🧮 Proyecto AR en Unity 6.2 para Android

🧰 Configuración Inicial

🎯 Objetivo

Desarrollar una experiencia de Realidad Aumentada modular y documentada, compatible con Android 10, utilizando Unity 6.2, AR Foundation y ARCore.

🧱 Entorno de Desarrollo

🧩 Unity

- **Versión:** Unity 6.2
- **Template:** AR Mobile
- **Nombre del proyecto:** `UnityARAndroid`
- **Ubicación:** Carpeta dedicada con control de versiones (Git recomendado)

📦 Módulos instalados desde Unity Hub

- Android Build Support
- Android SDK & NDK Tools
- OpenJDK
- ARCore XR Plugin (desde Package Manager)

🧮 Dispositivo de Pruebas

🔍 Información del dispositivo

- **Modelo:** Redmi Note 8 Pro
- **Versión de Android:** 10 (QP1A.190711.020)
- **Modo desarrollador:** Activado

⚙️ Configuraciones necesarias

- ⬛ Activar opciones de desarrollador
- ⬛ Activar depuración USB

- ☑ Permitir instalación de apps vía USB
- ☑ Conectar por cable y verificar con `adb devices`

🧩 Referencia al Template AR Mobile

🎯 Objetivo

Analizar la estructura del template oficial AR Mobile de Unity para extraer buenas prácticas y aplicarlas en una escena independiente personalizada.

📁 Estructura del Template AR Mobile

```
Assets/
└── AR Template/
    ├── Scenes/
    ├── Prefabs/
    ├── Scripts/
    ├── Materials/
    ├── UI/
    └── Resources/
```

👋 Bienvenida al Proyecto AR Mobile Template

Esta plantilla de proyecto busca ayudarte a crear experiencias de AR móviles atractivas. Proporciona un proyecto preconfigurado con paquetes de XR y componentes básicos para interacciones y la interfaz de usuario.

⚙️ Configuración Inicial

Antes de comenzar, selecciona la plataforma móvil en la que planeas realizar la implementación navegando a

```
File > Build Profile
```

y selecciona tu plataforma de destino.

🧩 Instalación de Plataforma

Si tu plataforma no aparece en la lista, asegúrate de tener instalado el módulo de la plataforma correcto navegando a tu proyecto desde la pestaña

`Unity Hub Projects > Editor Version > Add Platform`

, luego selecciona el Platform Module de destino de la lista.

📚 Referencia

Consulta la **Guía de inicio rápido** para obtener información más detallada sobre el contenido y la configuración utilizadas en esta plantilla.

🚀 Build de la escena `SampleScene` en Android

🎯 Objetivo

Compilar y desplegar la escena `SampleScene` en el dispositivo Android (Redmi Note 8 Pro) para validar la configuración inicial del proyecto AR.

🧱 Requisitos previos

- ⬛ Proyecto configurado con Unity 6.2
- ⬛ `SampleScene` abierta y funcional
- ⬛ ARCore XR Plugin instalado
- ⬛ Dispositivo conectado por USB y reconocido por `adb`
- ⬛ Modo desarrollador y depuración USB activados

⚙️ Configuración de Build

🔧 Ajustes en `Build Profiles`

1. Ir a `File > Build Profiles`
2. Seleccionar plataforma: `Android`
3. Añadir la escena activa: `SampleScene`
4. Verificar que esté marcada en la lista de escenas

5. Pongo `Run Device` a mi dispositivo
6. Activado `Development Build`
7. Click en `Switch Platform`

📦 Ajustes en `Project Settings`

1. Click en `Project Settings`
2. Click en `Player`
3. En `Other Settings` :
   - `Graphics API` : solo `OpenGLES3`
   - `Minimum API Level` : Android 10.0 (API level 29)
   - `Target API Level` : Android 10.0 (API level 29)
   - `Package Name` : `com.jmcp.UnityARAndroid`
   - `Scripting Backend` : IL2CPP
   - `ARM64` : activado
4. En `XR Plug-in Management` :
   - Activar `Google ARCore` , en Plug-in Providers, para Android

📲 Proceso de Build

1. Pulsar `Build And Run` , crear una nueva carpeta en el proyecto llamada `Builds` , poner nombre a la `.apk` y guardar.
2. Actualizar SDK, si te lo pide.
3. Comprobar que la aplicación funciona correctamente en el dispositivo. Si es así, bravo.

📁 Vamos a aplicar este conocimiento para desarrollar nuestra propia versión del Template, en la aplicación ARAndroid.

⚠️ Advertencia importante sobre paquetes inmutables

Unity puede mostrar el siguiente mensaje si se modifica un archivo dentro de un paquete gestionado por el Package Manager:

```
The following asset(s) located in immutable packages were unexpectedly altered. Making chang
```

Esto significa que se ha editado un archivo que forma parte de un paquete inmutable (por ejemplo, `AR Foundation`). Estos archivos **no deben modificarse directamente**, ya que cualquier cambio puede perderse al actualizar, reinstalar o limpiar la caché del proyecto.

✅ Para evitar este problema:

- Copia el asset que deseas modificar a tu carpeta `Assets/` del proyecto.
- Trabaja sobre esa copia y no sobre el original.

❌ No edites archivos dentro de `Packages/`, incluso si Unity lo permite temporalmente.

Mantener esta práctica ayuda a evitar errores inesperados y asegura la estabilidad del proyecto.

🛠️ Solución al error "State comes from an incompatible keyword space" en Unity XR Simulator

Al trabajar con el XR Simulator en Unity usando URP (Universal Render Pipeline), puede aparecer el siguiente error:

> **State comes from an incompatible keyword space**
> Esto ocurre cuando Unity intenta renderizar un material con un shader que no es compatible con el pipeline activo.

🔍 Causa

El error se debe a que el sistema de renderizado intenta usar un shader del Built-in Render Pipeline (`Simulation/Standard Lit`) mientras el proyecto está configurado para URP (`Universal Render Pipeline/Lit`). Esto genera un conflicto de keywords internos.

✅ Solución efectiva

1. **Limpiar la caché de iluminación global (GI Cache):**

   - Ir a `Edit > Preferences > GI Cache`
   - Hacer clic en **Clear Cache**

   Esto fuerza a Unity a regenerar los datos de iluminación y materiales, eliminando estados incompatibles que pueden causar errores en el Simulator.

2. **Verificar que todos los materiales usen shaders compatibles con URP:**

- Usar shaders como `Universal Render Pipeline/Lit`, `Simple Lit`, o `Unlit`.
- Evitar `Standard`, `Legacy`, o shaders personalizados no adaptados.

3. **Convertir materiales si el proyecto fue migrado:**

   - Ir a `Edit > Rendering > Materials > Convert Selected Built-in to URP`
   - Seleccionar los materiales y convertirlos a URP.

🧪 Resultado

Después de limpiar la GI Cache, el error desaparece y el XR Simulator funciona correctamente con los materiales URP.

> 💡 Recomendación: Siempre revisar los materiales de los prefabs que se usan en simulación para evitar conflictos entre pipelines.

🎮 Configuración de XRI Default Input Actions

Para habilitar interacciones en XR con el XR Interaction Toolkit, se utiliza el asset `XRI Default Input Actions`.

✅ Pasos de configuración

1. Importar el paquete `XR Interaction Toolkit` desde el Package Manager.
2. Importar el sample `XRI Default Input Actions`.
3. Asignar el asset en el componente `XR Input` del XR Rig.
4. Añadir el prefab `XR Device Simulator` si se trabaja en simulación.
5. Verificar y personalizar los bindings de entrada según las necesidades del proyecto.

> 💡 Este sistema permite manejar interacciones como selección, activación, movimiento y rotación en entornos XR.

