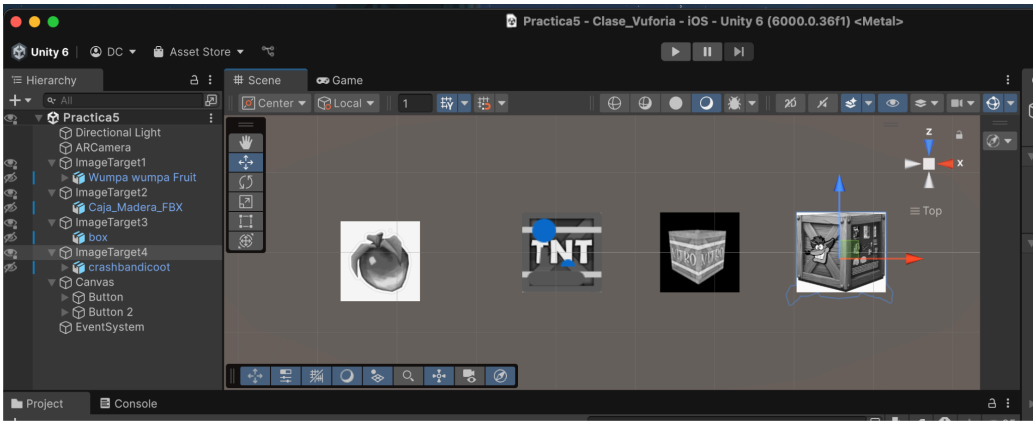
Práctica #5 Flores Cruz Daniel

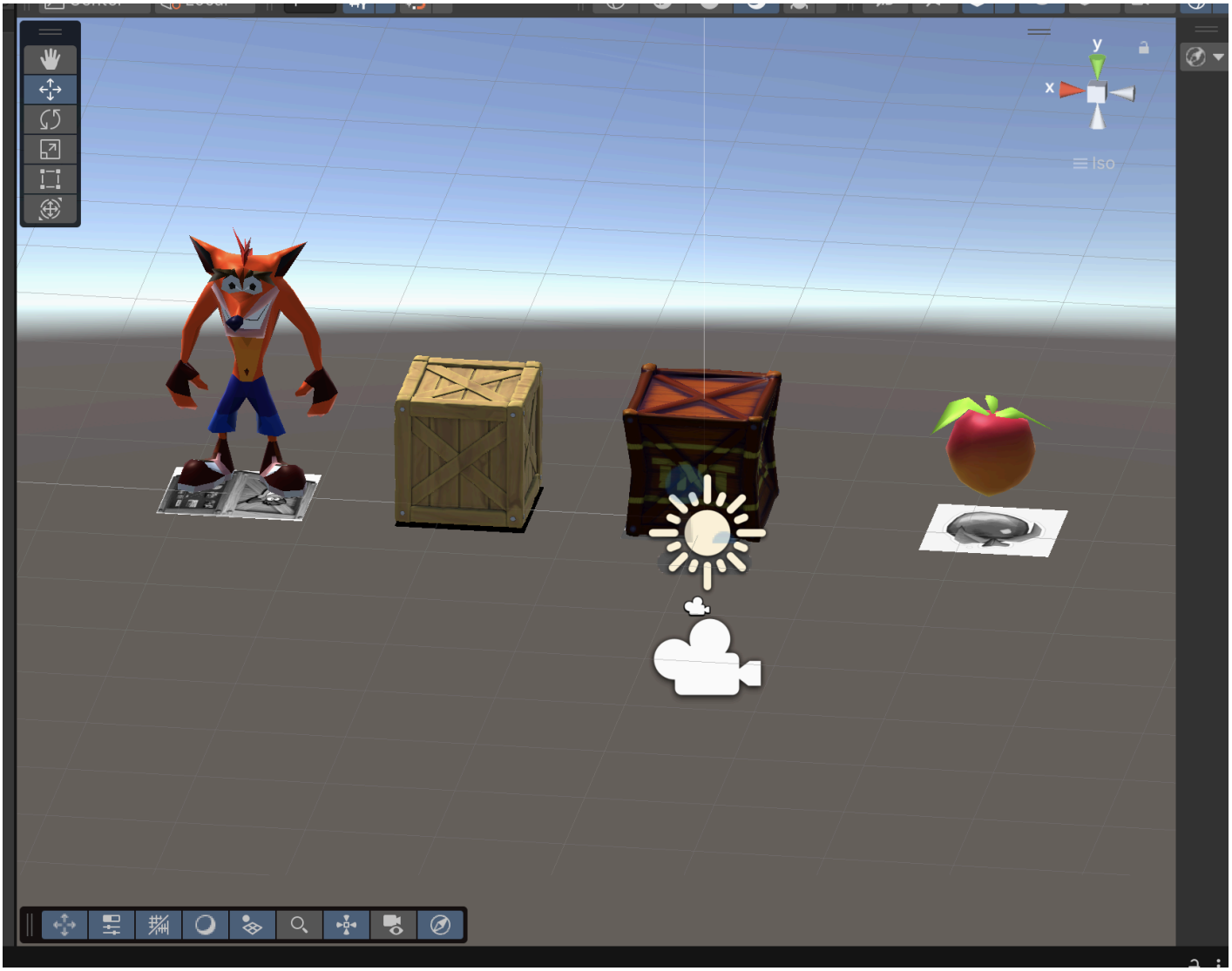
Fecha de entrega : 10 de Abril del 2025 316209513

