



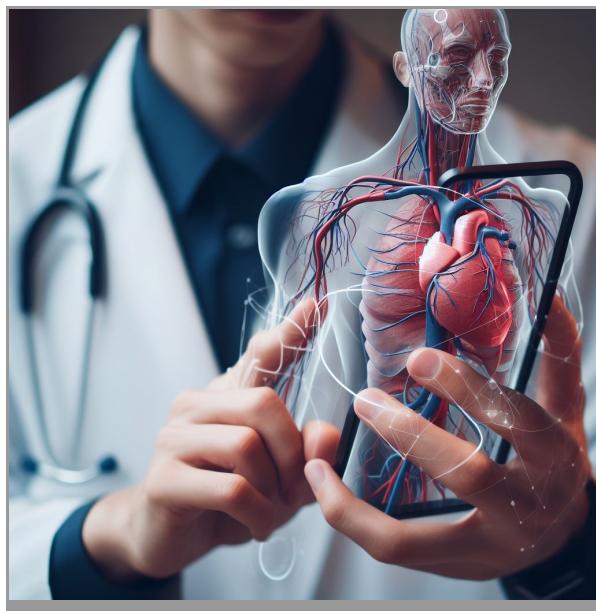
**UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN
CICLO 2023-I**

Curso: CC221: Arquitectura de computadores

Docente: Cesar Martin Cruz Salazar

Proyecto Formativo - Parte 2

“Realidad aumentada: Aplicación a la enseñanza de la anatomía”



Integrantes:

JHARVY JONAS CADILLO TARAZONA (20210184E)

MARTIN ALONSO CENTENO LEÓN (20210161E)

CINVER ALEM ESPINOZA VALERA (20220277F)

J. ALAN C. ESPINOZA HUAMAN (20160739I)

Diciembre, 2023

INICIO

1. Introducción.....	3
2. Objetivos.....	3
3. Resumen Teórico.....	3
4. Arquitectura de la aplicación.....	4
5. Paquetes y Dependencias.....	7
5.1. AR Foundation.....	7
• AR session.....	7
• AR session Origin.....	7
• AR camera manager.....	7
• AR Raycast Manager.....	7
5.2. AR Core.....	7
5.3. DoTween.....	8
5.4. UniGLTF.....	8
6. Scripts en Unity.....	9
6.1. Game Manager.....	9
6.2. Data Manager.....	11
6.3. UI Manager.....	11
6.4. Plane Manager.....	13
6.5. Models 3D.....	14
6.6. Models Button.....	15
6.7. Share Screenshot.....	15
6.8. Button Toggle.....	17
6.9. UI Animation Button.....	18
6.10. AR Interaction Manager.....	20
7. Casos de Uso y Ejemplos prácticos.....	23
8. Observaciones.....	23
9. Conclusiones.....	24
10. Referencias.....	24
11. Repositorio.....	24

1. Introducción

En nuestra aplicación de realidad aumentada, combinamos la innovadora tecnología de AR con la educación. El código subyacente utiliza algoritmos avanzados para superponer información médica en el mundo real, facilitando la comprensión y aprendizaje. Con modelos tridimensionales y funciones interactivas, creamos una experiencia inmersiva. Nuestro código modular permite futuras expansiones, asegurando que nuestra app puede mejorarse progresivamente. En resumen, nuestra app no solo presenta una tecnología avanzada, sino también puede ser efectiva en la educación, mejorando la enseñanza en aspectos médicos como la anatomía.

2. Objetivos

- Aprender y hacer uso de las herramientas de Unity para desarrollar una aplicación con Realidad Aumentada
- Desarrollar la aplicación de realidad aumentada para que sea implementada en la visualización de los modelos y brindar una herramienta pedagógica en el aprendizaje de la anatomía humana.

3. Resumen Teórico

La Realidad Aumentada (RA) ha emergido como una tecnología innovadora que transforma la manera en que interactuamos con el mundo que nos rodea. Al integrar elementos digitales en el entorno real, la RA mejora la percepción del usuario al proporcionar información en diversas formas, como texto, imágenes, sonidos, videos y modelos tridimensionales. Esta tecnología se apoya en dispositivos como smartphones, tablets o gafas de Realidad Aumentada (AR) para superponer información digital en tiempo real sobre el mundo físico, creando así una experiencia más rica y contextual.