📁 Jerarquía de la Escena en Unity: ARAndroid

```
ARAndroid
├── Directional Light
├── AR Session
│   ├── AR Session
│   └── AR Input Manager
├── XR Interaction Manager
│   └── XR Interaction Manager (Script)
```

```
├── XR Origin (Mobile AR)
│   ├── XR Origin (Script)
│   ├── AR Plane Manager
│   ├── AR Raycast Manager
│   ├── AR Anchor Manager
│   └── Camera Offset
│       └── Main Camera
│           ├── Camera
│           ├── Audio Listener
│           ├── AR Camera Manager
│           ├── AR Camera Background
│           ├── Tracked Pose Driver (Input System)
│           └── Universal Additional Camera Data (Script)
├── ARSystemCheck
│   ├── Input System Check (Script)
│   ├── AR Availability Checker (Script)
│   └── Material Checker (Script)
├── ARReport
│   ├── Info DisPlay (Script)
│   └── Message Controller (Script)
├── UIAR
│   └── Canvas
│       ├── PanelOptions
│       │   ├── ButtonDelAll
│       │   │   └── TextDelAll
│       │   ├── ButtonDelFire
│       │   │   └── TextDelFire
│       │   └── CrosshairButton
│       │       └── CrosshairButtonText
│       ├── PanelInfo
│       │   └── TextInfo
│       └── MiraTelescopica
├── EventSystem
└── ARSelectionManager
    ├── AR Object Placer And Selector (Script)
    └── AR Gesture Detector (Script)
```

📁 Jerarquía del Prefad usado: TheTrain (cofeccionado en Blender. Apuntes del modulo en documento aparte.)

```
TheTrain (Prefad Asset)
├── Tag (Interactable)
├── Mesh Filter
├── Mesh Renderer
```

```
│   └── Materials (Element 0)
├── Box Collider
├── Rigidbody
├── XR Grab Interactable
└── AR Object Manipulator (Script)
```

📁 Scripts como componentes reseñables x GameObject

🧊 XR Interaction Manager

✅ Componentes requeridos

- XR Interaction Manager (Script)

🧊 ARSystemCheck

✅ Scripts requeridos

- Input System Check (Script)
- AR Availability Checker (Script)
- Material Checker (Script)

🧊 ARReport

✅ Scripts requeridos

- Info Display (Script)
- Message Controller (Script)

🧊 ARSelectionManager

✅ Scripts requeridos

- AR Object Placer And Selector (Script)
- AR Gesture Detector (Script)

📜 Historia cronológica de Unity Input System + AR Foundation + ARCore

Año / Versión	Input System (Unity)	AR Foundation	ARCore (Unity XR Plug-in)	Claves técnicas
2017 (Unity 2017.x)	Sistema heredado (`UnityEngine.Input`)	No existía aún AR Foundation	Primer soporte experimental de ARCore SDK para Unity	Input basado en `Input.GetTouch()` . ARCore se integraba mediante SDK externo.
2018 (Unity 2018.1)	Inicio del **nuevo Input System** (preview)	**AR Foundation 1.0** aparece como paquete	ARCore XR Plug-in inicial	AR Foundation introduce `ARSession` , `ARPlaneManager` . Input System aún en preview.
2019 (Unity 2019.x)	Nuevo Input System en preview estable	**AR Foundation 2.x**	ARCore XR Plug-in 2.x	Se añade soporte multiplataforma (ARKit + ARCore). Input System empieza a integrarse con XR.
2020 (Unity 2020 LTS)	Nuevo Input System disponible como paquete oficial (`com.unity.inputsystem`)	**AR Foundation 3.x**	ARCore XR Plug-in 3.x	Se introduce `XR Interaction Toolkit` . Input System recomendado para XR.
2021 (Unity 2021 LTS)	Input System gana soporte completo y coexistencia con sistema heredado	**AR Foundation 4.x**	ARCore XR Plug-in 4.x	Se añade soporte para **XR Origin**, mejoras en raycasting y simulación.
2022 (Unity	Input System obligatorio para XR Toolkit	**AR Foundation**	ARCore XR Plug-in 5.0	`ARSessionOrigin` deprecado →

Año / Versión	Input System (Unity)	AR Foundation	ARCore (Unity XR Plug-in)	Claves técnicas
2022.x)		**5.0**		`XROrigin`. Se integra `XR Simulation`. AR Foundation 5 requiere Input System.
2023–2025 (Unity 2023–2025)	Input System consolidado como estándar	**AR Foundation 5.x (actual)**	ARCore XR Plug-in 5.x	AR Foundation soporta: planos, imágenes, objetos, rostros, cuerpos, nubes de puntos, oclusión, participantes. Input System + EnhancedTouch habilitan gestos multitáctiles avanzados.

🌳 Arquitectura extendida: Input System + AR Foundation + ARCore + Scripts propios

```
Unity Project
├── Input System (com.unity.inputsystem)
│   ├── UnityEngine.InputSystem
│   │   ├── InputSystem.version → (usado en `InputSystemCheck`)
│   │   ├── InputAction / CallbackContext → (usado en `ARObjectPlacerAndSelector`)
│   │   └── Touchscreen / Mouse → (reemplazo del sistema heredado)
│   └── UnityEngine.InputSystem.EnhancedTouch
│       ├── Touch.activeTouches → (usado en `ARGestureDetector`)
│       ├── EnhancedTouchSupport.Enable() → (activado en `ARGestureDetector`)
│       └── Gestos multitáctiles (pinch, twist, drag)
│
├── AR Foundation (com.unity.xr.arfoundation)
│   ├── UnityEngine.XR.ARFoundation
│   │   ├── ARSession → (usado en `ARAvailabilityChecker`)
│   │   ├── ARPlaneManager → (usado en `ARObjectPlacerAndSelector`)
│   │   ├── ARRaycastManager → (usado en `ARObjectPlacerAndSelector`, `ARGestureDetector`)
│   │   ├── ARAnchorManager → (usado en `ARObjectPlacerAndSelector`)
```

```
│   │   └── XROrigin → (estructura base de la escena AR)
│   └── UnityEngine.XR.ARSubsystems
│       ├── TrackableType.PlaneWithinPolygon → (raycasts en `ARObjectPlacerAndSelector`)
│       ├── Pose → (colocación de objetos en `ARObjectPlacerAndSelector`)
│       └── ARSessionState → (validación en `ARAvailabilityChecker`)
│
├── ARCore XR Plug-in (com.unity.xr.arcore)
│   ├── Proveedor de AR Foundation para Android
│   ├── Implementa soporte nativo ARCore
│   ├── Conecta hardware → ARSubsystems
│   └── Gestiona instalación y disponibilidad de ARCore en dispositivos Android
│
└── Scripts propios (Assets/Scripts)
    ├── Diagnóstico
    │   ├── InputSystemCheck → Verifica Input System activo y versión
    │   ├── ARAvailabilityChecker → Comprueba soporte ARCore y gestiona instalación
    │   └── MaterialChecker → Sustituye materiales incompatibles con URP
    │
    ├── UI & Mensajería
    │   ├── InfoDisplay → Muestra mensajes en TextMeshPro con colores diferenciados
    │   └── MessageController → Centraliza mensajes de interacción (colocación, selección, e
    │
    ├── Interacción AR
    │   ├── ARObjectPlacerAndSelector → Coloca prefabs en planos, selecciona y elimina objet
    │   └── ARGestureDetector → Detecta gestos multitáctiles (mover, escalar, rotar)
    │
    └── Manipulación
        └── ARObjectManipulator → Script asociado a cada prefab para permitir selección y ma
```

🧭 Guía de uso: Physics, XR Toolkit e Input System en AR

Esta sección explica cuándo utilizar cada sistema de interacción en Unity para proyectos de Realidad Aumentada (AR), especialmente en dispositivos móviles Android.

⚙️ Comparativa de sistemas de interacción

Caso de uso	Physics (Raycast) ✅	XR Interaction Toolkit ✅
Detectar colisión con objetos 3D	✅ Sí	❌ No (usa colliders pero no raycast directo)
Selección por toque en pantalla	✅ Sí (`ScreenPointToRay`)	✅ Sí (`XRRayInteractor`)

Caso de uso	Physics (Raycast) ✅	XR Interaction Toolkit ✅
Interacción con gestos AR	❌ No	✅ Sí (`ARGestureInteractor`)
Manipulación física (fuerzas, gravedad)	✅ Sí	❌ No (no gestiona física)
Interacción con controladores XR	❌ No	✅ Sí (ray, direct, socket interactors)

🎮 Uso del nuevo Input System con XR Toolkit

El nuevo Input System es obligatorio cuando usas XR Interaction Toolkit. Permite:

- Mapear acciones como `Select` , `Activate` , `Hover` a gestos o botones.
- Usar `InputActionAsset` para definir controles multiplataforma.
- Integrar gestos táctiles con `EnhancedTouchSupport` .

