# Anexo

LA REALIDAD AUMENTADA:

APLICACIONES ACCESIBLES A NUESTRO DIA A DIA

Código fuente de las aplicaciones RA

Joan Esgleas

En este Anexo, por motivos de extensión sólo he incluido el código que he escrito yo o he modificado directamente, pero se ha de mencionar que la aplicación usa código del motor y del SDK que no figura en el anexo.

# Aplicación para identificar carteles de películas

Esta script detecta la cámara RA y la vincula a una cámara en el espació gráfico.

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

public class DetectArCamera : MonoBehaviour
{
    public Canvas canvas;
    [HideInInspector]
    public Camera cam;
    void OnEnable()
    {
        cam = Camera.main;
        canvas.worldCamera = cam;
    }
    private void Update()
    {
       canvas.transform.rotation = Quaternion.LookRotation(transform.position -
       cam.transform.position);
    }
}
```

Esta script permite resetear la aplicación.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class EscenaManager : MonoBehaviour
{
    public void Refresh()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Esta script permite ocultar en el espació grafico elementos a los cuales la cámara no está enfocando.

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

public class HideOffCamera : MonoBehaviour
{
    PeliculaDisplay peliculaDisplay;
    DetectArCamera detectArCamera;
    private void Start()
    {
        detectArCamera = GetComponent<DetectArCamera>();
        peliculaDisplay = GetComponent<PeliculaDisplay>();
    }
    void Update()
    {
        Vector3 puntoEnPantalla =
detectArCamera.cam.WorldToViewportPoint(gameObject.transform.position);
        bool enPantalla = puntoEnPantalla.z > 0 && puntoEnPantalla.x > 0 &&
puntoEnPantalla.x < 1 && puntoEnPantalla.y > 0 && puntoEnPantalla.y < 1;
        if(enPantalla == false)
        {
            peliculaDisplay.cruz();
        }
    }
}
```

Esta script hace almacenar las imágenes que pueden de ser detectadas en RA y les asigna un nombre, (esta es una script del SDK pero yo la modifique un poco para que trabajara mejor con mi aplicación).

```
using System;
using UnityEngine;
using UnityEngine.XR.ARSubsystems;

namespace UnityEditor.XR.ARSubsystems
{
    /// <summary>
    /// Extension methods for the <see cref="XRReferenceImageLibrary"/>.
    /// </summary>
    /// <remarks>
    /// At runtime, <see cref="XRReferenceImageLibrary"/>s are immutable. These
    /// Editor-only extension methods let you build and manipulate image
libraries
    /// in Editor scripts.
    /// </remarks>
    public static class XRReferenceImageLibraryExtensions
    {
        public float identificador;
        public string nombre;
        /// <summary>
        /// Creates an empty <c>XRReferenceImage<c/> and adds it to the library.
The new
        /// reference image is inserted at the end of the list of reference
images.
        /// </summary>
        /// <param name="library">The <see cref="XRReferenceImageLibrary"/> being
extended.</param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
        public static void Add(this XRReferenceImageLibrary library)
        {
            if (library == null)
                throw new ArgumentNullException("library");

            library.m_Images.Add(new XRReferenceImage
            {
                m_SerializedGuid = SerializableGuidUtil.Create(Guid.NewGuid())
            });
        }

        /// <summary>
        /// Set the texture on the reference image.
        /// </summary>
        /// <param name="library">The <see cref="XRReferenceImageLibrary"/> being
extended.</param>
        /// <param name="index">The reference image index to modify.</param>
        /// <param name="texture">The texture to set.</param>
        /// <param name="keepTexture">Whether to store a strong reference to the
texture. If <c>true</c>,
        /// the texture will be available in the Player. Otherwise,
<c>XRReferenceImage.texture</c> will be set to <c>null</c>.</param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
```

```
        /// <exception cref="System.IndexOutOfRangeException">Thrown if <paramref
name="index"/> is not between 0 and <paramref name="library"/><c>.count -
1</c>.</exception>
        public static void SetTexture(this XRReferenceImageLibrary library, int
index, Texture2D texture, bool keepTexture)
        {
            ValidateAndThrow(library, index);

            var referenceImage = library.m_Images[index];
            referenceImage.m_SerializedTextureGuid =
SerializableGuidUtil.Create(GetGuidForTexture(texture));
            referenceImage.m_Texture = keepTexture ? texture : null;
            library.m_Images[index] = referenceImage;
        }

        /// <summary>
        /// Sets the <c>XRReferenceImage.specifySize</c> value on the
<c><c>XRReferenceImage</c> at <paramref name="index"/>.
        /// This value is read-only in the Player; it can only be modified in the
Editor.
        /// </summary>
        /// <param name="library">The <c>XRReferenceImageLibrary</c> being
extended.</param>
        /// <param name="index">The index of the reference image within the
library to modify.</param>
        /// <param name="specifySize">Whether <c>XRReferenceImage.size</c> is
specified.</param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
        /// <exception cref="System.IndexOutOfRangeException">Thrown if <paramref
name="index"/> is not between 0 and <paramref name="library"/><c>.count -
1</c>.</exception>
        public static void SetSpecifySize(this XRReferenceImageLibrary library,
int index, bool specifySize)
        {
            ValidateAndThrow(library, index);

            var image = library.m_Images[index];
            image.m_SpecifySize = specifySize;
            library.m_Images[index] = image;
        }

        /// <summary>
        /// Sets the <c>XRReferenceImage.size</c> value on the
<c><c>XRReferenceImage</c> at <paramref name="index"/>.
        /// This value is read-only in the Player; it can only be modified in the
Editor.
        /// </summary>
        /// <param name="library">The <c>XRReferenceImageLibrary</c> being
extended.</param>
        /// <param name="index">The index of the reference image within the
library to modify.</param>
        /// <param name="size"></param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
        /// <exception cref="System.IndexOutOfRangeException">Thrown if <paramref
name="index"/> is not between 0 and <paramref name="library"/><c>.count -
1</c>.</exception>
        public static void SetSize(this XRReferenceImageLibrary library, int
index, Vector2 size)
        {
            ValidateAndThrow(library, index);
            var image = library.m_Images[index];
            image.m_Size = size;
            library.m_Images[index] = image;
        }

        /// <summary>
        /// Sets the <c>XRReferenceImage.name</c> value on the
<c><c>XRReferenceImage</c> at <paramref name="index"/>.
        /// This value is read-only in the Player; it can only be modified in the
Editor.
        /// </summary>
        /// <param name="library">The <c>XRReferenceImageLibrary</c> being
extended.</param>
        /// <param name="index">The index of the reference image within the
library to modify.</param>
        /// <param name="name"></param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
        /// <exception cref="System.IndexOutOfRangeException">Thrown if <paramref
name="index"/> is not between 0 and <paramref name="library"/><c>.count -
1</c>.</exception>
        public static void SetName(this XRReferenceImageLibrary library, int
index, string name)
        {
            ValidateAndThrow(library, index);

            var image = library.m_Images[index];
            image.m_Name = name;
            image.m_Name = nombre;
            library.m_Images[index] = image;
            library.m_Images[index] = identificador;
        }

        /// <summary>
        /// Removes the <see cref="XRReferenceImage"/> at <paramref
name="index"/>.
        /// </summary>
        /// <param name="library">The <see cref="XRReferenceImageLibrary"/> being
extended.</param>
        /// <param name="index">The index in the list of images to
remove.</param>
        /// <exception cref="System.ArgumentNullException">Thrown if <paramref
name="library"/> is <c>null</c>.</exception>
        /// <exception cref="System.IndexOutOfRangeException">Thrown if <paramref
name="index"/> is not between 0 and <paramref name="library"/><c>.count -
1</c>.</exception>
        public static void RemoveAt(this XRReferenceImageLibrary library, int
index)
        {
            ValidateAndThrow(library, index);

            library.m_Images.RemoveAt(index);
        }

        static Guid GetGuidForTexture(Texture2D texture)
        {
            if (texture == null)
                return Guid.Empty;

            string guid;
            long localId;
            if (AssetDatabase.TryGetGUIDAndLocalFileIdentifier(texture, out guid, 
out localId))
                return new Guid(guid);

            return Guid.Empty;
        }

        static void ValidateAndThrow(XRReferenceImageLibrary library, int index)
        {
            if (library == null)
                throw new ArgumentNullException("library");

            if (library.count == 0)
                throw new IndexOutOfRangeException("The reference image library 
is empty; cannot index into it.");

            if (index < 0 || index >= library.count)
                throw new IndexOutOfRangeException(string.Format("{0} is out of 
range. 'index' must be between 0 and {1}", index, library.count - 1));
        }
    }
}
```