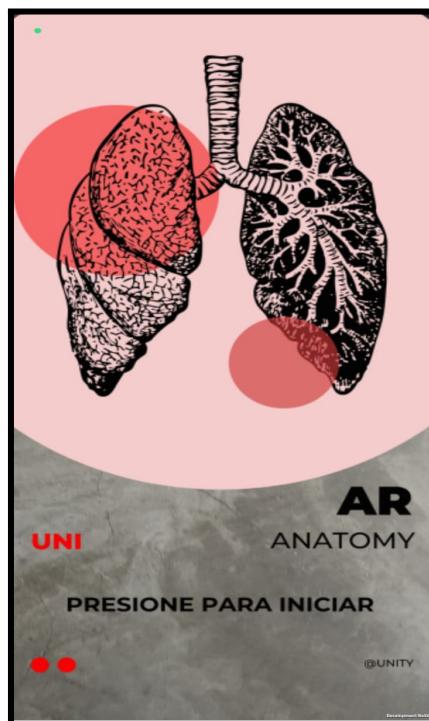
Una de las definiciones más destacadas de la RA es la propuesta por Milgram y Kishino, quienes introdujeron el Continuo de Realidad-Virtualidad. Este continuo describe una escala que va desde la realidad pura hasta la realidad virtual completa, destacando cómo la RA se sitúa en algún punto intermedio al combinar elementos del mundo real con elementos virtuales. Por otro lado, Azuma proporciona una definición más específica al establecer que la RA es interactiva en tiempo real, combina elementos reales y virtuales, y se registra en 3D.

Uno de los campos donde la RA ha demostrado un impacto significativo es la medicina. Esta misma ha revolucionado la forma en que los profesionales de la salud diagnostican, tratan y educan a los pacientes.

4. Arquitectura de la aplicación

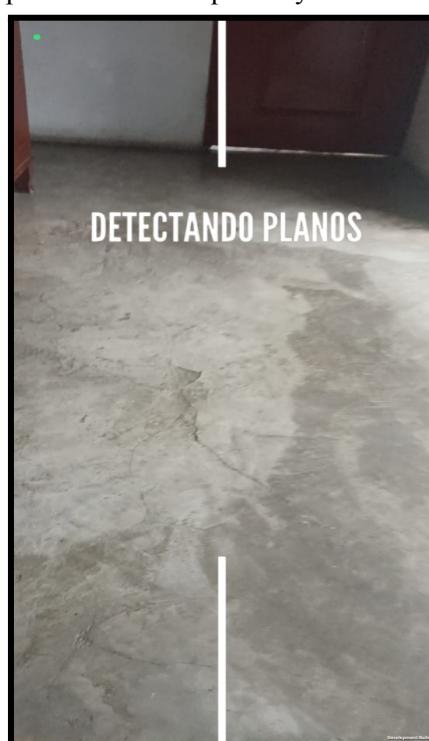
En el desarrollo de nuestro proyecto de realidad aumentada, hemos optado por una arquitectura basada en un patrón de eventos, donde los scripts se suscriben a eventos específicos para gestionar la comunicación entre componentes de manera eficiente y desacoplada. ¿Cómo funciona? Por ejemplo en nuestra aplicación tenemos los siguientes estados:

Inicio: Estado donde comienza la aplicación y nos avisa que presionemos para comenzar.



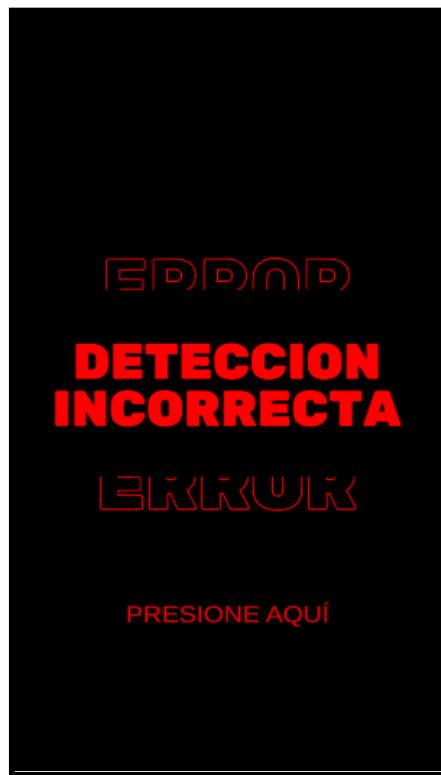
Figura[1]: Pantalla de inicio de la aplicación.

Detección: Estado en el cual la aplicación detecta planos y los almacena para ser usado en otro estado.



Figura[2]: Pantalla de detección de planos de la aplicación.

Detección Incorrecta: Estado que advierte al usuario que no se detectó planos o que hubo un error en la detección.



Figura[3]: Pantalla de error de detección de la aplicación

Menú Principal: Estado en el cual el usuario puede acceder a los modelos que implementamos y opciones de la aplicación como salir, detectar más planos, regresar al inicio o salir de la aplicación.