Ejemplo típico: definir una acción `Tap` que se activa con un toque en pantalla y se conecta a un `XRRayInteractor` .

🖐️ Uso de XR Toolkit con Touch

En AR móvil, puedes usar el componente `AR Gesture Interactor` del XR Toolkit para traducir toques en pantalla a gestos como:

- Selección
- Escalado
- Rotación
- Traslación

Esto se logra con el `TouchscreenGestureInputController` , que convierte toques en eventos de interacción XR.

🧊 TouchscreenGestureInputController

🎯 Objetivo

Convertir los **toques en pantalla** en **eventos de interacción XR** dentro de proyectos móviles con **Unity + AR Foundation + Input System**.

Este controlador traduce gestos táctiles (tap, pinch, drag, rotate) en acciones que pueden ser entendidas por el **XR Interaction Toolkit**.

📂 Namespace y dependencias

- `UnityEngine.InputSystem` → Base del nuevo sistema de entrada.
- `UnityEngine.InputSystem.EnhancedTouch` → Soporte multitáctil avanzado.
- `UnityEngine.XR.Interaction.Toolkit` → Framework de interacción XR que recibe los eventos.

📜 Claves de uso en tu proyecto

- **Dónde lo usaste:**
  - Integrado en la escena **ARAndroid**, junto con `ARObjectPlacerAndSelector` y `ARGestureDetector`.
  - Conectado al **XR Interaction Manager** para traducir gestos en acciones XR.
- **Cómo funciona:**
  - Captura los toques desde `Touchscreen.current`.
  - Los convierte en **InputActions** (`Select`, `Activate`, `Translate`, `Rotate`, `Scale`).
  - Envía esos eventos al **XR Interaction Toolkit**, que los interpreta como interacciones sobre objetos AR.

🧠 Flujo de interacción

1. El usuario toca la pantalla → `TouchscreenGestureInputController` detecta el gesto.
2. El gesto se traduce en un **evento XR** (ej. `Select` o `Scale`).
3. El **XR Interaction Toolkit** aplica la acción al objeto seleccionado (`XR Grab Interactable`, `ARObjectManipulator`).
4. Los scripts propios (`MessageController`, `InfoDisplay`) muestran feedback en la UI.

💡 Notas técnicas

- Es obligatorio tener el **Input System activo** en `Project Settings > Player > Active Input Handling`.
- Funciona en conjunto con el asset `XRI Default Input Actions`, que define los bindings de gestos.

- Se recomienda probar en dispositivo físico Android para validar sensibilidad y precisión de los gestos.
- Complementa a `ARGestureDetector`, que implementa lógica personalizada de pinch-to-scale y twist-to-rotate.

📜 Ejemplo de integración

```
// Ejemplo conceptual de cómo se conecta
var gestureController = new TouchscreenGestureInputController();
gestureController.OnTap += () => xrInteractor.Select();
gestureController.OnPinch += (scaleFactor) => xrInteractor.Scale(scaleFactor);
gestureController.OnTwist += (angle) => xrInteractor.Rotate(angle);
```

> Nota: Unity ofrece el componente `TouchscreenGestureInputController` dentro del XR Interaction Toolkit, que convierte automáticamente los toques en pantalla en eventos XR.
> En este proyecto no se utiliza directamente, ya que la lógica de gestos se implementa manualmente en los scripts `ARObjectPlacerAndSelector` y `ARGestureDetector`, aprovechando `ARInputActions` y `EnhancedTouchSupport` para un control más granular.

📊 Comparativa: TouchscreenGestureInputController vs ARGestureDetector (implementación propia)

Aspecto	TouchscreenGestureInputController (XR Toolkit)	ARGestureDetector (Proyec
Origen	Parte del paquete **XR Interaction Toolkit**	Script propio desarrollado en C
Namespace principal	`UnityEngine.XR.Interaction.Toolkit.Inputs`	`UnityEngine.InputSystem.Enha`
Configuración	Se conecta automáticamente al `XRI Default Input Actions`	Usa `EnhancedTouchSupport` y lá manual en `Update()`
Gestos soportados	Tap, Drag, Pinch, Rotate (predefinidos)	Tap, Drag, Pinch-to-Scale, Twist Rotate (personalizados)
Flexibilidad	Limitada a bindings preconfigurados	Total control sobre sensibilidad umbrales y feedback en UI

Aspecto	TouchscreenGestureInputController (XR Toolkit)	ARGestureDetector (Proyec
Feedback al usuario	No incluye mensajes por defecto	Integrado con `MessageControl` `InfoDisplay`
Integración con XR	Directa con `XR Interaction Manager`	Se integra con `ARObjectManipu` `ARObjectPlacerAndSelector`
Ventaja principal	Rápida implementación sin código adicional	Control granular y adaptado a Android
Limitación principal	Menos personalizable, dependiente de bindings	Requiere más código y manter

🧊 ARInputActions.cs (generado desde ARInputActions.inputactions)

🎯 Objetivo

Definir y encapsular todas las **acciones de entrada táctil** del proyecto, utilizando el **nuevo Input System de Unity**.
Este archivo `.cs` se genera automáticamente por Unity a partir del asset `.inputactions`, y permite acceder a las acciones como clases tipadas y estructuradas.

📂 Origen y generación

- **Archivo fuente:** `ARInputActions.inputactions`
- **Generado por:** Unity Input System (al marcar "Generate C# Class" en el asset)
- **Clase generada:** `ARInputActions`
- **Ubicación recomendada:** Carpeta `Scripts/Input/` o `Scripts/Runtime/`

📁 Estructura del asset: `ARInputActions.inputactions`

```
## 📁 Estructura del asset: ARInputActions.inputactions

```plaintext
Action Map: ARPlayer
├── Tap
│   └── Binding: Touchscreen/PrimaryTouch/Press
├── DoubleTap
│   └── Binding: Touchscreen/Touch*/Press
│       └── Interaction: MultiTap
```
```

```
│           ├── Tap Count: 2
│           ├── Tap Spacing: 0.2
│           ├── Max Tap Duration: 0.3
│           └── Press Point: 0.1
└── TouchPosition
    └── Binding: Touchscreen/Position
```

📜 Clase generada: ARInputActions.cs

```
public class ARInputActions : IInputActionCollection2, IDisposable
{
    public InputActionMap ARPlayer;
    public InputAction Tap;
    public InputAction DoubleTap;
    public InputAction TouchPosition;
    ...
}
```

🧩 Implementacion en ARAndroid

```
private ARInputActions inputActions;

void Awake()
{
    inputActions = new ARInputActions();
}

void OnEnable()
{
    inputActions.ARPlayer.Enable();
    inputActions.ARPlayer.Tap.performed += OnTouchPerformed;
    inputActions.ARPlayer.DoubleTap.performed += ctx => OnDoubleTap();
}
```

🧠 Recomendaciones prácticas

- **Para AR móvil con gestos**: usa XR Toolkit + AR Foundation + Input System.
- **Para interacción física directa**: usa Physics con raycasts desde la cámara.
- **Para UI o botones en pantalla**: usa Input System directamente o eventos de Unity UI.

- **Evita mezclar Physics y XR Toolkit en el mismo flujo de interacción**, salvo que estés haciendo una verificación extra (por ejemplo, raycast físico para detectar objetos antes de activar una interacción XR).

📌 Notas técnicas

- `Physics.Raycast` funciona bien para detectar objetos con colliders desde la cámara.
- `XRRayInteractor` permite rayos desde controladores o desde la cámara en AR.
- `EnhancedTouchSupport` debe activarse en `Awake()` para usar gestos táctiles avanzados.
- El nuevo Input System requiere que el proyecto esté configurado con `Active Input Handling: Input System Package (New)`.

🧊 Objetivo The Train interactuable, instanciado en plano detectado: agarre, movimiento, escala, rotación y destrucción, utilizando Scripts propios.

Para que el prefad sea completamente interactuable en XR:

✅ Componentes requeridos

- `Tag Interactable`
- `Transform`
- `Mesh Filter`
- `Mesh Renderer`
- `Box Collider`
- `Rigidbody` (con `Is Kinematic`)

✅ Scripts requeridos

- `XR Grab Interactable (Script)`
- `AR Object Manipulator (Script)`
- XR Interaction Manager, en la escena

🧮 Escalado con dos dedos en dispositivos móviles (Unity 6.2)

En Unity 6.2, el XR Interaction Toolkit no incluye soporte nativo para gestos multitáctiles como pinch-to-scale en pantallas táctiles. Para habilitar esta funcionalidad en dispositivos móviles, se requiere una implementación personalizada.