Esta script asigna un nombre a cada imagen una vez registrada por la cámara y la vincula a un “Scriptable Object”.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

[RequireComponent(typeof(ARTrackedImageManager))]
public class MultiplesImagenTraking : MonoBehaviour
{
    [SerializeField]
    private GameObject[] arObjectsPrefabs;
    [SerializeField]
    private Vector3 EscalaDeLosPrefabas = new Vector3(0.1f, 0.1f, 0.1f);
    [SerializeField]
    private Text NobreDeImagenText;
    [SerializeField]
    private float id;
    [SerializeField]
    private string nombrePelicula;



    private ARTrackedImageManager trackedImageManager;

    private Dictionary<string, GameObject> arObjects = new Dictionary<string,
GameObject>();

    void Awake()
    {
        trackedImageManager = GetComponent<ARTrackedImageManager>();
        foreach (GameObject arObject in arObjectsPrefabs)
        {
            GameObject newARObject = Instantiate(arObject, Vector3.zero,
Quaternion.identity);
            newARObject.SetActive(false);
            newARObject.name = arObject.name;
            arObjects.Add(arObject.name, newARObject);
        }
    }

    void OnEnable()
    {
        trackedImageManager.trackedImagesChanged += OnTrackedImagesChanged;
    }

    void OnDisable()
    {
        trackedImageManager.trackedImagesChanged -= OnTrackedImagesChanged;
    }


    void OnTrackedImagesChanged(ARTrackedImagesChangedEventArgs eventArgs)
    {
        foreach (ARTrackedImage trackedImage in eventArgs.added)
        {
            UpdateARImage(trackedImage);
        }

        foreach (ARTrackedImage trackedImage in eventArgs.updated)
        {
            UpdateARImage(trackedImage);
        }

        foreach (ARTrackedImage trackedImage in eventArgs.removed)
        {
            arObjects[trackedImage.name].SetActive(false);
        }
    }

    private void UpdateARImage(ARTrackedImage trackedImage)
    {
        XRReferenceImageLibraryExtensions().identificador = id;
        XRReferenceImageLibraryExtensions().nombre = nombrePelicula;
        NobreDeImagenText.text = trackedImage.referenceImage.name;
        AssignGameObject(trackedImage.referenceImage.name, 
trackedImage.transform.position);
    }

    void AssignGameObject(string name, Vector3 newPosition)
    {
        if (arObjectsPrefabs != null)
        {
            arObjects[name].SetActive(true);
            arObjects[name].transform.position = newPosition;
            arObjects[name].transform.localScale = EscalaDeLosPrefabas;
        }
    }
}
```

Esta script crea el “Scriptable Object” donde se almacena toda la información de la película.

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

[CreateAssetMenu(fileName = "Nueva Pelicula", menuName = "Pelicula")]
public class Pelicula : ScriptableObject
{
    //Información sobre la Pelicula
    public string nombreDePelicula;
    public string urlNetflix;
    public string urltrailer;

    public float year;
    public float duracion;
    public string pais;
    public string director;
    public string guion;
    public string musica;
    public string fotografia;
    public string reparto;
    public string sinposis;

    public float criticaAudiencia;
    public float crticaProfesional;
}
```

Esta script hace visible la UI al detectar la imagen correspondiente, también da función a todos los botones de la UI y por último vincula la información del “Scriptable Object” a la UI.

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

public class PeliculaDisplay : MonoBehaviour
{
    // Scriptable Object
    public Pelicula pelicula;

    //UI
    public GameObject aboutWindow;
    public GameObject sinopsisWindow;
    public GameObject netflixBoton;
    public GameObject abautBoton;
    public GameObject tituloWindow;
    public GameObject botonAtras;
    public GameObject activadorDeCartel;
    public GameObject cruzBotton;
    public GameObject trailerObject;
    public GameObject playButton;
    public GameObject videoCheck;

    //Objetos de textos
    public TextMeshProUGUI namePelicula;
    public TextMeshProUGUI year;
    public TextMeshProUGUI duracion;
    public TextMeshProUGUI pais;
    public TextMeshProUGUI director;
    public TextMeshProUGUI guion;
    public TextMeshProUGUI musica;
    public TextMeshProUGUI fotografia;
    public TextMeshProUGUI reparto;

    public TextMeshProUGUI sinopsis;

    public TextMeshProUGUI criticaAudiencia;
    public TextMeshProUGUI criticaProfesional;

    public VideoPlayer trailer;

    //Detec Camera
    public Canvas canvas;
    [HideInInspector]
    public Camera cam;

    void OnEnable()
    {
        activadorDeCartel.SetActive(false);

        aboutWindow.SetActive(false);
        botonAtras.SetActive(false);
        sinopsisWindow.SetActive(true);
        netflixBoton.SetActive(true);
        abautBoton.SetActive(true);
        tituloWindow.SetActive(true);
        videoCheck.SetActive(false);
        playButton.SetActive(true);
        trailerObject.SetActive(true);

        trailer.Stop();

        namePelicula.text = pelicula.nombreDePelicula;
        sinopsis.text = pelicula.sinposis;
        year.text = " Año: " + pelicula.year;
        duracion.text = " Duración: " + pelicula.duracion + " min";
        pais.text = " País: " + pelicula.pais;
        director.text = " Dirección: " + pelicula.director;
        guion.text = " Guion: " + pelicula.guion;
        musica.text = " Música: " + pelicula.musica;
        fotografia.text = " Fotografía: " + pelicula.fotografia;
        reparto.text = " Reparto: " + pelicula.reparto;
        criticaAudiencia.text = "" + pelicula.criticaAudiencia;
        criticaProfesional.text = "" + pelicula.crticaProfesional;

        cam = Camera.main;
        canvas.worldCamera = cam;

    }
    void Update(){

        canvas.transform.rotation = Quaternion.LookRotation(transform.position - 
cam.transform.position);