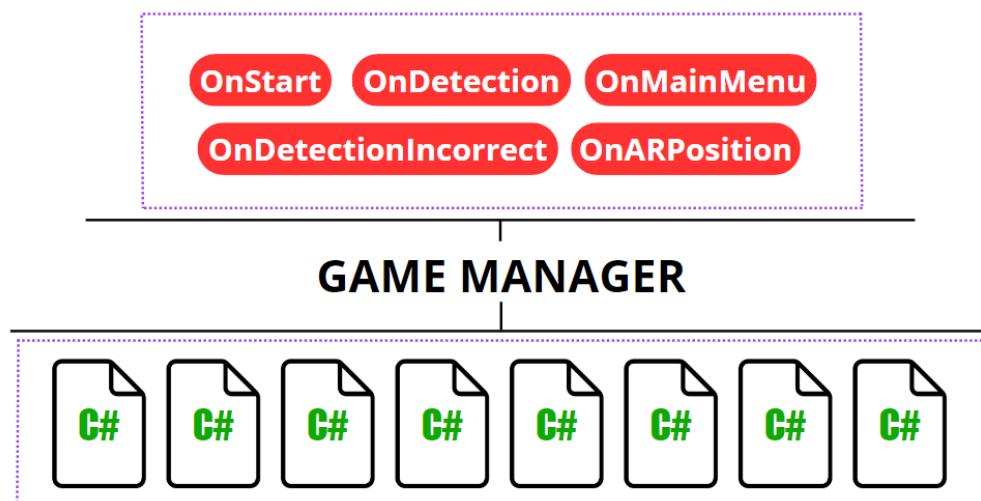
Figura[4]: Pantalla del menú de la aplicación

Posición Modelos: Estado en el cual se posiciona el modelo en el plano que se detectó en el estado Detección, nos da la opción de aceptar la posición o eliminar.



Figura[5]: Pantalla de la aceptación de la posición del modelo

Entonces representamos estos estados como eventos: OnStart, OnDetection, OnMainMenu, OnDetectionIncorrect, OnARPosition y todos estos eventos forman parte del script GameManager y gracias a este script, muchos scripts forman parte y están suscritos a estos eventos como por ejemplo la interfaz que cambia dependiendo de qué evento está siendo llamado.



Figura[6]: Estructura de la arquitectura de la aplicación AR-Anatomy

5. Paquetes y Dependencias

Al realizar la aplicación AR-Anatomy se utilizó varios paquetes para que la aplicación funcione correctamente, algunos para los modelos, algunos para el código y también para la animación del interfaz de usuario. Los paquetes utilizados han sido los siguientes:

5.1. AR Foundation

Es un paquete de desarrollo de Unity que facilita la creación de aplicaciones de realidad aumentada (**AR**). Este paquete presenta una interfaz para que la usen los desarrolladores de Unity, pero no implementa ninguna función de realidad aumentada por sí misma. Este mismo permite crear experiencias de Realidad Aumentada tanto en iOS como Android.

- **AR session**

Una escena de realidad aumentada (**AR**) debe incluir un componente **AR Session**. Este último controla el ciclo de vida para toda aplicación AR habilitando o deshabilitando la AR en cada plataforma puesta.

- **AR session Origin**

El propósito de un **AR Session Origin** es el de transformar cada uno de las funciones de trackeo por parte de la aplicación, tales como superficies planas y para conocer la posición, orientación y escala de puntos característicos en cualquier escena de Unity.

- **AR camera manager**

El **AR Camera Manager** habilita diferentes funciones para una cámara de realidad aumentada (**ARCamera**), esto incluye la administración de la cámara del dispositivo a usar, y las propiedades que establecen las estimaciones de la luz.

- **AR Raycast Manager**

También conocido como "**Hit testing**", ray casting te permite determinar dónde interseca un rayo con un objeto rastreable dentro de pruebas con planos o en la nube de puntos. La interfaz de Ray Cast es similar a la del módulo Unity Physics, pero dado que los rastreables de RA no necesariamente tienen presencia en el mundo de la física, **AR Foundation** proporciona una interfaz separada.

5.2. AR Core

ARCore es una plataforma de realidad aumentada desarrollada por Google para dispositivos Android. Su objetivo principal es permitir a los desarrolladores crear experiencias de realidad aumentada (AR) en una amplia variedad de dispositivos Android, ofreciendo funciones avanzadas de seguimiento y detección de entorno sin necesidad de hardware adicional.

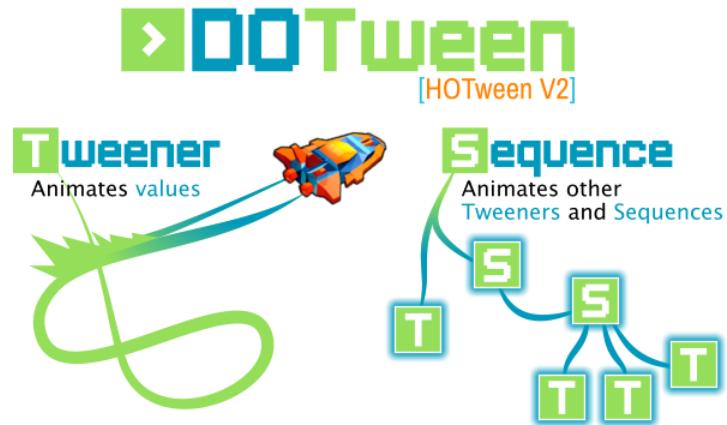


Figura[7]: Logo de ARCore de Google

Además de identificar puntos clave, ARCore puede detectar superficies planas, como una tabla o el piso, y también puede estimar la iluminación promedio en el área a su alrededor. Estas funciones se combinan para permitir que ARCore cree su propia comprensión del mundo que lo rodea.