🧩 Consideraciones clave

✅ Requisitos técnicos

- Unity 6.2 o superior
- Input System activo (no el sistema de entrada heredado)
- XR Interaction Toolkit instalado
- AR Foundation (opcional, si se trabaja en entornos AR)
- `Touchscreen.current` disponible para detectar gestos táctiles

🖐️ Limitaciones del XR Grab Interactable

- El componente `XR Grab Interactable` está diseñado para controladores XR (VR/AR).
- No responde a gestos táctiles como pinch o swipe en pantallas móviles.
- La opción `Two Handed Manipulation` solo funciona con dos interacciones XR simultáneas (ej. dos controladores), no con dos dedos en pantalla.

✅ Solución recomendada: Script personalizado

Se recomienda usar un script que detecte dos dedos en pantalla y escale el objeto en tiempo real: Para permitir que el prefad se escale con dos dedos en móviles.

> 💡 Recomendación: probar en dispositivo físico para validar sensibilidad del gesto.

🧪 Compatibilidad con Input System activo

Cuando el proyecto usa el nuevo Input System, no se puede usar `UnityEngine.Input`. En su lugar:

✅ Reemplazos recomendados

- `Input.touchCount` → `Touch.activeTouches.Count`
- `Input.GetTouch(0)` → `Touch.activeTouches[0]`
- `Input.GetMouseButtonDown(0)` → `Mouse.current.leftButton.wasPressedThisFrame`

🧠 Activar soporte táctil

```
EnhancedTouchSupport.Enable();
```

> 💡 Esto permite que el sistema de colocación funcione correctamente en móviles y en el editor con el nuevo Input System.

🧩 Namespaces clave en los scripts

🔧 `UnityEngine`

- **Rol:** Base de todos los componentes de Unity.
- **Uso en scripts:** GameObjects, MonoBehaviour, Transform, Debug.
- **Clave:** Es el namespace raíz, sin él no se pueden instanciar ni manipular objetos en la escena.

🔧 `UnityEngine.InputSystem`

- **Rol:** Nuevo sistema de entrada de Unity.
- **Uso en scripts:**
  - `InputSystem.version` (diagnóstico en `InputSystemCheck`).
  - `InputAction.CallbackContext` (gestión de eventos en `ARObjectPlacerAndSelector`).
- **Clave:** Sustituye al antiguo `UnityEngine.Input`, permitiendo acciones configurables y multiplataforma.

🔧 `UnityEngine.InputSystem.EnhancedTouch`

- **Rol:** Extensión del Input System para gestos multitáctiles.
- **Uso en scripts:**
  - `Touch.activeTouches` (detección de toques en `ARGestureDetector`).
  - `EnhancedTouchSupport.Enable()` / `Disable()` (activación del soporte multitáctil).
- **Clave:** Permite implementar gestos como pinch-to-scale y twist-to-rotate en móviles.

🔧 `UnityEngine.XR.ARFoundation`

- **Rol:** Framework para AR en Unity.
- **Uso en scripts:**
  - `ARSession.CheckAvailability()` (en `ARAvailabilityChecker`).
  - `ARRaycastManager.Raycast()` (en `ARObjectPlacerAndSelector`).
  - `ARPlaneManager`, `ARAnchorManager` (gestión de planos y anclajes).
- **Clave:** Es la base de la detección de planos, raycasts y anclajes en ARCore.

🔧 `UnityEngine.XR.ARSubsystems`

- **Rol:** Define tipos y estructuras de bajo nivel para AR.
- **Uso en scripts:**
  - `TrackableType.PlaneWithinPolygon` (detección de planos en raycasts).
  - `Pose` (posición y rotación de objetos instanciados).
- **Clave:** Proporciona los tipos que conectan ARFoundation con el hardware.

🔧 `TMPro`

- **Rol:** Namespace de TextMeshPro.
- **Uso en scripts:**
  - `TextMeshProUGUI` (en `InfoDisplay` y `ARObjectPlacerAndSelector`).
- **Clave:** Permite mostrar mensajes claros y estilizados en la UI.

🔧 `UnityEngine.UI`

- **Rol:** UI tradicional de Unity.
- **Uso en scripts:**
  - `RawImage` (mirilla en `ARObjectPlacerAndSelector`).
- **Clave:** Complementa TextMeshPro para elementos gráficos simples.

🧠 Conclusión técnica

Los **namespaces son las verdaderas claves** del proyecto:

- **`UnityEngine.InputSystem`** y **`EnhancedTouch`** convierten los gestos táctiles en eventos configurables.
- **`ARFoundation`** y **`ARSubsystems`** traducen esos eventos en acciones sobre planos y objetos en el mundo AR.
- **`TMPro`** y **`UnityEngine.UI`** garantizan que cada interacción tenga feedback inmediato y claro en la interfaz.

🧊 Caracteristicas y funcionalidades de los Scripts propios

🧊 InputSystemCheck (Script)

🎯 Objetivo

Validar que el **nuevo Input System** está correctamente instalado y activo en el proyecto, mostrando en consola la versión detectada.

🔧 Propiedades públicas

Este script no expone propiedades públicas, ya que su función es únicamente de verificación en tiempo de ejecución.

📂 Dependencias

- `UnityEngine`
- `UnityEngine.InputSystem`

📜 Métodos clave

- `Start()`
  - Muestra en la consola dos mensajes:
    - Confirmación de que el Input System está instalado.
    - La versión exacta del paquete Input System en uso.

🧠 Flujo de interacción

1. Se ejecuta automáticamente al iniciar la escena.
2. Envía mensajes al **Console Log** de Unity.
3. Permite verificar rápidamente que el proyecto está configurado con el **Input System Package (New)** en `Project Settings > Player > Active Input Handling`.

💡 Notas técnicas

- Este script es útil como **test de diagnóstico inicial** en proyectos AR.
- Si el log no muestra la versión, significa que el Input System no está correctamente referenciado o que el proyecto sigue usando el sistema heredado.
- Recomendación: mantener este script en un objeto vacío de la escena (`ARSystemCheck`) para centralizar validaciones.

📜 Código fuente

```
using UnityEngine;
using UnityEngine.InputSystem;
```

```
public class InputSystemCheck : MonoBehaviour
{
    void Start()
    {
        Debug.Log("Input System instalado correctamente");
        Debug.Log("Versión: " + InputSystem.version);
    }
}
```

🧊 ARAvailabilityChecker (Script)

🎯 Objetivo

Comprobar si el dispositivo soporta **ARCore** antes de iniciar la sesión de realidad aumentada.
Este script valida la disponibilidad de AR en el dispositivo Android y gestiona los estados de instalación o error.

🔧 Propiedades públicas

Este script no expone propiedades públicas directamente.
Se recomienda enlazarlo con componentes de UI para mostrar mensajes al usuario.

📂 Dependencias

- `UnityEngine`
- `UnityEngine.XR.ARFoundation`
- `System.Collections`

📜 Métodos clave

- `Start()`
  - Inicia la corrutina `CheckARAvailability()` al cargar la escena.
- `IEnumerator CheckARAvailability()`
  - Comprueba el estado de **ARSession**:
    - `None` o `CheckingAvailability` → Ejecuta `ARSession.CheckAvailability()`.
    - `Unsupported` → Muestra error en consola y en UI.
    - `NeedsInstall` → Solicita instalación de ARCore y muestra progreso.
    - `Available` → Inicializa la sesión AR.
- `ShowErrorMessage(string message)`
  - Método placeholder para mostrar mensajes de error en la UI.
- `InitializeAR()`

- Método placeholder para iniciar la experiencia AR cuando el dispositivo es compatible.

🧠 Flujo de interacción

1. Al iniciar la escena, se ejecuta la corrutina de verificación.
2. Se consulta el estado de **ARSession**.
3. Según el resultado:
   - Si no hay soporte → se informa al usuario y se detiene el flujo.
   - Si requiere instalación → se inicia el proceso de instalación de ARCore.
   - Si está disponible → se inicializa la experiencia AR.