        Vector3 puntoEnPantalla = 
cam.WorldToViewportPoint(gameObject.transform.position);
        bool enPantalla = puntoEnPantalla.z > 0 && puntoEnPantalla.x > 0 && 
puntoEnPantalla.x < 1 && puntoEnPantalla.y > 0 && puntoEnPantalla.y < 1;
        if(enPantalla == false)
        {
            cruz();
        }
    }
    public void netflix()
    {
        Application.OpenURL(pelicula.urlNetflix);
    }
    public void cruz()
    {
        activadorDeCartel.SetActive(true);
        aboutWindow.SetActive(false);
        botonAtras.SetActive(false);
        sinopsisWindow.SetActive(false);
        netflixBoton.SetActive(false);
        abautBoton.SetActive(false);
        tituloWindow.SetActive(false);
        cruzBotton.SetActive(false);
        trailer.Stop();
        videoCheck.SetActive(false);
        playButton.SetActive(false);
        trailerObject.SetActive(false);

    }
```

```
public void about()
{
    aboutWindow.SetActive(true);
    sinopsisWindow.SetActive(false);
    netflixBoton.SetActive(false);
    abautBoton.SetActive(false);
    tituloWindow.SetActive(false);
    botonAtras.SetActive(true);
    trailerObject.SetActive(false);
    videoCheck.SetActive(false);
    playButton.SetActive(false);
    trailerObject.SetActive(false);
    trailer.Pause();
}
public void atras()
{
    aboutWindow.SetActive(false);
    botonAtras.SetActive(false);
    sinopsisWindow.SetActive(true);
    netflixBoton.SetActive(true);
    abautBoton.SetActive(true);
    tituloWindow.SetActive(true);
    trailerObject.SetActive(false);
    videoCheck.SetActive(false);
    playButton.SetActive(true);
    trailerObject.SetActive(true);
    trailer.Pause();
}
public void activarCartel()
{
    aboutWindow.SetActive(false);
    botonAtras.SetActive(false);
    sinopsisWindow.SetActive(true);
    netflixBoton.SetActive(true);
    abautBoton.SetActive(true);
    tituloWindow.SetActive(true);
    activadorDeCartel.SetActive(false);
    cruzBotton.SetActive(true);
    videoCheck.SetActive(false);
    playButton.SetActive(true);
    trailerObject.SetActive(true);

    trailer.Stop();
}
public void trailerPlay()
{
    videoCheck.SetActive(true);
    playButton.SetActive(false);
    trailerObject.SetActive(true);
    trailer.Play();
}
public void trailerStop()
{
    videoCheck.SetActive(false);
    playButton.SetActive(true);
    trailerObject.SetActive(true);
    trailer.Pause();
}
public void youtube()
{
    Application.OpenURL(pelicula.urltrailer);
}
```

```
    public void Onscreen()
    {
        activadorDeCartel.SetActive(true);
    }
}
```

# Aplicación para representar puntos rectas y planos en RA

Esta script crea las líneas de colores del eje de coordenadas.

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

public class Guizmo : MonoBehaviour
{
    public int lineCount = 2;
    public float distancia = 50f;

    Vector3 pointB;
    private Color coloro;
    public enum colors
    {
        Rojo, Verde, Azul
    }
    public colors colores;

    public void Check_color()
    {
        switch (colores)
        {
            case colors.Rojo:
                coloro = Color.red;
                break;

            case colors.Verde:
                coloro = Color.green;
                break;

            case colors.Azul:
                coloro = Color.blue;
                break;
        }
    }

    static Material lineMaterial;
    private void Start()
    {
        Check_color();
    }
    static void CreateLineMaterial()
    {
        if (!lineMaterial)
        {
            Shader shader = Shader.Find("Hidden/Internal-Colored");
            lineMaterial = new Material(shader);
            lineMaterial.hideFlags = HideFlags.HideAndDontSave;
            lineMaterial.SetInt("_SrcBlend",
(int)UnityEngine.Rendering.BlendMode.SrcAlpha);
            lineMaterial.SetInt("_DstBlend",
(int)UnityEngine.Rendering.BlendMode.OneMinusSrcAlpha);
            lineMaterial.SetInt("_Cull",
(int)UnityEngine.Rendering.CullMode.Off);
            lineMaterial.SetInt("_ZWrite", 0);
        }
    }
    public void OnRenderObject()
    {
        Check_color();
        CreateLineMaterial();
        lineMaterial.SetPass(0);

        GL.PushMatrix();

        GL.MultMatrix(transform.localToWorldMatrix);

        GL.Begin(GL.LINES);
        for (int i = 0; i < lineCount; ++i)
        {
            float n = i / (float)lineCount;
            float angleV = n * Mathf.PI * 2;
            //float puntZ = Mathf.Sqrt(Mathf.Pow(Mathf.Cos(angleV),2f) +
Mathf.Pow(Mathf.Sin(angleV),2f));
            // Vertex colors cambian
            GL.Color(coloro);
            // 1 vertex a transform position
            GL.Vertex3(0, 0, 0);
            // un altre vertex al final del cercle
            pointB = new Vector3(Mathf.Cos(angleV) * distancia, Mathf.Sin(angleV)
* distancia, 0);
            GL.Vertex3(pointB.x,pointB.y,pointB.z);
        }
        GL.End();
        GL.PopMatrix();
    }

}
```

Esta script segmenta el eje de coordenadas y crea un elemento con texto cada 0,1 unidades en la dirección de cada eje.

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

public class ValorsDeCordenades : MonoBehaviour
{
    [SerializeField]
    float maxNumberOfCubes = 50;
    float nombreSegment;
    float perSegment;

    [SerializeField]
    GameObject numero;

    public float valor;

    ValorAlIniciar valorquetoma;

    private void Awake()
    {
        valorquetoma = numero.GetComponent<ValorAlIniciar>();
    }
    private void Update()
    {
        if (nombreSegment <= maxNumberOfCubes)
        {
            Creat();
        }
    }
    void Creat()
    {
        nombreSegment += 1;
        perSegment = (nombreSegment);

        for (int i = 1; i < nombreSegment; i++)
        {
            //X
            Vector3 disxpos = new Vector3(transform.position.x + 0.1f* + i,
transform.position.y, transform.position.z);
            Vector3 CreatPositionxpos = transform.position + (disxpos);
            Vector3 disxneg = new Vector3(transform.position.x + 0.1f * +i * -1,
transform.position.y, transform.position.z);
            Vector3 CreatPositionxneg = transform.position + (disxneg);

            //Y
            Vector3 disypos = new Vector3(transform.position.x ,
transform.position.y + 0.1f * +i, transform.position.z);
            Vector3 CreatPositionypos = transform.position + (disypos);
            Vector3 disyneg = new Vector3(transform.position.x,
transform.position.y + 0.1f * +i * -1, transform.position.z);
            Vector3 CreatPositionyneg = transform.position + (disyneg);

            //Z
            Vector3 diszpos = new Vector3(transform.position.x ,
transform.position.y, transform.position.z + 0.1f * +i);
            Vector3 CreatPositionzpos = transform.position + (diszpos);
            Vector3 diszneg = new Vector3(transform.position.x ,
transform.position.y, transform.position.z + 0.1f * +i * -1);
            Vector3 CreatPositionzneg = transform.position + (diszneg);

            valor = i;
            valorquetoma.valoress = valor;
            Instantiate(numero, CreatPositionxpos, Quaternion.identity);
            Instantiate(numero, CreatPositionxneg, Quaternion.identity);

            Instantiate(numero, CreatPositionypos, Quaternion.identity);
            Instantiate(numero, CreatPositionyneg, Quaternion.identity);