5.3. DoTween

DOTween es una biblioteca popular para Unity que simplifica y facilita la animación en el desarrollo de videojuegos. "DOTween" proviene de "**Do Tweening**" (Hacer Interpolación), lo que refleja su función principal de realizar interpolación para crear animaciones suaves y fluidas.



Figura[8]: Logo de DOTween
[DOTween \(HOTween v2\) \(demigiant.com\)](http://demigiant.com)

5.4. UniGLTF

UnityGLTF es una librería dentro de Unity3D para importar y exportar GLTF 2.0 assets. El objetivo de dicha librería es dar soporte a todo archivo GLTF. Dicha librería será usada para poder encontrar diferentes variedades de modelos y así tener una gama mayor de objetos a utilizar en el proyecto.

6. Scripts en Unity

Para el funcionamiento de nuestra aplicación AR-Anatomy se diseñaron varios códigos para el correcto funcionamiento de la aplicación. Estos scripts son hechos en Visual Studio Code y hechos en el lenguaje de programación C#.

6.1. Game Manager

Este script es una implementación simple de la arquitectura que hablamos en una de las secciones anteriormente de la aplicación en Unity, que nos dará un buen diseño y manejo.

```
1      using System.Collections;
2      using System.Collections.Generic;
3      using UnityEngine;
4      using System;
5
6  public class GameManager : MonoBehaviour
7  {
8      public event Action OnStart;
9      public event Action OnDetection;
10     public event Action OnDetectionIncorrect;
11     public event Action OnMainMenu;
12     public event Action OnARPosition;
13
14     public static GameManager instance;
15
16     private void Awake()
17     {
18         if (instance != null && instance != this)
19         {
20             Destroy(gameObject);
21         }
22         else
23         {
24             instance = this;
25         }
26     }
27     //Al iniciar, estara con el evento StarApp
28     void Start()
29     {
30         StartApp();
31     }
```

GameManager class es una clase que hereda de MonoBehaviour, **public event Action** se están declarando varios eventos, **public static GameManager instance** declara una instancia estática de la clase **GameManager**. **Awake()** se llama cuando se inicializa el script. **Start()** método de Unity que se llama en el primer frame de ejecución.

```

32     //Métodos que activan los eventos
33     public void StartApp()
34     {
35         OnStart?.Invoke();
36         Debug.Log("Inicio Activado");
37     }
38
39     public void MainMenu()
40     {
41         OnMainMenu?.Invoke();
42         Debug.Log("Menu Principal Activado");
43     }
44
45     public void ARPosition()
46     {
47         OnARPosition?.Invoke();
48         Debug.Log("ARPosicion Activado");
49     }
50
51     public void Detection()
52     {
53         OnDetection?.Invoke();
54         Debug.Log("Deteccion Activado");
55     }
56
57     public void DetectionIncorrect()
58     {
59         OnDetectionIncorrect?.Invoke();
60         Debug.Log("Deteccion Incorrecta Activado");
61     }
62
63     public void CloseApp()
64     {
65         Application.Quit();
66     }
67 }
```

StartApp() activa el evento **OnStart**. **MainMenu()** al llamar a este método, se dispara el evento **OnMainMenu**. **ARPosition()** este método activa el evento **OnARPosition**. **Detection()** al llamar a este método, se activa el evento **OnDetection**. **DetectionIncorrect()** este método desencadena el evento **OnDetectionIncorrect**. **CloseApp()** este método simplemente cierra la aplicación, utilizando la función **Application.Quit()**.

6.2. Data Manager

Este script está destinado a la gestión de botones del menú principal como su imagen, el modelo que lo contiene, el nombre y que se generen al entrar al inicio. Una ventaja es que se puede ser actualizado constantemente para futuras versiones.

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class ButtonsManager : MonoBehaviour
7  {
8      [SerializeField] private List<Models3D> models3d = new List<Models3D>();
9      [SerializeField] private GameObject buttonContainer;
10     [SerializeField] private ModelsButton modelsButton;
11
12     void Start()
13     {
14         GameManager.instance.OnMainMenu += CreateButtons;
15     }
16
17     private void CreateButtons()
18     {
19         foreach (var model in models3d)
20         {
21             ModelsButton modelbutton;
22             modelbutton = Instantiate(modelsButton, buttonContainer.transform);
23             modelbutton.ModelName = model.ModelName;
24             modelbutton.ModelImage = model.ModelImage;
25             modelbutton.Model3D = model.Model3D;
26             modelbutton.name = model.ModelName;
27         }
28     }
29 }
```

En el método **Start**, se instancia al evento **OnMainMenu** del **GameManager** llamando al método **CreateButtons**.