💡 Notas técnicas

- Este script es esencial para evitar **crashes** en dispositivos que no soportan ARCore.
- Se recomienda enlazar `ShowErrorMessage()` con un **Canvas UI** para mostrar mensajes visibles al usuario.
- `InitializeAR()` puede usarse para activar managers como `ARPlaneManager`, `ARRaycastManager` o cargar prefabs iniciales.
- Ideal mantenerlo dentro de un objeto vacío en la escena (`ARSystemCheck`) junto con otros scripts de validación.

📜 Código fuente

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using System.Collections;

public class ARAvailabilityChecker : MonoBehaviour
{
    void Start()
    {
        StartCoroutine(CheckARAvailability());
    }

    IEnumerator CheckARAvailability()
    {
        // Verificar soporte AR
        if ((ARSession.state == ARSessionState.None) ||
            (ARSession.state == ARSessionState.CheckingAvailability))
        {
            yield return ARSession.CheckAvailability();
        }

        if (ARSession.state == ARSessionState.Unsupported)
        {
            // AR no soportado en este dispositivo
            Debug.LogError("AR no disponible en este dispositivo");
```

```
            ShowErrorMessage("Este dispositivo no soporta ARCore");
        }
        else if (ARSession.state == ARSessionState.NeedsInstall)
        {
            // ARCore necesita instalarse
            Debug.Log("ARCore requiere instalación");
            yield return ARSession.Install();

            if (ARSession.state == ARSessionState.Installing)
            {
                Debug.Log("Instalando ARCore...");
            }
        }
        else
        {
            // AR disponible, iniciar sesión
            Debug.Log("AR disponible y listo");
            InitializeAR();
        }
    }

    private void ShowErrorMessage(string message)
    {
        // Mostrar UI de error al usuario
    }

    private void InitializeAR()
    {
        // Iniciar experiencia AR
    }
}
```

🧊 MaterialChecker (Script)

🎯 Objetivo

Detectar y reemplazar materiales conflictivos que no son compatibles con **URP (Universal Render Pipeline)**, evitando errores como
State comes from an incompatible keyword space en proyectos AR con XR Simulator.

🔧 Propiedades públicas

- `Material replacementMaterialURP` → Material de reemplazo compatible con URP.
- `string problematicMaterialName` → Nombre del material conflictivo a detectar (por defecto `"SimulationLitGray"`).
- `MessageController messageController` → Referencia al controlador de mensajes para mostrar feedback en la UI.

📂 Dependencias

- `UnityEngine`
- `MessageController` (script propio para mostrar mensajes en UI)

📜 Métodos clave

- `Start()`
  - Busca automáticamente un `MessageController` en la escena si no está asignado.
  - Llama a `ReplaceProblematicMaterials()` .
- `ReplaceProblematicMaterials()`
  - Itera todos los `Renderer` de la escena.
  - Comprueba cada material asociado:
    - Si coincide con `problematicMaterialName` o usa el shader `"Simulation/Standard Lit"` , lo reemplaza por `replacementMaterialURP` .
    - Registra el reemplazo en la UI mediante `messageController` .
  - Al finalizar:
    - Muestra mensaje de éxito si hubo reemplazos.
    - Muestra mensaje informativo si no se detectaron materiales conflictivos.

🧠 Flujo de interacción

1. Al iniciar la escena, el script busca materiales conflictivos.
2. Si encuentra alguno, lo reemplaza automáticamente por el material URP definido.
3. Informa al usuario mediante mensajes en consola y UI.
4. Garantiza que todos los materiales sean compatibles con URP, evitando errores de renderizado.

💡 Notas técnicas

- Este script es útil en proyectos migrados desde el **Built-in Render Pipeline** hacia URP.
- Se recomienda asignar un material URP genérico (`URP/Lit`) como `replacementMaterialURP` .
- El uso de `MessageController` permite dar feedback inmediato al usuario en la interfaz.
- Ideal mantenerlo dentro del objeto `ARSystemCheck` junto con otros scripts de validación.

📜 Código fuente

```
using UnityEngine;

public class MaterialChecker : MonoBehaviour
{
```

```
    public Material replacementMaterialURP;
    public string problematicMaterialName = "SimulationLitGray";
    public MessageController messageController;

    void Start()
    {
        if (messageController == null)
            messageController = Object.FindAnyObjectByType<MessageController>();

        ReplaceProblematicMaterials();
    }

    void ReplaceProblematicMaterials()
    {
        Renderer[] renderers = Object.FindObjectsByType<Renderer>(FindObjectsSortMode.None);
        int replacements = 0;

        foreach (Renderer renderer in renderers)
        {
            for (int i = 0; i < renderer.sharedMaterials.Length; i++)
            {
                Material mat = renderer.sharedMaterials[i];
                if (mat != null &&
                    (mat.name == problematicMaterialName || mat.shader.name == "Simulation/S
                {
                    renderer.materials[i] = replacementMaterialURP;
                    replacements++;
                    messageController?.ShowInfoMessage("[!] Reemplazo aplicado en: " + rende
                }
            }
        }

        if (replacements > 0)
        {
            messageController?.ShowInfoMessage("[OK] Materiales conflictivos reemplazados.")
        }
        else
        {
            messageController?.ShowInfoMessage("[OK] No se detectaron materiales conflictivo
        }
    }
}
```

🧊 InfoDisplay (Script)

🎯 Objetivo

Mostrar mensajes informativos, de advertencia o de error en la interfaz de usuario utilizando **TextMeshPro**.

Este script centraliza la visualización de mensajes con distintos colores según su tipo.

🔧 Propiedades públicas

- `TextMeshProUGUI infoText` → Referencia al componente de texto en la UI.
- `Color infoColor` → Color para mensajes informativos (por defecto blanco).
- `Color warningColor` → Color para advertencias (por defecto naranja).
- `Color errorColor` → Color para errores (por defecto rojo).

📂 Dependencias

- `UnityEngine`
- `TMPro` (TextMeshPro)

📜 Métodos clave

- `ShowInfo(string message)` → Muestra un mensaje informativo en color blanco.
- `ShowWarning(string message)` → Muestra un mensaje de advertencia en color naranja.
- `ShowError(string message)` → Muestra un mensaje de error en color rojo.
- `ClearMessage()` → Limpia el texto mostrado en la UI.
- `SetMessage(string message, Color color)` → Método interno que aplica el texto y el color al componente `infoText`.

🧠 Flujo de interacción

1. Otro script invoca `ShowInfo()`, `ShowWarning()` o `ShowError()` según el contexto.
2. El mensaje se muestra en la UI con el color correspondiente.
3. El usuario recibe feedback inmediato sobre el estado del sistema.
4. `ClearMessage()` permite limpiar la pantalla cuando ya no se necesita mostrar información.

💡 Notas técnicas

- Este script debe estar asociado a un objeto **Canvas** con un componente `TextMeshProUGUI`.
- Es recomendable usarlo junto con `MessageController` para gestionar mensajes globales.
- Permite diferenciar visualmente el tipo de mensaje, mejorando la experiencia de depuración y usuario final.

📜 Código fuente

```
using TMPro;
using UnityEngine;

public class InfoDisplay : MonoBehaviour
```

```
{
    public TextMeshProUGUI infoText;

