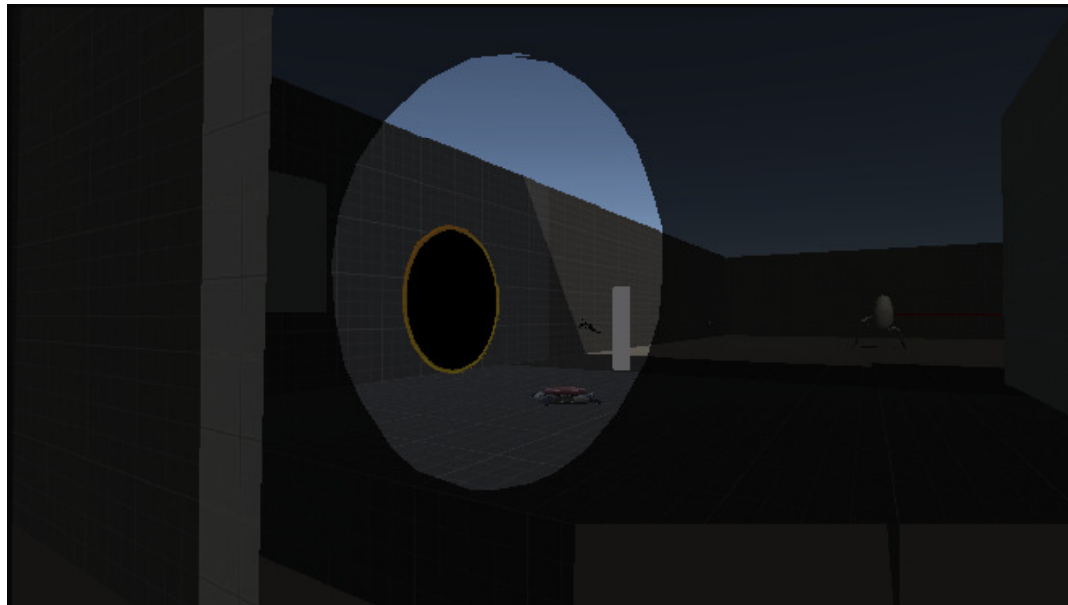
Portal - Resultado
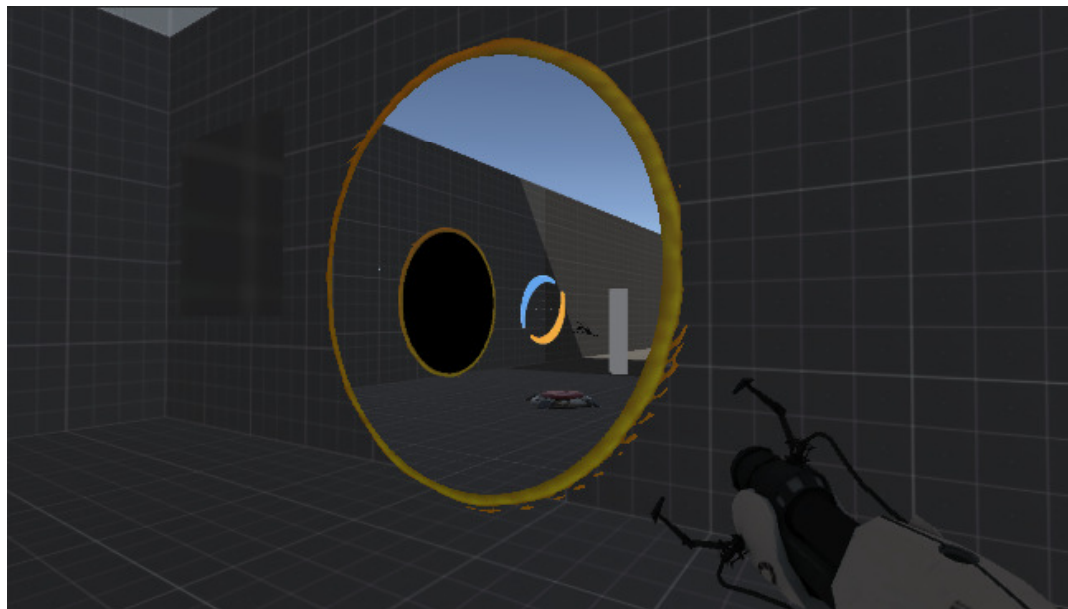
Portal Shader - Implementación
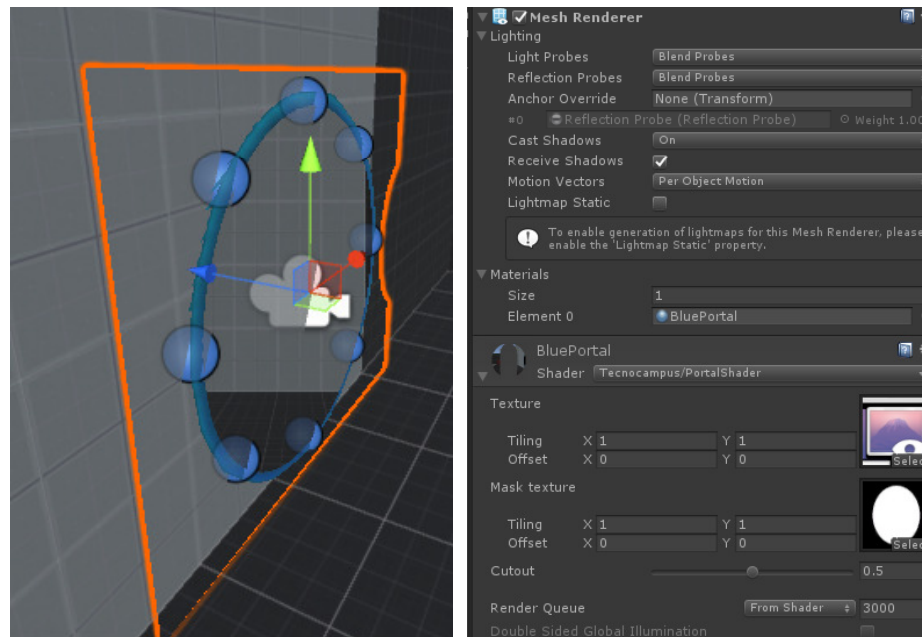
Portal Shader – Implementación

```
Shader "Tecnocampus/PortalShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _MaskTex("Mask texture", 2D) = "white" {}
        _Cutout("Cutout", Range(0.0, 1.0)) = 0.5
    }
    SubShader
    {
        Tags{ "Queue" = "Geometry " "IgnoreProjector" = "True" "RenderType" = "Opaque" }
        Lighting Off
        Cull Back
        ZWrite On
        ZTest Less

        Fog{ Mode Off }
```

Portal Shader – Implementación

```
Pass
{
    CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag

    #include "UnityCG.cginc"

    struct appdata
    {
            float4 vertex : POSITION;
            float2 uv : TEXCOORD0;
    };

    struct v2f
    {
            float4 vertex : SV_POSITION;
            float2 uv : TEXCOORD0;
            float4 screenPos : TEXCOORD1;
    };
```

Portal Shader – Implementación

```
v2f vert (appdata v)
{
            v2f o;
            o.vertex = UnityObjectToClipPos(v.vertex);
            o.uv = v.uv;
            o.screenPos = ComputeScreenPos(o.vertex);
            return o;
}

sampler2D _MainTex;
sampler2D _MaskTex;
float _Cutout;