El objetivo de esta practica es crear una narrativa interactiva no lineal mediante la integración de *Image Targets* de Vuforia en Unity, combinando técnicas de realidad aumentada con un sistema de control basado en interfaces de usuario. A diferencia de las practicas pasadas donde la detección de marcadores activa automáticamente la visualización de contenido asociado, esta implementación introduce un esquema de interacción bidireccional que permite un flujo narrativo dinámico y controlado por el usuario.

Tenemos dos mecanismos principales:

1. **Revelación controlada de elementos virtuales**
   Mediante un esquema de activación manual, los modelos asociados a cada *Image Target* permanecen ocultos hasta que el usuario interactúa con el botón "Mostrar".
2. **Sistema de transición espacial entre marcadores**
   Al activar la función "Avanzar", el sistema calcula la posición relativa de todos los *Image Targets*detectados y genera trayectorias interpoladas para los modelos visibles, los cuales se desplazan de manera fluida hacia el siguiente marcador en la secuencia de detección. Esta transición se implementa mediante corrutinas en Unity que utilizan interpolación lineal (*Vector3.Lerp*).




y
x
Iso

Vamos a tener una narración donde el objetivo es evitar la **TNT** para que crash no muera , y el chiste es solo deslizarse hacia las frutas.para eso vamos a usar el siguiente codigo:

Codigo

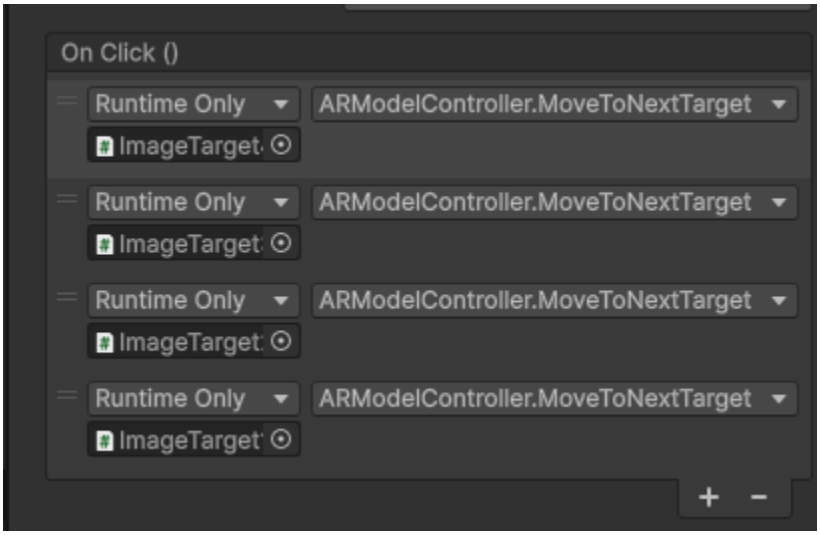
Al principio de la escena utilicé el método Start() para definir los modelos correspondientes a cada marcador (Image Target), estableciendo una relación entre ambos y asegurándome de que todos los modelos comenzaran desactivados. Esto me permitió mantener la escena limpia y solo mostrar los objetos cuando fueran realmente detectados.