            Instantiate(numero, CreatPositionzpos, Quaternion.identity);
            Instantiate(numero, CreatPositionzneg, Quaternion.identity);

        }
    }
    }
```

Esta script determina el valor de cada división hecha por la script anterior.

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

public class ValorAlIniciar : MonoBehaviour
{
    ValorsDeCordenades valorsDeCordenades;
    public float valoress;
    ARTapToPlaceObject placeObjetss;

    private void Start()
    {
        placeObjetss = FindObjectOfType<ARTapToPlaceObject>();
    }
    // Start is called before the first frame update
    void OnEnable()
    {
       TextMeshPro textis = gameObject.GetComponent<TextMeshPro>();
        if (transform.position.x < 0 || transform.position.y < 0 ||
transform.position.z < 0)
        {
            textis.text = "-" + valoress;
        }
        else
        {
            textis.text = "" + valoress;
        }


    }
}
```

Esta script sitúa un indicador y una vez la pantalla es pulsada esta script

también crea el eje de coordenadas.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.Experimental.XR;
using UnityEngine.XR.ARSubsystems;
using System;

public class ARTapToPlaceObject : MonoBehaviour
{
    public GameObject objectToPlace;
    public GameObject placementIndicator;

    private ARRaycastManager arOrigin;
    private Pose placementPose;
    private bool placementPoseIsValid = false;

    void Start()
    {
        arOrigin = FindObjectOfType<ARRaycastManager>();
        objectToPlace.SetActive(false);
    }

    void Update()
    {
        UpdatePlacementPose();
        UpdatePlacementIndicator();
        if (placementPoseIsValid && Input.touchCount > 0 &&
Input.GetTouch(0).phase == TouchPhase.Began)
        {
            PlaceObject();
        }
    }

    private void PlaceObject()
    {
        objectToPlace.transform.position = placementPose.position;
        objectToPlace.transform.rotation = placementPose.rotation;
        objectToPlace.SetActive(true);
    }

    private void UpdatePlacementIndicator()
    {
        if (placementPoseIsValid)
        {
            placementIndicator.SetActive(true);

placementIndicator.transform.SetPositionAndRotation(placementPose.position,
placementPose.rotation);
        }
        else
        {
            placementIndicator.SetActive(false);
        }
    }
    private void UpdatePlacementPose()
    {
        var screenCenter = Camera.current.ViewportToScreenPoint(new Vector3(0.5f,
0.5f));
        var hits = new List<ARRaycastHit>();
        arOrigin.Raycast(screenCenter, hits, TrackableType.Planes);

        placementPoseIsValid = hits.Count > 0;
        if (placementPoseIsValid)
        {
            placementPose = hits[0].pose;

            var cameraForward = Camera.current.transform.forward;
            var cameraBearing = new Vector3(cameraForward.x, 0,
cameraForward.z).normalized;
            placementPose.rotation = Quaternion.LookRotation(cameraBearing);
        }
    }
}
```

Esta script regula la UI que indica que pasos se han de seguir para situar el eje de coordenadas y fija su posición.

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

public class SituarSistemaDeReferencia : MonoBehaviour
{
    public GameObject primerPaso;
    public GameObject segundoPaso;
    public GameObject Indicador;
    public GameObject fondo;
    public GameObject boton;
    ARTapToPlaceObject scripSistemasRef;
    bool todochuta;
    void Start()
    {
        scripSistemasRef = gameObject.GetComponent<ARTapToPlaceObject>();
        todochuta = false;
    }
    private void Update()
    {
        if(Indicador.activeSelf == false)
        {
            if (todochuta == false)
            {
                primerPaso.SetActive(true);
                segundoPaso.SetActive(false);
                boton.SetActive(false);
            }
        }
        else
        {
            primerPaso.SetActive(false);
        }
        if (Indicador.activeSelf == true)
        {
            if (todochuta == false)
            {
                primerPaso.SetActive(false);
                segundoPaso.SetActive(true);
                boton.SetActive(true);
            }
        }
    }
    public void DesabilitarSistemaPlace()
    {
        todochuta = true;
        scripSistemasRef.enabled = false;
        primerPaso.SetActive(false);
        segundoPaso.SetActive(false);
        fondo.SetActive(false);
        Indicador.SetActive(false);
        boton.SetActive(false);
    }
}
```

Esta script crea y gestiona la UI y almacena los datos del inspector, además se encarga de hacer los cálculos y representar la intersección entre dos planos.

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

public class TabManager : MonoBehaviour
{
    public GameObject Inspector;
    public GameObject botonMostrarInspector;
    public GameObject bloqueDeElementos;
    GameObject OrigenDeCordenades;
    public int maxElementos = 11;
    public GameObject Elementobj;
    [HideInInspector]
    public int cantidadDeElementos;
    public int cantidadDePlanos=0;

    public RectTransform PanelElementos;

    GameObject[] tagInspec;
    bool InspectorActivador;

    public bool activarBuscaDeInterject = false;
    public List<GameObject> butPlan;
    public GameObject BotonInterjeccion;

    public bool botInter = false;

    string planeIndicator;
    public Interjec2 interS;
    public CradorRecta cradorRecta;

    [HideInInspector]
    public bool RectaDetectada;

    float PuntoUnoR;
    float PuntoDosR;
    float PuntoTresR;
    float VectorUnoR;
    float VectorDosR;
    float VectorTresR;
    private void Start()
    {
        Inspector.SetActive(true);
        tagInspec = GameObject.FindGameObjectsWithTag("Inspector");
        foreach (GameObject go in tagInspec)
        {
            go.SetActive(false);
        }
        InspectorActivador = false;
        butPlan = new List<GameObject>();

    }
    private void Update()
    {
        if (OrigenDeCordenades == null)
        {
            OrigenDeCordenades =
GameObject.FindGameObjectWithTag("OrigenDeCordenadas");
            botonMostrarInspector.SetActive(false);
        }
        else
        {
            if (InspectorActivador == false)
            {
                botonMostrarInspector.SetActive(true);
            }
            else
            {
                botonMostrarInspector.SetActive(false);
                tagInspec = GameObject.FindGameObjectsWithTag("Inspector");
            }
        }

        if (botInter == false)
        {

butPlan.AddRange(GameObject.FindGameObjectsWithTag("BotonInterjector"));
            foreach (GameObject gameObject in butPlan)
            {
                gameObject.SetActive(false);
            }
        }

    }
    public void MostrarInspector()
    {
        foreach (GameObject go in tagInspec)
        {
            go.SetActive(true);
        }
        InspectorActivador = true;
        botonMostrarInspector.SetActive(false);
    }
    public void OcultarInspector()
    {

        foreach (GameObject go in tagInspec)
        {
            go.SetActive(false);
        }
        InspectorActivador = false;
        botonMostrarInspector.SetActive(true);
    }
    public void BottonCrearPlanoInspector()
    {
        if (cantidadDeElementos < maxElementos)
        {
            GameObject obj = Instantiate(Elementobj, PanelElementos.transform);
            cantidadDeElementos++;
            cantidadDePlanos++;
            AdminElemento AdminElem = obj.GetComponent<AdminElemento>();
            obj.transform.SetParent(Inspector.transform);
            obj.transform.SetParent(bloqueDeElementos.transform);
            obj.transform.position = new Vector3(PanelElementos.position.x,
PanelElementos.position.y - 150 * (cantidadDeElementos - 1), 0);