    [Header("Colores de mensaje")]
    public Color infoColor = Color.white;
    public Color warningColor = new Color(1f, 0.65f, 0f); // naranja
    public Color errorColor = Color.red;

    public void ShowInfo(string message)
    {
        SetMessage(message, infoColor);
    }

    public void ShowWarning(string message)
    {
        SetMessage(message, warningColor);
    }

    public void ShowError(string message)
    {
        SetMessage(message, errorColor);
    }

    public void ClearMessage()
    {
        infoText.text = "";
    }

    private void SetMessage(string message, Color color)
    {
        if (infoText != null)
        {
            infoText.text = message;
            infoText.color = color;
        }
    }
}
```

🧊 MessageController (Script)

🎯 Objetivo

Centralizar la gestión de mensajes en la interfaz de usuario, utilizando el componente `InfoDisplay` para mostrar información, advertencias y errores durante la interacción en AR.
Este script actúa como puente entre la lógica del sistema y la UI, asegurando que el usuario reciba feedback inmediato.

🔧 Propiedades públicas

- `[SerializeField] private InfoDisplay infoDisplay` → Referencia al componente `InfoDisplay` que renderiza los mensajes en pantalla.
  - Si no se asigna manualmente, se busca automáticamente en la escena durante `Awake()`.

📂 Dependencias

- `UnityEngine`
- `InfoDisplay` (script propio que gestiona la visualización de mensajes con TextMeshPro)

📜 Métodos clave

- `ShowStartMessage()` → Muestra instrucción inicial: "*Toca una superficie para colocar el objeto.*"
- `ShowInfoMessage(string message)` → Muestra un mensaje informativo personalizado.
- `ShowPlacementSuccess()` → Confirma que un objeto fue colocado correctamente.
- `ShowPlacementWarning()` → Advierte que no se detectó una superficie válida.
- `ShowPlacementBlocked()` → Informa que ya existe un objeto en la posición seleccionada.
- `ShowPlacementPosition(Vector2 pos)` → Muestra la posición en la que se colocó el objeto.
- `ShowSelection(GameObject obj)` → Informa qué objeto fue seleccionado.
- `ShowNoSelection()` → Advierte que no hay objeto seleccionado.
- `ShowDeletion()` → Confirma que un objeto fue eliminado.
- `ShowError(string message)` → Muestra un mensaje de error en color rojo.
- `Clear()` → Limpia el mensaje actual en pantalla.

🧠 Flujo de interacción

1. Los distintos sistemas de AR (colocación, selección, eliminación) llaman a los métodos de `MessageController`.
2. Este script traduce los eventos en mensajes claros para el usuario.
3. `InfoDisplay` se encarga de renderizar el texto con el color correspondiente (informativo, advertencia o error).
4. El usuario recibe feedback inmediato sobre cada acción realizada.

💡 Notas técnicas

- Este script debe estar en un objeto central de la escena (`ARReport` recomendado).
- Permite desacoplar la lógica de interacción de la gestión de mensajes, manteniendo el código modular.
- Se recomienda enlazarlo con `MaterialChecker`, `ARAvailabilityChecker` y otros scripts para mostrar mensajes globales.
- Compatible con **TextMeshPro**, lo que asegura mejor legibilidad en UI móvil.

📜 Código fuente

```
using UnityEngine;

public class MessageController : MonoBehaviour
{
    [SerializeField] private InfoDisplay infoDisplay;

    private void Awake()
    {
        if (infoDisplay == null)
            infoDisplay = Object.FindAnyObjectByType<InfoDisplay>();
    }

    public void ShowStartMessage()
    {
        infoDisplay?.ShowInfo("Toca una superficie para colocar el objeto.");
    }

    public void ShowInfoMessage(string message)
    {
        infoDisplay?.ShowInfo(message);
    }

    public void ShowPlacementSuccess()
    {
        infoDisplay?.ShowInfo("Objeto colocado correctamente.");
    }

    public void ShowPlacementWarning()
    {
        infoDisplay?.ShowWarning("No se detectó una superficie válida.");
    }

    public void ShowPlacementBlocked()
    {
        infoDisplay?.ShowWarning("Ya hay un objeto en ese lugar.");
    }

    public void ShowPlacementPosition(Vector2 pos)
    {
        infoDisplay?.ShowInfo($"Objeto colocado en: {pos}");
    }

    public void ShowSelection(GameObject obj)
    {
        string name = obj.name;
        infoDisplay?.ShowInfo($"Objeto seleccionado: {name}");
    }

    public void ShowNoSelection()
    {
```

```
        infoDisplay?.ShowWarning("No hay objeto seleccionado.");
    }

    public void ShowDeletion()
    {
        infoDisplay?.ShowInfo("Objeto eliminado.");
    }

    public void ShowError(string message)
    {
        infoDisplay?.ShowError($"Error: {message}");
    }