Users > ada > AR > Clase_Vuforia > Assets > Move.cs

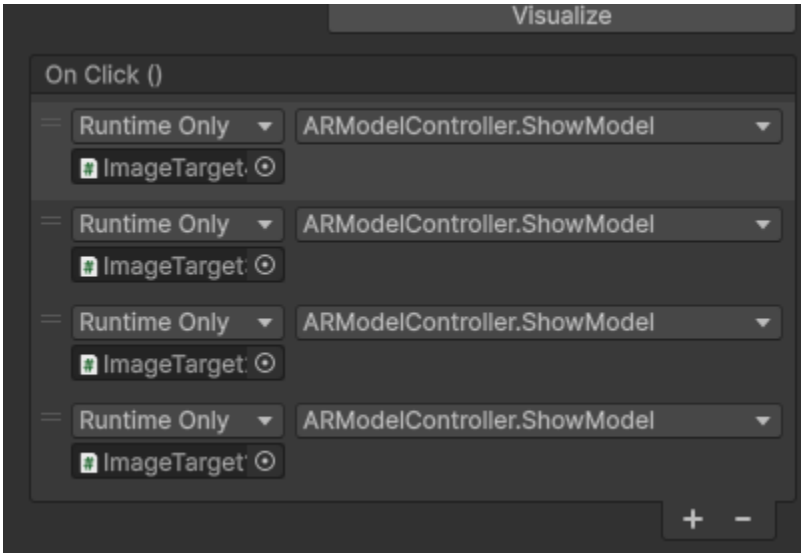
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class Move : MonoBehaviour
{
    public GameObject[] models;
    public ObserverBehaviour[] ImageTargets;
    public float speed = 1.0f;

    private bool isMoving = false;
    private List<ObserverBehaviour> currentDetectedTargets = new List<ObserverBehaviour>();
    private Dictionary<ObserverBehaviour, GameObject> targetModelMap = new Dictionary<ObserverBehaviour, GameObject>();

    void Start()
    {
        // Inicializar el mapeo
        for (int i = 0; i < ImageTargets.Length; i++)
        {
            if (i < models.Length && models[i] != null)
            {
                targetModelMap[ImageTargets[i]] = models[i];
                models[i].SetActive(false);
            }
        }
    }