            AdminElem.Elemento = "Plano";
            planeIndicator = "Plano";
            obj.SetActive(true);
        }
    }
    public void BottonCrearRectaInspector()
    {
        if (cantidadDeElementos < maxElementos)
        {
            GameObject obj = Instantiate(Elementobj, PanelElementos.transform);
            cantidadDeElementos++;
            AdminElemento AdminElem = obj.GetComponent<AdminElemento>();
            obj.transform.SetParent(Inspector.transform);
            obj.transform.SetParent(bloqueDeElementos.transform);
            obj.transform.position = new Vector3(PanelElementos.position.x,
PanelElementos.position.y - 150 * (cantidadDeElementos - 1), 0);

            AdminElem.Elemento = "Recta";
            obj.SetActive(true);
        }
    }
    public void BottonCrearPuntoInspector()
    {
        if (cantidadDeElementos < maxElementos)
        {
            GameObject obj = Instantiate(Elementobj, PanelElementos.transform);
            cantidadDeElementos++;
            AdminElemento AdminElem = obj.GetComponent<AdminElemento>();
            obj.transform.SetParent(Inspector.transform);
            obj.transform.SetParent(bloqueDeElementos.transform);
            obj.transform.position = new Vector3(PanelElementos.position.x,
PanelElementos.position.y - 150 * (cantidadDeElementos - 1), 0);
            AdminElem.Elemento = "Punto";
            obj.SetActive(true);
        }
    }

    public void Detectarintergetcion()
    {
        if (planeIndicator == "Plano")
        {
            botInter = true;
            foreach (GameObject gameObject in butPlan)
            {
                gameObject.SetActive(true);
            }
        }

    }
    public void IndetectarIntegracion()
    {
            botInter = false;
            foreach (GameObject gameObject in butPlan)
            {
                gameObject.SetActive(false);
            }
        RectaDetectada = true;

    }
    public void CrearRecta()
    {

        if (cantidadDeElementos < maxElementos)
        {
            interS = GameObject.FindObjectOfType<Interjec2>();
            float A = interS.A;
            float B = interS.B;
            float C = interS.C;
            float D = interS.D;
            float E = interS.pA;
            float F = interS.pB;
            float G = interS.pC;
            float H = interS.pD;
            float M = ((A * F) - (E * B));
            if (M != 0)
            {
                PuntoUnoR = ((H * B) - (D * F)) / ((A * F) - (E * B));
                PuntoDosR = ((D * E) - (A * H)) / ((A * F) - (E * B));
                PuntoTresR = 0;
                VectorUnoR = ((G * B) - (C * F)) / ((A * F) - (E * B));
                VectorDosR = ((C * E) - (G * A)) / ((A * F) - (E * B));
                VectorTresR = 1;
            }
            if (M == 0)
            {
                PuntoUnoR = 0;
                PuntoDosR = ((C * H) - (D * G)) / ((B * G) - (F * C));
                PuntoTresR = ((F*D) - (B * H)) / ((B * G) - (F * C));
                VectorUnoR = 1;
                VectorDosR = ((C * E) - (G * A)) / ((B * G) - (F * C));
                VectorTresR = ((F * A) - (B * E)) / ((B * G) - (F * C));
            }

            GameObject obj = Instantiate(Elementobj, PanelElementos.transform);
            cantidadDeElementos++;
            AdminElemento AdminElem = obj.GetComponent<AdminElemento>();
            obj.transform.SetParent(Inspector.transform);
            obj.transform.SetParent(bloqueDeElementos.transform);
            obj.transform.position = new Vector3(PanelElementos.position.x,
PanelElementos.position.y - 150 * (cantidadDeElementos - 1), 0);
            AdminElem.P1.text = PuntoUnoR.ToString();
            AdminElem.P2.text = PuntoDosR.ToString();
            AdminElem.P3.text = PuntoTresR.ToString();
            AdminElem.D1.text = VectorUnoR.ToString();
            AdminElem.D2.text = VectorDosR.ToString();
            AdminElem.D3.text = VectorTresR.ToString();
            AdminElem.Elemento = "Recta";
            obj.SetActive(true);
        }

    }
}
```

Esta script elimina los elementos deseados, su uso ha sido exclusivo durante el desarrollo, pero no es ejecutable en la aplicación final.

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

public class ActivarInterjeccion : MonoBehaviour
{
    public GameObject[] CosasInabilitas;

    public void BotonInabilitar()
    {
        for (int i = 0; i < CosasInabilitas.Length; i++)
        {
            CosasInabilitas[i].SetActive(false);
        }
    }
}
```

Esta script gestiona toda la UI en la cual el usuario introduce la información de los puntos, planos o rectas.

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

public class AdminElemento : MonoBehaviour
{

    //Creador de Plano
    public GameObject Plano;
    public GameObject CearPlanoInspector;
    public InputField ComponenteADelPlano;
    public InputField ComponenteBDelPlano;
    public InputField ComponenteCDelPlano;
    public InputField ComponenteDDelPlano;
    public Slider EscaladorDelPlano;
    public Dropdown SelectorDeColoresPlano;
    public CreadorDePlanos creadorPlano;

    [Space(20)]
    //Creador de recta
    public GameObject Recta;
    public GameObject CearRectaInspector;
    public InputField P1;
    public InputField P2;
    public InputField P3;
    public InputField D1;
    public InputField D2;
    public InputField D3;
    public Slider EscaladorDelRecta;
    public Dropdown SelectorDeColoresRecta;
    CradorRecta creadorRecta;
    [Space(20)]
    //Creador de punto
    public GameObject Punto;
    public GameObject CearPuntoInspector;
    public InputField CordenadaA;
    public InputField CordenadaB;
    public InputField CordenadaC;
    public InputField NombrePunto;
    public Dropdown SelectorDeColoresPunto;
    CreadorDePuntos creadorDePuntos;
    [Space(20)]
    public TextMeshProUGUI ElementoText;
    public TextMeshProUGUI FormulaTexto;
    public Image ColorenInspector;
    public string Elemento;
    public Color ColorGeneral;
    TabManager tabManager;

    GameObject OrdCoord;
    Transform OrigenDeCordenadas;

    public Image ColorElemento;
    GameObject tabuladorM;
    bool Mostrable = true;
    int numelemento;

    public float CantidadDePlanos = 0;