    public void Clear()
    {
        infoDisplay?.ClearMessage();
    }
}
```

🧊 ARObjectPlacerAndSelector (Script)

🎯 Objetivo

Gestionar la **colocación, selección y eliminación de objetos AR** en planos detectados con AR Foundation, integrando gestos táctiles, mensajes en UI y efectos visuales.
Este script centraliza la interacción del usuario con prefabs en la escena AR.

🔧 Propiedades públicas

- AR Managers
  - `GameObject objectToPlace` → Prefab a instanciar en planos detectados.
  - `ARRaycastManager raycastManager` → Manager para raycasts en planos AR.
  - `ARPlaneManager planeManager` → Manager de planos AR.
  - `ARAnchorManager anchorManager` → Manager de anclajes AR.
  - `ARGestureDetector gestureDetector` → Detector de gestos para manipulación de objetos.
- UI & Messaging
  - `MessageController messageController` → Controlador de mensajes en UI.
  - `RawImage crosshairImage` → Imagen de la mirilla en pantalla.
  - `TextMeshProUGUI crosshairButtonText` → Texto del botón de mirilla.
- Visual Settings

- `GameObject shotVFX` → Prefab de efectos visuales al disparar/eliminar.

- **Placement Settings**
  - `float placementThreshold` → Distancia mínima para evitar colocar objetos demasiado cerca.

📂 Dependencias

- `UnityEngine`
- `UnityEngine.InputSystem`
- `UnityEngine.XR.ARFoundation`
- `UnityEngine.XR.ARSubsystems`
- `TMPro`
- `UnityEngine.UI`
- Scripts propios: `MessageController`, `ARGestureDetector`, `ARObjectManipulator`

📜 Métodos clave

- **Eventos de entrada**
  - `OnEnable()` → Activa acciones de Input System (`Tap`, `DoubleTap`).
  - `OnDisable()` → Desactiva acciones de Input System.
  - `OnTouchPerformed()` → Gestiona toque en pantalla: selección o colocación de objeto.
  - `OnDoubleTap()` → Elimina objeto seleccionado con efectos visuales.

- **Colocación y selección**
  - `TrySelectObject(Vector2 screenPosition)` → Intenta seleccionar un objeto AR.
  - `AnchorObject(GameObject obj, Pose pose)` → Crea un `ARAnchor` y lo asigna al objeto instanciado.
  - `DeleteAllPrefabs()` → Elimina todos los objetos interactuables de la escena.

- **UI y mirilla**
  - `ToggleCrosshair()` → Activa/desactiva la mirilla en pantalla.
  - `FireAndDestroyCenterObject()` → Dispara al objeto en el centro de la pantalla si la mirilla está activa.

- **Eliminación con efectos**

- `DestroyWithVFX(GameObject targetObject)` → Corrutina que reproduce efectos visuales antes de destruir el objeto.

🧠 Flujo de interacción

1. El usuario toca la pantalla:
   - Si toca un objeto → se selecciona.
   - Si toca un plano → se instancia el prefab en esa posición.
2. El sistema valida proximidad con otros objetos (`placementThreshold`).
3. Los objetos instanciados se anclan con `ARAnchor` para mayor estabilidad.
4. El usuario puede:
   - Eliminar objetos con **doble toque**.
   - Eliminar todos los objetos con `DeleteAllPrefabs()` .
   - Activar la **mirilla** y disparar al objeto en el centro de la pantalla.
5. Los mensajes se muestran en UI mediante `MessageController` .

💡 Notas técnicas

- El uso de `ARAnchor` asegura que los objetos permanezcan estables en el espacio AR.
- `placementThreshold` evita superposición de objetos.
- Compatible con **Input System** y gestos multitáctiles.
- Se recomienda asignar `shotVFX` para mejorar la experiencia visual al eliminar objetos.
- Integrar con `ARGestureDetector` para manipulación avanzada (rotación, escala, traslación).

📜 Código fuente

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

public class ARObjectPlacerAndSelector : MonoBehaviour
{
    [Header("AR Managers")]
    [SerializeField] private GameObject objectToPlace;
    [SerializeField] private ARRaycastManager raycastManager;
```

```
[SerializeField] private ARPlaneManager planeManager;
[SerializeField] private ARAnchorManager anchorManager;
[SerializeField] private ARGestureDetector gestureDetector;

[Header("UI & Messaging")]
[SerializeField] private MessageController messageController;
[SerializeField] private RawImage crosshairImage;
[SerializeField] private TextMeshProUGUI crosshairButtonText;

[Header("Visual Settings")]
[SerializeField] private GameObject shotVFX;

[Header("Placement Settings")]
[SerializeField] private float placementThreshold = 0.1f;

private List<GameObject> placedObjects = new List<GameObject>();
private List<ARRaycastHit> hits = new List<ARRaycastHit>();
private ARInputActions inputActions;
private ARObjectManipulator currentSelectedObject;
private bool crosshairVisible = false;


void Awake()
{
    inputActions = new ARInputActions();
}

void OnEnable()
{
    inputActions.ARPlayer.Enable();
    inputActions.ARPlayer.Tap.performed += OnTouchPerformed;
    inputActions.ARPlayer.DoubleTap.performed += ctx => OnDoubleTap();
}

void OnDisable()
{
    inputActions.ARPlayer.Tap.performed -= OnTouchPerformed;
    inputActions.ARPlayer.DoubleTap.performed -= ctx => OnDoubleTap();
    inputActions.ARPlayer.Disable();
}
private void OnDoubleTap()
{
    Debug.Log("Double Tap detectado");
    messageController?.ShowInfoMessage("[INFO] Double Tap.");

    Vector2 touchPosition = inputActions.ARPlayer.TouchPosition.ReadValue<Vector2>();
    Ray ray = Camera.main.ScreenPointToRay(touchPosition);

    if (Physics.Raycast(ray, out RaycastHit hit))
    {
        GameObject targetObject = hit.collider.gameObject;

        // Validar que el objeto tiene el tag correcto y está seleccionado
```

```
        if (hit.collider.CompareTag("Interactable") && targetObject == currentSelectedOb
        {
            if (shotVFX != null)
                Instantiate(shotVFX, hit.point, Quaternion.identity);

            StartCoroutine(DestroyWithVFX(targetObject));
            messageController?.ShowDeletion();
        }
        else
        {
            messageController?.ShowInfoMessage("[INFO] Doble toque sobre objeto no selec
        }
    }
}
private void OnTouchPerformed(InputAction.CallbackContext context)
{
    // Leer posición del toque
    Vector2 touchPosition = inputActions.ARPlayer.TouchPosition.ReadValue<Vector2>();

    // Intentar seleccionar un objeto primero
    if (TrySelectObject(touchPosition))
        return;

    // Validaciones defensivas
    if (raycastManager == null || hits == null)
    {
        Debug.LogWarning("RaycastManager o lista de hits no está disponible.");
        return;
    }

    if (objectToPlace == null)
    {
        Debug.LogWarning("Prefab no asignado: objectToPlace es null.");
        return;
    }

    // Intentar colocar objeto si no se seleccionó ninguno
    if (raycastManager.Raycast(touchPosition, hits, TrackableType.PlaneWithinPolygon) &&
    {
        Pose hitPose = hits[0].pose;

        // Limpiar referencias nulas en la lista
        placedObjects.RemoveAll(obj => obj == null);

        // Verificar proximidad con objetos existentes
        foreach (GameObject obj in placedObjects)
        {
            if (Vector3.Distance(obj.transform.position, hitPose.position) < placementTh
            {
                messageController?.ShowPlacementBlocked();
                Debug.Log("Ya existe un objeto cerca. No se colocará otro.");
                return;
            }
```

```
        }

        // Instanciar y anclar el nuevo objeto
        GameObject placedObject = Instantiate(objectToPlace, hitPose.position, hitPose.r
        placedObjects.Add(placedObject);
        AnchorObject(placedObject, hitPose);

        messageController?.ShowPlacementPosition(hitPose.position);
        Debug.Log($"Objeto colocado en {hitPose.position}");
    }
}

private bool TrySelectObject(Vector2 screenPosition)
{
    Ray ray = Camera.main.ScreenPointToRay(screenPosition);
    if (Physics.Raycast(ray, out RaycastHit hit))
    {
        ARObjectManipulator manipulator = hit.collider.GetComponent<ARObjectManipulator>
        if (manipulator != null)
        {
            if (currentSelectedObject != null)
            {
                currentSelectedObject.Deselect();
                gestureDetector.selectedObject = null;
            }

            currentSelectedObject = manipulator;
            currentSelectedObject.Select();
            gestureDetector.selectedObject = currentSelectedObject;

            return true;
        }
    }

    if (currentSelectedObject != null)
    {
        currentSelectedObject.Deselect();
        currentSelectedObject = null;
        gestureDetector.selectedObject = null;
    }

    return false;
}

private void AnchorObject(GameObject obj, Pose pose)
{
    // Crear un nuevo Anchor en la posición detectada
    GameObject anchorGO = new GameObject("ARAnchor");
    anchorGO.transform.position = pose.position;

    // ⚠️ Importante: no tocar la rotación del anchor
    anchorGO.transform.rotation = Quaternion.identity;
```

```
        // Añadir el componente ARAnchor
        anchorGO.AddComponent<ARAnchor>();

        // Reparentar el objeto conservando su transform local
        obj.transform.SetParent(anchorGO.transform, true);
    }

    public void DeleteAllPrefabs()
    {
        foreach (GameObject obj in GameObject.FindGameObjectsWithTag("Interactable"))
        {
            StartCoroutine(DestroyWithVFX(obj));
        }

        currentSelectedObject = null;
        gestureDetector.selectedObject = null;

        messageController?.ShowInfoMessage("[DEL] Todos los objetos eliminados.");
    }
    // Método para activar/desactivar la mirilla
    public void ToggleCrosshair()
    {
        crosshairVisible = !crosshairVisible;
        crosshairImage.gameObject.SetActive(crosshairVisible);

        if (crosshairVisible)
        {
            messageController?.ShowInfoMessage("[VER] Mirilla activada.");
            crosshairButtonText.text = "Desactivar Mirilla";
        }
        else
        {
            messageController?.ShowInfoMessage("[VER] Mirilla desactivada.");
            crosshairButtonText.text = "Activar Mirilla";
        }
    }

    // Método para disparar y destruir solo si la mirilla está activa
    public void FireAndDestroyCenterObject()
    {
        if (!crosshairVisible)
        {
            messageController?.ShowInfoMessage("[INF] La mirilla no está activa. Actívala pr
            return;
        }

        Vector2 screenCenter = new(Screen.width / 2f, Screen.height / 2f);
        Ray ray = Camera.main.ScreenPointToRay(screenCenter);

        if (Physics.Raycast(ray, out RaycastHit hit) && hit.collider.CompareTag("Interactabl
        {
            GameObject targetObject = hit.collider.gameObject;
```

```
        // Instanciar efectos de disparo
        if (shotVFX != null)
            Instantiate(shotVFX, hit.point, Quaternion.identity);

        StartCoroutine(DestroyWithVFX(targetObject));
        messageController?.ShowDeletion();
    }
    else
    {
        messageController?.ShowInfoMessage("[INF] No hay objeto en el centro.");
    }
}
private IEnumerator DestroyWithVFX(GameObject targetObject)
{
    if (targetObject == null) yield break;

    // Obtener el componente SelectableObject
    ARObjectManipulator selectable = targetObject.GetComponent<ARObjectManipulator>();
    ParticleSystem vfx = null;

    if (selectable != null)
    {
        Transform vfxTransform = selectable.transform.Find("DeleteVFX");
        if (vfxTransform != null)
        {
            GameObject vfxGO = vfxTransform.gameObject;
            vfxGO.SetActive(true);
            vfx = vfxGO.GetComponent<ParticleSystem>();

            if (vfx != null)
            {
                vfxGO.transform.SetParent(null);
                vfxGO.transform.position = targetObject.transform.position;
                vfxGO.transform.rotation = targetObject.transform.rotation;

                yield return null; // Esperar un frame

                vfx.Play();

                float delay = vfx.main.duration + vfx.main.startLifetime.constant;
                //Destroy(vfxGO, delay);
            }
        }
    }

    yield return new WaitForSeconds(0.1f); // Pequeña pausa para asegurar visibilidad

    placedObjects.Remove(targetObject);

    Transform anchorTransform = targetObject.transform.parent;
    if (anchorTransform != null && anchorTransform.GetComponent<ARAnchor>() != null)
    {
        Destroy(anchorTransform.gameObject);
```

```
        }
        else
        {
            Destroy(targetObject);
        }

        if (vfx != null)
        {
            float delay = vfx.main.duration + vfx.main.startLifetime.constantMax;
            yield return new WaitForSeconds(delay);
        }
    }

}
```

🧊 ARGestureDetector (Script)

🎯 Objetivo

Detectar y gestionar **gestos multitáctiles** en dispositivos móviles para manipular objetos AR seleccionados.
Este script permite mover, escalar y rotar objetos en tiempo real mediante gestos táctiles, integrando feedback en la UI.