    void Update()
    {
        currentDetectedTargets = GetDetectedTargets();
    }

    public void ShowAllDetectedModels()
    {
        foreach (var target in currentDetectedTargets)
        {
            if (targetModelMap.ContainsKey(target))
            {
                targetModelMap[target].SetActive(true);
            }
        }
    }
```

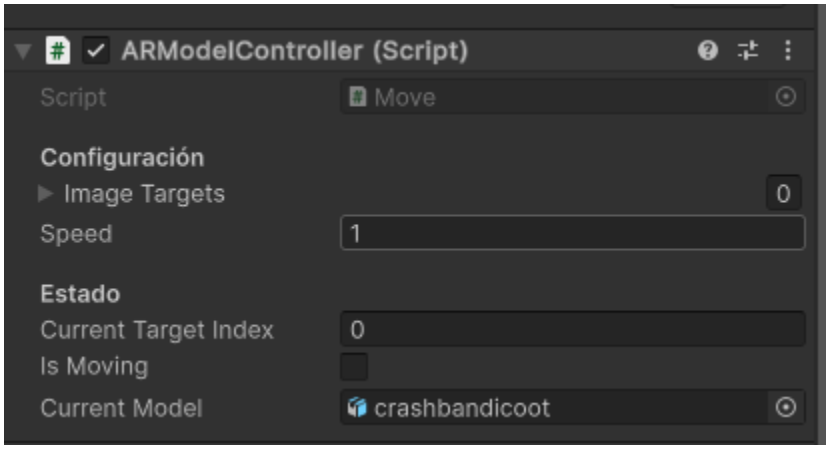
A través del método Update(), logré mantener un seguimiento constante de qué marcadores estaban siendo rastreados por Vuforia en tiempo real. Esto lo hice consultando el estado de cada ObserverBehaviour y almacenando únicamente aquellos que estuvieran activos (es decir, en estado TRACKED o EXTENDED_TRACKED).

Luego, mediante la función ShowAllDetectedModels(), logré que se activaran los modelos correspondientes a los marcadores detectados. Esto permitió que, en el momento en que un marcador apareciera en cámara, su modelo asociado se mostrara automáticamente en la escena.
Para agregar interactividad, desarrollé el método MoveAllToNextMarker(), el cual lanza una corrutina que se encarga de mover los modelos visibles hacia la posición del siguiente marcador detectado. Esta animación de movimiento se realiza suavemente usando interpolación lineal (Vector3.Lerp) y solo se ejecuta si hay más de un marcador visible, evitando comportamientos innecesarios.




```
private IEnumerator MoveAllModels()
{
    isMoving = true;

    List<GameObject> visibleModels = new List<GameObject>();
    foreach (var target in currentDetectedTargets)
    {
        if (targetModelMap.ContainsKey(target) && targetModelMap[target].activeSelf)
        {
            visibleModels.Add(targetModelMap[target]);
        }
    }

    if (visibleModels.Count == 0)
    {
        isMoving = false;
        yield break;
    }

    foreach (var model in visibleModels)
    {
        ObserverBehaviour currentTarget = FindCurrentTarget(model);
        if (currentTarget == null) continue;

        int currentIndex = currentDetectedTargets.IndexOf(currentTarget);
        if (currentIndex < 0) continue;

        int nextIndex = (currentIndex + 1) % currentDetectedTargets.Count;
        ObserverBehaviour nextTarget = currentDetectedTargets[nextIndex];

        Vector3 startPosition = model.transform.position;
        Vector3 endPosition = nextTarget.transform.position;

        float journey = 0;
        while (journey <= 1f)
        {
            journey += Time.deltaTime * speed;
            model.transform.position = Vector3.Lerp(startPosition, endPosition, journey);
            yield return null;
        }

        UpdateModelAssociation(model, currentTarget, nextTarget);
    }

    isMoving = false;
}
```





Durante cada movimiento, me aseguré de mantener la relación entre cada modelo y su nuevo marcador, actualizando el mapeo en el diccionario que uso para asociar modelos con targets. Con esto, cada vez que un modelo se mueve, queda correctamente vinculado al nuevo marcador al que se desplazó.

```
// Método único GetDetectedTargets
private List<ObserverBehaviour> GetDetectedTargets()
{
    List<ObserverBehaviour> detectedTargets = new List<ObserverBehaviour>();
    foreach (ObserverBehaviour target in ImageTargets)
    {
        if (target != null && (target.TargetStatus.Status == Status.TRACKED ||
           target.TargetStatus.Status == Status.EXTENDED_TRACKED))
        {
            detectedTargets.Add(target);
        }
    }
    return detectedTargets;
}

private ObserverBehaviour FindCurrentTarget(GameObject model)
{
    foreach (var pair in targetModelMap)
    {
        if (pair.Value == model) return pair.Key;
    }
    return null;
}

private void UpdateModelAssociation(GameObject model, ObserverBehaviour oldTarget, ObserverBehaviour newTarget)
{
    if (targetModelMap.ContainsKey(oldTarget))
    {
        targetModelMap.Remove(oldTarget);
    }
    targetModelMap[newTarget] = model;
}
```

Resultados:

Al presionar el boton de la derecha se muestra el siguiente image target

Si nosotros presionamos el boton de la izquierda crash se movera hacia el siguiente image target