    private void Start()
    {
        Plano.SetActive(false);
        CearPlanoInspector.SetActive(false);
        Recta.SetActive(false);
        CearRectaInspector.SetActive(false);
        Punto.SetActive(false);
        CearPuntoInspector.SetActive(false);
        tabuladorM = GameObject.FindGameObjectWithTag("TabManager");
        tabManager = tabuladorM.GetComponent<TabManager>();
        numelemento = tabManager.cantidadDeElementos;
    }
    private void Update()
    {
        if (OrdCoord == null)
        {
            OrdCoord = GameObject.FindGameObjectWithTag("OrigenDeCordenadas");
            OrigenDeCordenadas = OrdCoord.transform;
        }
        gameObject.transform.SetSiblingIndex(tabManager.cantidadDeElementos -
numelemento);
        ColorenInspector.color = ColorGeneral;
        #region ElementoPlano de Texto
        ElementoText.text = Elemento;
        #endregion
        if (Elemento == "Plano"&& Mostrable == true)
        {
            if (Plano.activeSelf == false)
            {
                Plano.SetActive(true);
                CearPlanoInspector.SetActive(true);
            }
            if (creadorPlano == null)
            {
                creadorPlano = Plano.GetComponent<CreadorDePlanos>();
            }
            ManagerPlano();
            #region Formula Plano Texto
            FormulaTexto.text = creadorPlano.A + "x+" + creadorPlano.C + "y+" +
creadorPlano.B + "z+" + creadorPlano.D + "=0";
            #endregion
        }
        if (Elemento == "Recta" && Mostrable == true)
        {
            if(Recta.activeSelf == false)
            {
                Recta.SetActive(true);
                CearRectaInspector.SetActive(true);
            }
            if(creadorRecta == null)
            {
                creadorRecta = Recta.GetComponent<CradorRecta>();
            }
            ManagerRecta();
            #region Formula Recta Texto
            FormulaTexto.text ="("+
creadorRecta.punto.x+","+creadorRecta.punto.z+","+creadorRecta.punto.y+")"+"+"+"λ
"+"("+ creadorRecta.VectorDirector.x+","+ creadorRecta.VectorDirector.z+","+
creadorRecta.VectorDirector.y+")"+"=X";
            #endregion
        }
        if(Elemento == "Punto" && Mostrable == true)
        {
            if(Punto.activeSelf == false)
            {
                Punto.SetActive(true);
                CearPuntoInspector.SetActive(true);
            }
            if (creadorDePuntos == null)
            {
                creadorDePuntos = Punto.GetComponent<CreadorDePuntos>();
            }
            ManagerPunto();
            #region Formula Punto Texto
            FormulaTexto.text = NombrePunto.text+"=" + "(" +
creadorDePuntos.cordenadas.x + "," + creadorDePuntos.cordenadas.z + "," +
creadorDePuntos.cordenadas.y + ")" ;
            #endregion
        }
    }
    void ManagerPlano()
    {
        Plano.transform.position = OrdCoord.transform.position;
        Plano.transform.rotation = OrdCoord.transform.rotation;

        #region DeterminarColor
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Rojo")
        {
            ColorRojo();

        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Azul")
        {
            ColorAzul();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Cian")
        {
            ColorCian();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Gris")
        {
            ColorGris();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Verde")
        {
            ColorVerde();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Lila")
        {
            ColorLila();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Blanco")
        {
            ColorBlanco();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Amarillo")
        {
            ColorAmarillo();
        }
        if (SelectorDeColoresPlano.options[SelectorDeColoresPlano.value].text ==
"Negro")
        {
            ColorNegro();
        }
        #endregion
        creadorPlano.A = float.Parse(ComponenteADelPlano.text);
        creadorPlano.B = float.Parse(ComponenteBDelPlano.text);
        creadorPlano.C = float.Parse(ComponenteCDelPlano.text);
        creadorPlano.D = float.Parse(ComponenteDDelPlano.text);
        creadorPlano.escala = EscaladorDelPlano.value;
        creadorPlano.colorPlano = ColorGeneral;
    }
    void ManagerRecta()
    {
        Recta.transform.position = OrdCoord.transform.position;
        Recta.transform.rotation = OrdCoord.transform.rotation;

        #region DeterminarColor2
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Rojo")
        {
            ColorRojo();

        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Azul")
        {
            ColorAzul();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Cian")
        {
            ColorCian();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Gris")
        {
            ColorGris();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Verde")
        {
            ColorVerde();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Lila")
        {
            ColorLila();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Blanco")
```

```
        {
            ColorBlanco();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Amarillo")
        {
            ColorAmarillo();
        }
        if (SelectorDeColoresRecta.options[SelectorDeColoresRecta.value].text ==
"Negro")
        {
            ColorNegro();
        }
        #endregion
        creadorRecta.punto.x = float.Parse(P1.text);
        creadorRecta.punto.z = float.Parse(P2.text);
        creadorRecta.punto.y = float.Parse(P3.text);
        creadorRecta.VectorDirector.x = float.Parse(D1.text);
        creadorRecta.VectorDirector.z = float.Parse(D2.text);
        creadorRecta.VectorDirector.y = float.Parse(D3.text);
        creadorRecta.escala = EscaladorDelRecta.value;
        creadorRecta.colorRecta = ColorGeneral;
    }

    void ManagerPunto()
    {
        #region DeterminarColor3
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Rojo")
        {
            ColorRojo();

        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Azul")
        {
            ColorAzul();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Cian")
        {
            ColorCian();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Gris")
        {
            ColorGris();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Verde")
        {
            ColorVerde();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Lila")
        {
            ColorLila();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Blanco")
        {
            ColorBlanco();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Amarillo")
        {
            ColorAmarillo();
        }
        if (SelectorDeColoresPunto.options[SelectorDeColoresPunto.value].text ==
"Negro")
        {
            ColorNegro();
        }
        #endregion
        Punto.transform.position = OrdCoord.transform.position;
        Punto.transform.rotation = OrdCoord.transform.rotation;
        creadorDePuntos.cordenadas.x = float.Parse(CordenadaA.text);
        creadorDePuntos.cordenadas.z = float.Parse(CordenadaB.text);
        creadorDePuntos.cordenadas.y = float.Parse(CordenadaC.text);
        creadorDePuntos.NombreGrafico.text = NombrePunto.text;
        creadorDePuntos.colorPunto = ColorGeneral;
    }

    #region Colores

    public void ColorAzul()
    {
        ColorGeneral = Color.blue;

    }
    public void ColorRojo()
    {
        ColorGeneral = Color.red;

    }
    public void ColorNegro()
    {
        ColorGeneral = Color.black;

    }
    public void ColorCian()
    {
        ColorGeneral = Color.cyan;
    }
    public void ColorGris()
    {
        ColorGeneral = Color.gray;
    }
    public void ColorVerde()
    {
        ColorGeneral = Color.green;
    }
    public void ColorLila()
    {
        ColorGeneral = Color.magenta;

    }
    public void ColorBlanco()
    {
        ColorGeneral = Color.white;

    }
    public void ColorAmarillo()
    {
        ColorGeneral = Color.yellow;