```
16     private void CreateButtons()
17     {
18         foreach (var model in models3d)
19         {
20             ModelsButton modelbutton;
21             modelbutton = Instantiate(modelsButton, buttonContainer.transform);
22             modelbutton.ModelName = model.ModelName;
23             modelbutton.ModelImage = model.ModelImage;
24             modelbutton.Model3D = model.Model3D;
25             modelbutton.name = model.ModelName;
26         }
27     }
28 }
29 }
```

Este método se encarga de crear botones en la interfaz de usuario (UI) para cada modelo 3D en la lista **models3D**. Para cada modelo, se instala un nuevo botón (**ModelsButton**) y se configuran sus propiedades (nombre, imagen, modelo 3D) según la información del modelo actual.

6.3. UI Manager

Este script es una parte de la gestión de la interfaz de usuario (UI) en Unity, utilizamos el paquete DoTween para la animación entre estados o canvas para que sea más cómodo al ser usada por el usuario.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using DG.Tweening;
5  using System;
6  using UnityEngine.UI;
7
8  public class UIManager : MonoBehaviour
9  {
10    public static UIManager instance;
11    [SerializeField]
12    private GameObject inicioCanvas;
13    [SerializeField]
14    private GameObject menuPrincipalCanvas;
15    [SerializeField]
16    private GameObject posicionModelosCanvas;
17    [SerializeField]
18    private GameObject deteccionCanvas;
19    [SerializeField]
20    private GameObject deteccionIncorrectCanvas;
21    void Start()
22    {
23      instance = this;
24      GameManager.instance.OnStart += ActivateStartApp;
25      GameManager.instance.OnMainMenu += ActivateMainMenu;
26      GameManager.instance.OnARPosition += ActivateARPosition;
27      GameManager.instance.OnDetection += ActivateDetection;
28      GameManager.instance.OnDetectionIncorrect += ActivateDetectionIncorrect;
29    }

```

Implementa un patrón Singleton para garantizar una única instancia globalmente accesible. En el método Start, se suscribe a eventos específicos del GameManager para activar funciones de gestión de transición de cada estado o canvas del interfaz de usuario

```

1  public void ActivateDetectionIncorrect()
2  {
3    deteccionCanvas.transform.GetChild(0).transform.DOScale(new Vector3(4, 4, 4), 0.001f);
4    deteccionIncorrectCanvas.transform.GetChild(0).transform.DOScale(new Vector3(1, 1, 1), 0.4f);
5    deteccionIncorrectCanvas.transform.GetChild(1).transform.DOScale(new Vector3(1, 1, 1), 0.4f);
6  }
7  public void ActivateDetection()
8  {
9    //Este canvas está arriba, fuera de la pantalla
10   inicioCanvas.transform.GetChild(0).transform.DOMoveY(4500, 0.5f);
11   inicioCanvas.transform.GetChild(1).transform.DOMoveY(4500, 0.5f);
12   deteccionCanvas.transform.GetChild(0).transform.DOScale(new Vector3(1, 1, 1), 0.1f);
13   deteccionIncorrectCanvas.transform.GetChild(0).transform.DOScale(new Vector3(0, 0, 0), 0.001f);
14   deteccionIncorrectCanvas.transform.GetChild(1).transform.DOScale(new Vector3(0, 0, 0), 0.001f);
15   menuPrincipalCanvas.transform.GetChild(0).transform.DOScale(new Vector3(0, 0, 0), 0.001f);
16   menuPrincipalCanvas.transform.GetChild(1).transform.DOScale(new Vector3(0, 0, 0), 0.001f);
17   menuPrincipalCanvas.transform.GetChild(2).transform.DOScale(new Vector3(0, 0, 0), 0.001f);
18 }

```

Este Script es parte de la gestión de la interfaz de usuario (UI) de la aplicación AR. Mediante los assets proporcionados por Dotween se agregan animaciones a los cambios de canvas.

6.4. Plane Manager

Este script está relacionado con la gestión de planos (lo configuramos para que sea planos horizontales) en una aplicación realidad aumentada y que se utilizara en los siguientes scripts. Utilizamos principalmente el paquete ARFoundation.

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using Unity.VisualScripting;
5  using UnityEngine;
6  using UnityEngine.SceneManagement;
7  using UnityEngine.UI;
8  using UnityEngine.XR.ARFoundation;
9
10 public class PlaneManager : MonoBehaviour
11 {
12     [SerializeField] private ARPlaneManager aRPlaneManager;
13     [SerializeField] private Button buttonStarDetection;
14     [SerializeField] private Button buttonAddPlanes;
15     [SerializeField] private Button buttonDeleteAll;
16
17     int CantPlane = 1;
18     private List<ARPlane> planes = new List<ARPlane>();
19
20     private void OnEnable()
21     {
22         aRPlaneManager.planesChanged += PlanesFound;
23     }
24
25     private void OnDisable()
26     {
27         aRPlaneManager.planesChanged -= PlanesFound;
28     }
}
```

Estos métodos se llaman automáticamente cuando el script se habilita y deshabilita, y agrega planos o quita respectivamente.

```
1  private void PlanesFound(ARPlanesChangedEventArgs planeData)
2  {
3      if (planeData != null && planeData.added.Count > 0)
4      {
5          planes.AddRange(planeData.added);
6          if (planes.Count >= CantPlane) // Comprueba si se ha alcanzado el umbral
7          {
8              CantPlane++;
9              UIManager.instance.ActivateMainMenu();
10         }
11     else
12     {
13         UIManager.instance.ActivateDetectionIncorrect();
14     }
15 }
16 }
```

Este método se llama cuando hay cambios en los planos detectados por el **ARPlaneManager**. Si se detectan nuevos planos (**planeData.added.Count > 0**), se agregan a la lista "planes". Si la cantidad de planos detectados es mayor o igual al contador **CantPlane**, se incrementa **CantPlane** y se activa el menú principal. En caso contrario, se activa una interfaz indicando una detección incorrecta.