🔧 Propiedades públicas

- AR Managers
  - `ARObjectManipulator selectedObject` → Objeto actualmente seleccionado para manipulación.
  - `ARRaycastManager raycastManager` → Manager para raycasts en planos AR.
- UI & Messaging
  - `MessageController messageController` → Controlador de mensajes para mostrar feedback en la interfaz.

📂 Dependencias

- `UnityEngine`
- `UnityEngine.InputSystem`
- `UnityEngine.InputSystem.EnhancedTouch`

- `UnityEngine.XR.ARFoundation`
- `UnityEngine.XR.ARSubsystems`
- Scripts propios: `MessageController` , `ARObjectManipulator`

📜 Métodos clave

- **Ciclo de vida**
  - `OnEnable()` → Activa soporte multitáctil (`EnhancedTouchSupport.Enable()`).
  - `OnDisable()` → Desactiva soporte multitáctil y reinicia valores.
  - `Update()` → Evalúa número de toques activos y ejecuta el gesto correspondiente.
- **Gestos táctiles**
  - `HandleSingleTouch(Touch touch)` → Detecta arrastre y mueve el objeto seleccionado.
  - `HandlePinchToScale(Touch touch0, Touch touch1)` → Escala el objeto según la distancia entre dos dedos.
  - `HandleTwistToRotate(Touch touch0, Touch touch1)` → Rota el objeto según el ángulo entre los movimientos de dos dedos.
  - `MoveSelectedObject(Vector2 screenPosition)` → Actualiza la posición del objeto en el plano detectado, conservando su rotación.

🧠 Flujo de interacción

1. El usuario selecciona un objeto AR en la escena.
2. El script detecta gestos multitáctiles:
   - **Un dedo** → arrastre para mover el objeto.
   - **Dos dedos** → gesto de pinza para escalar y giro para rotar.
3. El objeto se actualiza en tiempo real en la escena AR.
4. Los mensajes de estado se muestran en la UI mediante `MessageController` .

💡 Notas técnicas

- `EnhancedTouchSupport` debe estar habilitado para usar gestos multitáctiles.
- El movimiento se limita a planos detectados por `ARRaycastManager` .
- El escalado está limitado entre `0.5x` y `2x` para evitar deformaciones extremas.
- La rotación se suaviza con `Mathf.Lerp` para mayor estabilidad visual.
- Ideal para complementar el flujo de selección y colocación de objetos gestionado por `ARObjectPlacerAndSelector` .

📜 Código fuente

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.EnhancedTouch;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using Touch = UnityEngine.InputSystem.EnhancedTouch.Touch;
using InputTouchPhase = UnityEngine.InputSystem.TouchPhase;

public class ARGestureDetector : MonoBehaviour
{
    [Header("AR Managers")]
    public ARObjectManipulator selectedObject;
    [SerializeField] private ARRaycastManager raycastManager;

    [Header("UI & Messaging")]
    [SerializeField] private MessageController messageController;

    private float initialDistance;
    private Vector3 initialScale;
    private const float movementThreshold = 5f;
    private readonly List<ARRaycastHit> rayHits = new();

    private void OnEnable() => EnhancedTouchSupport.Enable();

    private void OnDisable()
    {
        EnhancedTouchSupport.Disable();
        initialDistance = 0f;
    }

    private void Update()
    {
        if (selectedObject == null || !selectedObject.IsSelected || !EnhancedTouchSupport.en
            return;

        var touches = Touch.activeTouches;

        switch (touches.Count)
        {
            case 1:
                HandleSingleTouch(touches[0]);
                break;
            case 2:
                HandlePinchToScale(touches[0], touches[1]);
                HandleTwistToRotate(touches[0], touches[1]);
                break;
```

```
        default:
            initialDistance = 0f;
            break;
    }
}

private void HandleSingleTouch(Touch touch)
{
    bool isDragging = touch.phase == InputTouchPhase.Moved && touch.delta.magnitude > mo

    if (isDragging)
        MoveSelectedObject(touch.screenPosition);
}

private void HandlePinchToScale(Touch touch0, Touch touch1)
{
    float currentDistance = Vector2.Distance(touch0.screenPosition, touch1.screenPositio

    if (initialDistance == 0f)
    {
        initialDistance = currentDistance;
        initialScale = selectedObject.transform.localScale;
        messageController?.ShowInfoMessage("[aA] Escalando...");
        return;
    }

    float scaleFactor = Mathf.Clamp(currentDistance / initialDistance, 0.5f, 2f);
    selectedObject.transform.localScale = initialScale * scaleFactor;
}

private void HandleTwistToRotate(Touch touch0, Touch touch1)
{
    float rawAngle = Vector2.SignedAngle(touch0.delta, touch1.delta);
    float smoothAngle = Mathf.Lerp(0f, rawAngle, 0.1f);

    selectedObject.transform.Rotate(Vector3.up, smoothAngle);
    messageController?.ShowInfoMessage($"[ROT] {selectedObject.name} rotado ({smoothAngl
}

private void MoveSelectedObject(Vector2 screenPosition)
{
    if (raycastManager == null) return;

    rayHits.Clear();
    if (raycastManager.Raycast(screenPosition, rayHits, TrackableType.PlaneWithinPolygon
    {
        Pose hitPose = rayHits[0].pose;

        // Solo actualizar posición, conservar rotación actual
        selectedObject.transform.SetPositionAndRotation(hitPose.position, selectedObject

        messageController?.ShowInfoMessage($"[MOV] {selectedObject.name} movido.");
    }
```

```
    }
}
```

🧮 ¿Touch?, ¿Qué Touch?

🎯 Colofón

Este proyecto muestra cómo el **nuevo Input System de Unity** redefine la interacción táctil en entornos de **Realidad Aumentada móvil (Android + ARCore)**.
El título *"¿Touch?, ¿Qué Touch?"* alude a la transición desde el sistema heredado hacia un modelo más robusto, escalable y multitáctil, donde cada gesto se convierte en un evento trazable y reproducible.

🧩 Síntesis de los Scripts

- **InputSystemCheck**
  Verifica que el **nuevo Input System** está activo y muestra la versión instalada, asegurando compatibilidad desde el inicio.

- **ARAvailabilityChecker**
  Comprueba si el dispositivo soporta **ARCore**, gestiona la instalación cuando es necesaria y evita fallos en dispositivos no compatibles.

- **MaterialChecker**
  Detecta y reemplaza materiales conflictivos, garantizando compatibilidad con **URP** y evitando errores de renderizado en simulación.

- **InfoDisplay**
  Centraliza la visualización de mensajes en la UI con **TextMeshPro**, diferenciando entre información, advertencias y errores.

- **MessageController**
  Actúa como puente entre la lógica del sistema y la UI, mostrando mensajes claros sobre colocación, selección y eliminación de objetos.

- **ARObjectPlacerAndSelector**
  Gestiona la colocación de prefabs en planos detectados, su selección, eliminación y la interacción con la mirilla, integrando raycasts y anclajes AR.

- **ARGestureDetector**
  Implementa gestos multitáctiles (arrastre, pinch-to-scale, twist-to-rotate) para manipular objetos AR en tiempo real, con soporte de **EnhancedTouch**.

🧠 Conclusión

El conjunto de scripts convierte la interacción táctil en un **sistema modular y validado paso a paso**, donde cada acción del usuario —desde un toque simple hasta un gesto complejo— se traduce en una respuesta clara, visual y documentada.

Este enfoque asegura:

- **Compatibilidad** con Android y ARCore.
- **Robustez** frente a errores de materiales y disponibilidad AR.
- **Escalabilidad** gracias al nuevo Input System y su soporte multitáctil.
- **Claridad** mediante mensajes en UI que guían al usuario en todo momento.

💡 Reflexión final

El *touch* ya no es solo un evento aislado, sino un **lenguaje completo de interacción** en AR móvil. Con el nuevo Input System, cada gesto se convierte en una pieza fundamental para construir experiencias inmersivas, reproducibles y mantenibles en Unity.