    }
    #endregion

    public void Ajustes()
    {
        Mostrable = true;
        if (Elemento == "Plano")
        {
            CearPlanoInspector.SetActive(true);
        }
        if (Elemento == "Recta")
        {
            CearRectaInspector.SetActive(true);
        }
        if(Elemento == "Punto")
        {
            CearPuntoInspector.SetActive(true);
        }
    }
    public void Acceptar()
    {
        Mostrable = false;
        CearPlanoInspector.SetActive(false);
        CearRectaInspector.SetActive(false);
        CearPuntoInspector.SetActive(false);
    }
}
```

Esta script se encarga de calcular y representar un plano acorde con los datos que ha impuesto el usuario, también almacena datos necesarios para calcular la intersección de planos.

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

public class CreadorDePlanos : MonoBehaviour
{
    public GameObject OrigenDeCordenades;
    public float A;
    public float B;
    public float C;
    public float D;

    public Vector3 centroTriangulo;

    public float escala;

    public Vector3 punto1;
    public Vector3 punto2;
    public Vector3 punto3;

    public Vector3 puntoX;
    public Vector3 puntoY;
    public Vector3 puntoZ;

    public Vector3 puntDeTallX;
    public Vector3 puntDeTallY;
    public Vector3 puntDeTallZ;

    public Vector3 punto1b;
    public Vector3 punto2b;
    public Vector3 punto3b;

    public Vector3 proba;

    public Color colorPlano;

    Plane plano;

    GameObject origendeCordenadas;

    Vector3[] vertex;
    int[] triangles;

    float ancho;
    float altura;

    MeshCollider col;

    Mesh mesh;
    private void OnEnable()
    {
        mesh = gameObject.GetComponent<MeshFilter>().mesh;
        mesh.Clear();
        OrigenDeCordenades =
GameObject.FindGameObjectWithTag("OrigenDeCordenadas");
        col = gameObject.GetComponent<MeshCollider>();
        gameObject.transform.parent = null;
    }
    private void Update()
    {
        float e = escala;
        float s = e;
        if (A == 0)
        {
            if (e<1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }

            if (B > 0 && C > 0)
            {
                punto1 = new Vector3(0, ((-B * e) - D) / C, e);
                punto2 = new Vector3(s, s, (-C * s - D) / B);
                punto3 = new Vector3(-s, s, (-C * s - D) / B);
            }
            if (C < 0)
            {
                punto1 = new Vector3(0, e, ((-B)*e-D)/C);
                punto2 = new Vector3(-s, (C * s - D) / B, -s);
                punto3 = new Vector3(s, (C * s - D) / B, -s);
            }
            if (B < 0)
            {
                punto1 = new Vector3(0, s, ((-B) * s - D) / C);
                punto2 = new Vector3(-s, (C * s - D) / B, -s);
                punto3 = new Vector3(s, (C * s - D) / B, -s);
            }
            if (B < 0 && C < 0)
            {
                punto1 = new Vector3(0, s, ((-C * s) - D) / B);
                punto2 = new Vector3(s,-s,(C*s-D)/B);
                punto3 = new Vector3(-s,-s,(C*s-D)/B);
            }
        }
        if (B == 0)
        {
            if (e < 1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }
            if (A > 0 && C > 0)
            {
                punto1 = new Vector3((C*s-D)/A,-s,0);
                punto2 = new Vector3(-s,(C*s-D)/A,-s);
                punto3 = new Vector3(-s,(C*s-D)/A,s);
            }
            if (A < 0)
            {
                punto1 = new Vector3(-e, (-C * e - D) / A,0) ;
                punto2 = new Vector3(s, (-A * s - D) / -C, s);
                punto3 = new Vector3(s, (-A*s - D) / -C, -s);
            }
            if (C < 0)
            {
                punto1 = new Vector3(s,(-A*s - D) / C,-s);
                punto2 = new Vector3(s, (-A * s - D) / C, s);
                punto3 = new Vector3((C * e - D) / A, -e,0);
            }
            if (A < 0 && C < 0)
            {
                punto1 = new Vector3((C * e - D) / A, -e, 0);
                punto2 = new Vector3(-s, (C * s - D) / A, -s);
                punto3 = new Vector3(-s, (C * s - D) / A, s);
            }
        }
        if (C == 0)
        {
            if (e < 1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }
            if (A > 0 && B > 0)
            {
                punto1 = new Vector3((B * s - D) / A, -s, 0);
                punto2 = new Vector3(-s, (B * s - D) / A, -s);
                punto3 = new Vector3(-s, (B * s - D) / A, s);
            }
            if (A < 0)
            {
                punto1 = new Vector3(-e, (-B * e - D) / A, 0);
                punto2 = new Vector3(s, (-A * s - D) / -B, s);
                punto3 = new Vector3(s, (-A * s - D) / -B, -s);
            }
            if (B < 0)
            {
                punto1 = new Vector3(s, (-A * s - D) / B, -s);
                punto2 = new Vector3(s, (-A * s - D) / B, s);
                punto3 = new Vector3((B * e - D) / A, -e, 0);
            }
            if (A < 0 && B < 0)
            {
                punto1 = new Vector3((B * e - D) / A, -e, 0);
                punto2 = new Vector3(-s, (B * s - D) / A, -s);
                punto3 = new Vector3(-s, (B * s - D) / A, s);
            }
        }
        if (A == 0 && C == 0)
        {
            if (e < 1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }
            punto1 = new Vector3(s, D, s);
            punto2 = new Vector3(-s, D, s);
            punto3 = new Vector3(s, D, -s);
        }
        if (A == 0 && B == 0)
        {
            if (e < 1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }
            punto1 = new Vector3(D, s, s);
            punto2 = new Vector3(D, -s, s);
            punto3 = new Vector3(D, s, -s);
        }
        if (C == 0 && B == 0)
        {
            if (e < 1)
            {
                s = 1;
            }
            else
            {
                s = e;
            }
            punto1 = new Vector3(s, s, D);
            punto2 = new Vector3(s, -s, D);
            punto3 = new Vector3(-s, s, D);
        }
        if (A == 0 && B == 0 && C == 0)
        {
            punto1 = new Vector3(0,0,0);
            punto2 = new Vector3(0, 0, 0);
            punto3 = new Vector3(0, 0, 0);
        }
        if (A < 0&&C != 0 && B != 0)
        {
            punto1 = new Vector3(((-B * e - C * e - D) / A), e, e);
            punto2 = new Vector3(-e, (-A * -e - C * e - D) / B, e);
            punto3 = new Vector3(-e, e, (-A * -e - B * e - D) / C);
        }
        if (B < 0&&A != 0 && C != 0)
        {
            punto1 = new Vector3(((-B * -e - C * e - D) / A), -e, e);
            punto2 = new Vector3(e, (-A * e - C * e - D) / B, e);
            punto3 = new Vector3(e, -e, (-A * e - B * -e - D) / C);
        }
        if (C < 0&& A!=0&&B!=0)
        {
            punto1 = new Vector3(((-B * e - C * -e - D) / A), e, -e);
            punto2 = new Vector3(e, (-A * e - C * -e - D) / B, -e);
            punto3 = new Vector3(e, e, (-A * e - B * e - D) / C);
        }
        if (A < 0 && B < 0&&C!=0)
        {
            punto1 = new Vector3(((-B * -e - C * e - D) / A), -e, e);
            punto2 = new Vector3(-e, (-A * -e - C * e - D) / B, e);
            punto3 = new Vector3(-e, -e, (-A * -e - B * -e - D) / C);
```

```
        }
        if (A < 0 && C < 0&&B!=0)
        {
            punto1 = new Vector3(((-B * e - C * -e - D) / A), e, -e);
            punto2 = new Vector3(-e, (-A * -e - C * -e - D) / B, -e);
            punto3 = new Vector3(-e, e, (-A * -e - B * e - D) / C);
        }
        if (B < 0 && C < 0&&A!=0)
        {
            punto1 = new Vector3(((-B * -e - C * -e - D) / A), -e, -e);
            punto2 = new Vector3(e, (-A * e - C * -e - D) / B, -e);
            punto3 = new Vector3(e, -e, (-A * e - B * -e - D) / C);
        }
        if (A > 0&&B>0&&C>0)
        {
            punto1 = new Vector3(((-B * e - C * e - D) / A), e, e);
            punto2 = new Vector3(e, (-A * e - C * e - D) / B, e);
            punto3 = new Vector3(e, e, (-A * e - B * e - D) / C);
        }
        if (A < 0 && B < 0 && C < 0)
        {
            punto1 = new Vector3(((-B * e - C * e - D) / A), e, e);
            punto2 = new Vector3(e, (-A * e - C * e - D) / B, e);
            punto3 = new Vector3(e, e, (-A * e - B * e - D) / C);
        }

        punto1b = punto1 / 10;
        punto2b = punto2 / 10;
        punto3b = punto3 / 10;

        MakeMeshData();

        CreatMesh();

        ColorearPlano();
    }
    void MakeMeshData()
    {
        vertex = new Vector3[]
        {
           punto1b,
           punto2b,
           punto3b,
           punto3b,
           punto2b,
           punto1b,
        };
        triangles = new int[] { 0, 1, 2, 3, 4, 5 };
        Vector3[] normals = new Vector3[4]
        {
            -Vector3.forward,
            -Vector3.forward,
            -Vector3.forward,
            -Vector3.forward
        };
        mesh.normals = normals;

        Vector2[] uv = new Vector2[4]
        {
            new Vector2(0, 0),
            new Vector2(1, 0),
            new Vector2(0, 1),
            new Vector2(1, 1)
        };
        mesh.uv = uv;

        col.sharedMesh = mesh;
    }

    void ColorearPlano()
    {
        Vector3[] vertices = mesh.vertices;
        Color[] colors = new Color[vertices.Length];

        for (int i = 0; i < vertices.Length; i++)
            colors[i] = colorPlano;

        mesh.colors = colors;
    }
    private void CreatMesh()
    {

        mesh.vertices = vertex;
        mesh.triangles = triangles;
    }
}
```