```

1      void Start()
2      {
3          buttonStarDetection.onClick.AddListener(StartDetection);
4          buttonAddPlanes.onClick.AddListener(StartDetection);
5          buttonDeleteAll.onClick.AddListener(StopPlaneDetection);
6      }
7      public void StartDetection()
8      {
9          aRPlaneManager.enabled = true;
10         UIManager.instance.ActivateDetection();
11     }
12     public void StopPlaneDetection()
13     {
14         //Eliminar planos detectados
15         Debug.Log("Se eliminó los planos");
16         aRPlaneManager.requestedDetectionMode =
17         UnityEngine.XR.ARSubsystems.PlaneDetectionMode.None;
18         foreach (var plane in planes)
19         {
20             plane.gameObject.SetActive(false);
21         }
22         CantPlane = 0;
23     }
24 }
```

"**StartDetection**" activa la detección de planos (**ARPlaneManager.enabled = true**) y activa la interfaz de detección en el **UIManager**.

"**StopPlaneDetection**" desactiva la detección de planos, oculta los planos detectados y restablece el contador **CantPlane**.

6.5. Models 3D

Este script crea un tipo de scriptable object (**Models3D**) que contiene información sobre cada modelo 3D, su nombre específico, la imagen asociada a su respectivo botón y una referencia al **GameObject** que representa el modelo en sí.

```

1      using System.Collections;
2      using System.Collections.Generic;
3      using UnityEngine;
4
5      [CreateAssetMenu]
6      public class Models3D : ScriptableObject
7      {
8          public string ModelName;
9          public Sprite ModelImage;
10         public GameObject Model3D;
11     }
```

6.6. Models Button

Este script se encarga de gestionar los botones asociados a los modelos 3D para la aplicación de AR. Cuando el botón es clicado, se ejecuta el método Create3DModel, que instancia un modelo 3D asociado.

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  //using UnityEditor.SearchService;
5  using UnityEngine;
6  using UnityEngine.UI;
7
8  public class ModelsButton : MonoBehaviour
9  {
10    private string modelName;
11    private Sprite modelImage;
12    private GameObject model3D;
13    private ARInteractionManager interactionManager;
14    public string ModelName { set => modelName = value; }
15    public Sprite ModelImage { set => modelImage = value; }
16    public GameObject Model3D { set => model3D = value; }
17
18    void Start()
19    {
20      transform.GetChild(0).GetComponent<RawImage>().texture = modelImage.texture;
21      var button = GetComponent<Button>();
22      button.onClick.AddListener(GameManager.instance.ARPosition);
23      button.onClick.AddListener(Create3DModel);
24      interactionManager = FindObjectOfType<ARInteractionManager>();
25    }
26
27    private void Create3DModel()
28    {
29      interactionManager.Model3D = Instantiate(model3D);
30    }
31  }
```

6.7. Share Screenshot

Este script se encarga de tomar una captura de pantalla (screenshot) por medio de presionar un botón con icono de cámara en el estado del menú principal y compartirlo a través de aplicaciones nativas en dispositivos móviles.

```

1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using System.IO;
5   using UnityEngine;
6   using UnityEngine.XR.ARFoundation;
7
8   public class ShareScreenShot : MonoBehaviour
9   {
10      [SerializeField] private GameObject mainMenuCanvas;
11      private ARPointCloudManager aRPointCloudManager;
12
13      void Start()
14      {
15          aRPointCloudManager = FindObjectOfType<ARPointCloudManager>();
16      }
17
18      public void TakeScreenShot()
19      {
20          TurnOnOffARContents();
21          StartCoroutine(TakeScreenshotAndShare());
22      }
23
24      private void TurnOnOffARContents()
25      {
26          // Implement logic to turn on/off AR contents
27          // For example:
28          if (mainMenuCanvas != null)
29          {
30              mainMenuCanvas.SetActive(!mainMenuCanvas.activeSelf);
31          }
32      }

```

Definimos una clase pública **ShareScreenShot** que maneja la captura y el intercambio de capturas. Método **Void Start()** busca el **ARPointCloudManager** en la escena al iniciar. **TakeScreenShot()** activa/desactiva los contenidos, luego inicia la captura y el proceso de intercambio de capturas de pantalla.

```

1   private IEnumerator TakeScreenshotAndShare()
2   {
3       yield return new WaitForEndOfFrame();
4       Texture2D ss = new Texture2D(Screen.width, Screen.height, TextureFormat.RGB24, false);
5       ss.ReadPixels(new Rect(0, 0, Screen.width, Screen.height), 0, 0);
6       ss.Apply();
7       string filePath = Path.Combine(Application.temporaryCachePath, "shared img.png");
8       File.WriteAllBytes(filePath, ss.EncodeToPNG());
9       // To avoid memory leaks
10      Destroy(ss);
11      new NativeShare().AddFile(filePath)
12          .SetSubject("Subject goes here").SetText("Hey, how do you like the image?")
13          .SetCallback((result, shareTarget) => Debug.Log("Share result: " + result + ", selected
14          app: " + shareTarget))
15          .Share();
16      TurnOnOffARContents();
17  }