fixed4 frag (v2f i) : SV_Target
{
            i.screenPos /= i.screenPos.w;

            fixed4 l_MaskColor= tex2D(_MaskTex, i.uv);
            if (l_MaskColor.a < _Cutout)
                            clip(-1);
            fixed4 col = tex2D(_MainTex, float2(i.screenPos.x, i.screenPos.y));

            return col;
}
ENDCG
        }
    }
}
```
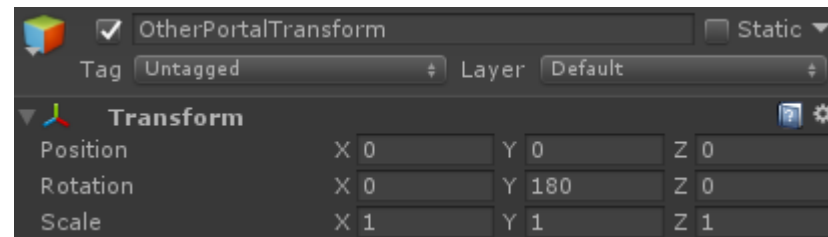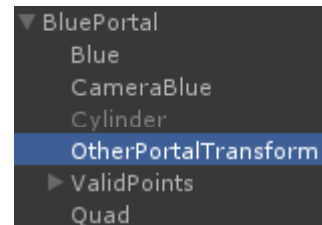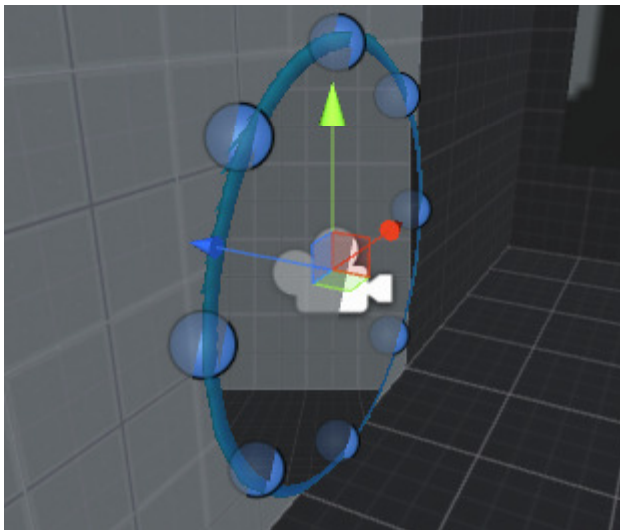
Portal – Cálculo posición de cámara (opción 1)

Portal – Cálculo posición de cámara (opción 1) Implementación

```
Vector3 l_ReflectedPosition=m_MirrorTransform.InverseTransformPoint(m_PlayerCamera.position);
Vector3 l_ReflectedDirection=m_MirrorTransform.InverseTransformDirection(m_PlayerCamera.forward);
m_MirrorPortal.m_PortalCamera.transform.position=m_MirrorPortal.transform.TransformPoint(l_ReflectedPosition);
m_MirrorPortal.m_PortalCamera.transform.forward=m_MirrorPortal.transform.TransformDirection(l_ReflectedDirecti
    on);

m_PortalCamera.nearClipPlane=Vector3.Distance(m_PortalCamera.transform.position,
    this.transform.position)+m_NearClipOffset;
```

Portal – Cálculo posición de cámara (opción 2)
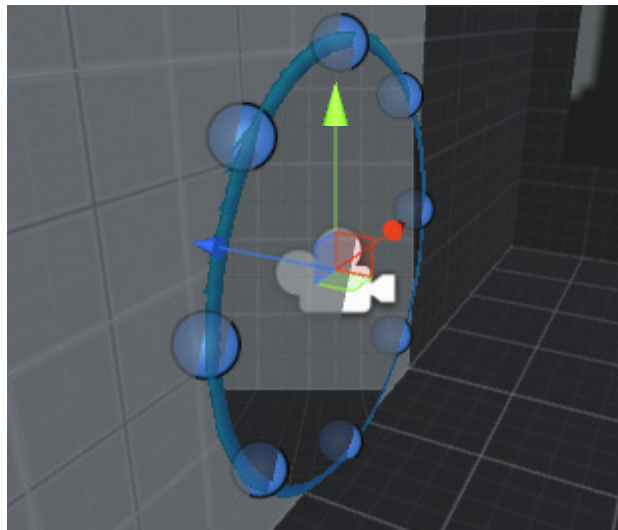
# Portal – Cálculo posición de cámara (opción 2) Implementación

```