Esta script se encarga de calcular y representar una recta acorde con los datos que ha impuesto el usuario.

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

public class CradorRecta : MonoBehaviour
{
    public GameObject OrigenDeCordenades;
    LineRenderer linea;
    public float escala;
    public Vector3 puntoX;
    public Vector3 punto;
    public Vector3 VectorDirector;
    Vector3 puntoXp;
    public Color colorRecta;
    private void OnEnable()
    {
        OrigenDeCordenades = 
GameObject.FindGameObjectWithTag("OrigenDeCordenadas");
    }
    private void Update()
    {
         if (OrigenDeCordenades == null)
         {
            OrigenDeCordenades = 
GameObject.FindGameObjectWithTag("OrigenDeCordenadas");
         }
        gameObject.transform.position = OrigenDeCordenades.transform.position;
        gameObject.transform.rotation = OrigenDeCordenades.transform.rotation;
        if(linea == null)
        {
            linea = gameObject.GetComponent<LineRenderer>();
        }
        float n = escala;
        puntoX = punto + VectorDirector * n;
        puntoXp = punto + (VectorDirector * (n-1)) * -1;
        DibujarLinea();
    }
    void DibujarLinea()
    {
        Vector3 p1p = transform.localToWorldMatrix * new Vector4((punto.x) / 10, 
(punto.y) / 10, (punto.z) / 10, 1);
        Vector3 p2p = transform.localToWorldMatrix * new Vector4(puntoX.x / 10, 
puntoX.y / 10, puntoX.z / 10, 1);
        Vector3 p3p = transform.localToWorldMatrix * new Vector4(puntoXp.x / 10, 
puntoXp.y / 10, puntoXp.z / 10, 1);
        linea.SetPosition(1, p1p);
        linea.SetPosition(0, p2p);
        linea.SetPosition(2, p3p);
        linea.startColor = colorRecta;
        linea.endColor = colorRecta;
    }
}
```

Esta script se encarga de calcular y representar un punto acorde con los datos que ha impuesto el usuario.

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

public class CreadorDePuntos : MonoBehaviour
{
    public Vector3 cordenadas;
    public GameObject puntoGrafico;
    public TextMeshPro NombreGrafico;

    [HideInInspector]
    public Color colorPunto;

    Renderer rendererGraphics;
    void OnEnable()
    {
        puntoGrafico.SetActive(true);
        rendererGraphics = puntoGrafico.GetComponent<Renderer>();
        gameObject.transform.parent = null;
    }

    // Update is called once per frame
    void Update()
    {
        if(puntoGrafico.activeSelf == false)
        {
            puntoGrafico.SetActive(true);
        }
        puntoGrafico.transform.position = transform.position + (cordenadas / 10);
        if(rendererGraphics == null)
        {
            rendererGraphics = puntoGrafico.GetComponent<Renderer>();
        }
        if(rendererGraphics.material.color != colorPunto)
        {
            rendererGraphics.material.color = colorPunto;
        }
        if(NombreGrafico.color != colorPunto)
        {
            NombreGrafico.color = colorPunto;
        }
    }
}
```

Esta script se encarga de gestionar la UI usada para registrar las intersecciones.

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

public class AdminInterjecciones : MonoBehaviour
{
    public GameObject[] botonPlanos;
    public bool activarBuscaDeInterject = false;
    public List<GameObject> butPlan;
    void Start()
    {
        butPlan = new List<GameObject>();
    }

    // Update is called once per frame
    void Update()
    {
        butPlan.AddRange(GameObject.FindGameObjectsWithTag("BotonInterjector"));
        //if (activarBuscaDeInterject == true)
        //{
        //for (int i = 0; i < botonPlanos.Length; i++)
        //{
        //botonPlanos[i].SetActive(true);
        //}
        //}
        //else
        //{
        if (activarBuscaDeInterject == true)
        {
            for (int i = 0; i < botonPlanos.Length; i++)
            {
                botonPlanos[i].SetActive(false);
            }
        }
        //}
    }
}
```

Esta script se encarga de almacenar los componentes de los planos.

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

public class Interjec2 : MonoBehaviour
{
    public float A;
    public float B;
    public float C;
    public float D;
    public float pA;
    public float pB;
    public float pC;
    public float pD;
    public TabManager tbm;

    private void OnEnable()
    {
        A = 3.45879f;
        B = 3.45879f;
        C = 3.45879f;
        D = 3.45879f;
        pA = 3.45879f;
        pB = 3.45879f;
        pC = 3.45879f;
        pD = 3.45879f;
        tbm = GameObject.FindObjectOfType<TabManager>();
        tbm.RectaDetectada = false;
    }

    // Update is called once per frame
    void Update()
    {
        if(pD!= 3.45879f && tbm.RectaDetectada==false)
        {
            tbm.CrearRecta();
            tbm.IndetectarIntegracion();
        }
    }
}
```

Esta script se encarga de activar la intersección y pasarla a la script anterior donde se hacen los cálculos.

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

public class Interaccion : MonoBehaviour
{
    public AdminElemento adminElemento;
    public float A;
    public float B;
    public float C;
    public float D;
    public Interjec2 interscript;
    private void Update()
    {
        A = float.Parse(adminElemento.ComponenteADelPlano.text);
        B = float.Parse(adminElemento.ComponenteCDelPlano.text);
        C = float.Parse(adminElemento.ComponenteBDelPlano.text);
        D = float.Parse(adminElemento.ComponenteDDelPlano.text);
        if(interscript== null)
        {
            interscript=FindObjectOfType<Interjec2>();
        }
    }

    public void ActivarInterjeccon()
    {
        if(interscript.A == 3.45879f)
        {
            interscript.A = A;
        }
        else
        {
            interscript.pA = A;
        }
        if (interscript.B == 3.45879f)
        {
            interscript.B = B;
        }
        else
        {
            interscript.pB = B;
        }
        if (interscript.C == 3.45879f)
        {
            interscript.C = C;
        }
        else
        {
            interscript.pC = C;
        }
        if (interscript.D == 3.45879f)
        {
            interscript.D = D;
        }
        else
        {
            interscript.pD = D;
        }
        gameObject.SetActive(false);
    }

}
```