```

La captura se toma en una corutina **TakeScreenshotAndShare()** se guarda como una imagen PNG en la caché temporal. Finalmente vuelve a activar contenidos AR después de compartir la captura

6.8. Button Toggle

Este script se declara la clase **ButtonToggle**, ésta representa un interruptor en Unity.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5
6  public class ButtonToggle :
7  MonoBehaviour
8  {
9      bool currentState;
10     [SerializeField]
11     UnityEvent turnedOn;
12     [SerializeField]
13     UnityEvent turnedOff;
14
15     public void ToggleState()
16     {
17         currentState = !currentState;
18
19         if (currentState == true)
20         {
21             TurnOn();
22         }
23         else
24         {
25             TurnOff();
26         }
27     }
28
29     public void TurnOn()
30     {
31         currentState = true;
32         turnedOn?.Invoke();
33     }
34
35     public void TurnOff()
36     {
37         currentState = false;
38         turnedOff?.Invoke();
39     }
40 }
```

El método **ToggleState** invierte el estado actual del interruptor (**currentState**). Si estaba encendido, lo apagaba; si estaba apagado, lo encendía.

El método **TurnOn** establece el estado del interruptor en "encendido" y activa el evento **turnedOn**, esto asegura que el evento solo se invoque si no es nulo. En el caso del método **TurnOff** el interruptor se establecerá en el estado "apagado" y activará su evento.

6.9. UI Animation Button

Este script está diseñado para manejar la animación del movimiento de los botones al presionar otro botón. Incluye posición inicial y final además, de la animación que el programador le parezca más cómodo.

```
1      using System.Collections;
2      using System.Collections.Generic;
3      using UnityEngine;
4
5      public class UIAnimationButton : MonoBehaviour
6      {
7
8          [SerializeField]
9          float duration;
10         [SerializeField]
11         float delay;
12
13         [SerializeField]
14         AnimationCurve animationCurve;
15         [SerializeField]
16         RectTransform target;
17
18         [SerializeField]
19         Vector3 startingPoint;
20         [SerializeField]
21         Vector3 finalPoint;
22         [ContextMenu("Fade in")]
23         public void FadeIn()
24         {
25             StopAllCoroutines();
26             StartCoroutine(FadeInCoroutine(startingPoint, finalPoint));
27         }
28         [ContextMenu("Fade out")]
29         public void FadeOut()
30         {
31             StopAllCoroutines();
32             StartCoroutine(FadeOutCoroutine(startingPoint, finalPoint));
33         }
```

Se señalan los objetos entrantes para la animación de los botones luego se definen los métodos “**FadeIn**” y “**FadeOut**” que detienen todas las rutinas con **StopAllCoroutines()** y comienza una nueva coroutine para iniciar la animación con el método **FadeInCoroutine()** y **FadeOutCoroutine()**

```

1  IEnumerator FadeInCoroutine(Vector3 a, Vector3 b)
2  {
3      Vector3 staringPoint = a;
4      Vector3 finalPoint = b;
5      float elapsed = 0;
6      while (elapsed <= delay)
7      {
8          elapsed += Time.deltaTime;
9          yield return null;
10     }
11     elapsed = 0;
12     while (elapsed <= duration)
13     {
14         float percentage = elapsed / duration;
15         float curvePercentage = animationCurve.Evaluate(percentage);
16         elapsed += Time.deltaTime;
17         Vector3 currentPosition = Vector3.LerpUnclamped(staringPoint, finalPoint, curvePercentage);
18         target.anchoredPosition = currentPosition;
19         yield return null;
20     }
21     target.anchoredPosition = finalPoint;
22 }
23
24 IEnumerator FadeOutCoroutine(Vector2 a, Vector2 b)
25 {
26     Vector3 staringPoint = a;
27     Vector3 finalPoint = b;
28     float elapsed = 0;
29     while (elapsed <= delay)
30     {
31         elapsed += Time.deltaTime;
32         yield return null;
33     }
34     elapsed = 0;
35     while (elapsed <= duration)
36     {
37         float percentage = elapsed / duration;
38         float curvePercentage = animationCurve.Evaluate(percentage);
39         elapsed += Time.deltaTime;
40         Vector3 currentPosition = Vector3.LerpUnclamped(finalPoint, staringPoint, curvePercentage);
41         target.anchoredPosition = currentPosition;
42         yield return null;
43     }
44     target.anchoredPosition = staringPoint;
45 }
46 }
```

Esta sección del script controla y maneja la lógica específica de las animaciones. Se encargan del retardado, cálculo del porcentaje de avance, evaluación de la curva de animación, interpolación entre puntos inicial y final, y actualización de la posición de RectTransform durante el tiempo de la animación.

6.10. AR Interaction Manager

Este script está diseñado para manejar la interacción del usuario con el modelo 3D que se genera, interacción como mover y rotar el modelo en nuestra aplicación en unity

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.XR.ARFoundation;
5  using UnityEngine.XR.ARSubsystems;
6  using UnityEventSystems;
7
8  public class ARInteractionManager : MonoBehaviour
9  {
10     [SerializeField] private Camera ARCamera;
11     private ARRaycastManager ARRaycastManager;
12     private List<ARRaycastHit> hits = new List<ARRaycastHit>();
13
14     private GameObject ARPointer;
15     private GameObject model3D;
16     private GameObject SelectModel;
17     private bool InitialPosition;
18     private bool OverUI;
19     private bool is3DModelTouch;
20
21     private Vector2 PosicionInicialTouch;
22     public GameObject Model3D{
23
24         set{
25             model3D = value;
26             model3D.transform.position = ARPointer.transform.position;
27             model3D.transform.parent = ARPointer.transform;
28             InitialPosition = true;
29         }
30     }
}
```