Vector3 l_EulerAngles=transform.rotation.eulerAngles;
Quaternion l_Rotation=Quaternion.Euler(l_EulerAngles.x, l_EulerAngles.y+180.0f, l_EulerAngles.z);
Matrix4x4 l_WorldMatrix=Matrix4x4.TRS(transform.position, l_Rotation, transform.localScale);

Vector3 l_ReflectedPosition=l_WorldMatrix.inverse.MultiplyPoint3x4(m_PlayerCamera.position);
Vector3 l_ReflectedDirection=l_WorldMatrix.inverse.MultiplyVector(m_PlayerCamera.forward);
m_MirrorPortal.m_PortalCamera.transform.position=m_MirrorPortal.transform.TransformPoint(l_ReflectedPosition);
m_MirrorPortal.m_PortalCamera.transform.forward=m_MirrorPortal.transform.TransformDirection(l_ReflectedDirecti
    on);

m_PortalCamera.nearClipPlane=Vector3.Distance(m_PortalCamera.transform.position,
    this.transform.position)+m_NearClipOffset;
```

Refraction Cube
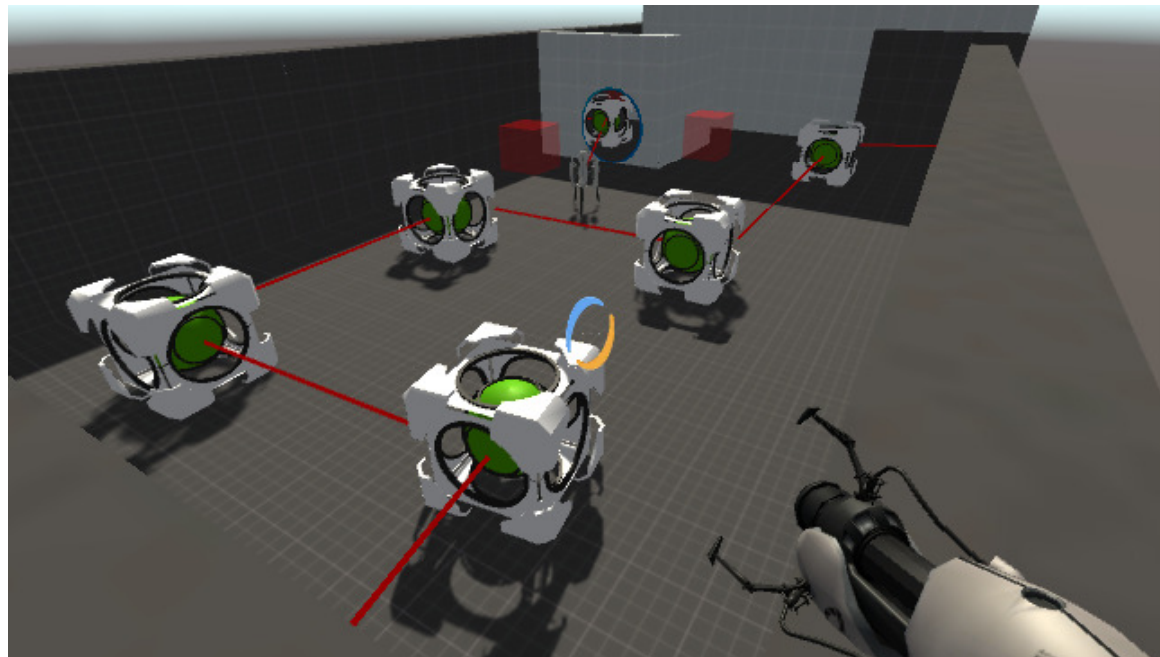
Refraction Cube – Implementación

```
void Update()
{
    m_LineRenderer.gameObject.SetActive(m_CreateRefraction);
    m_CreateRefraction=false;
}
public void CreateRefraction()
{
    m_CreateRefraction=true;
    Vector3 l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
    RaycastHit l_RaycastHit;
    if(Physics.Raycast(new Ray(m_LineRenderer.transform.position, m_LineRenderer.transform.forward), out l_RaycastHit, m_MaxDistance,
    m_CollisionLayerMask.value))
    {
        l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
        if(l_RaycastHit.collider.tag=="RefractionCube")
        {
            //Reflect ray
            l_RaycastHit.collider.GetComponent<RefractionCube>().CreateRefraction();
        }
        //Other collisions
    }
    m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
}
```

Refraction en Portal – Implementación

```
void Update()
{
      m_LineRenderer.gameObject.SetActive(m_CreateRefraction);
      m_CreateRefraction=false;
}
public void CreateRefraction()
{
      m_CreateRefraction=true;
      Vector3 l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
      RaycastHit l_RaycastHit;
      if(Physics.Raycast(new Ray(m_LineRenderer.transform.position, m_LineRenderer.transform.forward), out l_RaycastHit, m_MaxDistance,
      m_CollisionLayerMask.value))
      {
            l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
            if(l_RaycastHit.collider.tag=="RefractionCube")
            {
                  //Reflect ray
                  l_RaycastHit.collider.GetComponent<RefractionCube>().CreateRefraction();
            }
            //Other collisions
      }
      m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
}
```