Establece el **modelo 3D** proporcionado por cada **scriptable object**. Cuando se asigna un valor, posiciona el modelo en el **AR Pointer** y lo hace hijo del ARPointer.

```
1  void Start()
2  {
3      ARPointer = transform.GetChild(0).gameObject;
4      ARRaycastManager = FindObjectOfType<ARRaycastManager>();
5      GameManager.instance.OnMainMenu += SetPosition;
6  }
```

Inicializa variables y registra un evento para el menú principal.

```

1   void Update()
2   {
3       if(InitialPosition)
4       {
5           Vector2 middlePointScreen = new Vector2(Screen.width / 2, Screen.height / 2);
6           ARRaycastManager.Raycast(middlePointScreen, hits, TrackableType.Planes);
7           if (hits.Count>0)
8           {
9               transform.position = hits[0].pose.position;
10              transform.rotation = hits[0].pose.rotation;
11              ARPointer.SetActive(true);
12              InitialPosition = false;
13          }
14      }
15
16      if (Input.touchCount == 1)
17      {
18          Touch touchOne = Input.GetTouch(0);
19          if(touchOne.phase == TouchPhase.Began){
20              var touchPosition = touchOne.position;
21              OverUI = TapOverUI(touchPosition);
22              is3DModelTouch = TapOverModel3D(touchPosition);
23          }
24
25          if(touchOne.phase == TouchPhase.Moved){
26              if(ARRaycastManager.Raycast(touchOne.position, hits, TrackableType.Planes))
27              {
28                  Pose hitPose = hits[0].pose;
29                  if(!OverUI && is3DModelTouch){
30                      transform.position = hitPose.position;
31
32                  }
33              }
34          }
}

```

Si el **InitialPosition** está en true acciona un rayo desde el centro de la pantalla (**middlePointScreen**) hacia el plano detectado por AR.

Si hay intersección (**hits.Count > 0**), coloca el objeto **ARPointer** en esa posición y orientación. Si hay un solo toque (**Input.touchCount == 1**). Al inicio del toque, se determina si está sobre la interfaz de usuario o un modelo 3D. Luego, si existe algún movimiento, ajusta la posición del modelo sobre el plano creado.

```

1   if(Input.touchCount == 2)
2   {
3       Touch touchTwo = Input.GetTouch(1);
4       if (touchOne.phase == TouchPhase.Began || touchTwo.phase == TouchPhase.Began)
5       {
6           PosicionInicialTouch = touchTwo.position - touchOne.position;
7       }
8       if (touchOne.phase == TouchPhase.Moved || touchTwo.phase == TouchPhase.Moved)
9       {
10          Vector2 currentTouch = touchTwo.position - touchOne.position;
11          float angulo = Vector2.SignedAngle(PosicionInicialTouch, currentTouch);
12          model3D.transform.rotation = Quaternion.Euler(0,model3D.transform.eulerAngles.y -
13          angulo, 0);
14          PosicionInicialTouch = currentTouch;
15      }
16  }
17  if(is3DModelTouch && model3D == null && !OverUI)
18  {
19      GameManager.instance.ARPosition();
20      model3D = SelectModel;
21      SelectModel = null;
22      ARPointer.SetActive(true);
23      transform.position = model3D.transform.position;
24      model3D.transform.parent = ARPointer.transform;
25  }
26 }
27 }
```

Si hay dos toques (**Input.touchCount == 2**). Al inicio de los toques, guarda la posición inicial relativa de los toques realizados por el usuario. Luego, cuando se acciona un movimiento, calcula el ángulo entre la posición actual y la inicial para rotar el modelo 3D.

```

1   private bool TapOverUI(Vector2 touchPosition){
2
3       PointerEventData evento = new PointerEventData(EventSystem.current);
4       evento.position = new Vector2(touchPosition.x, touchPosition.y);
5
6       List<RaycastResult> resultado = new List<RaycastResult>();
7       EventSystem.current.RaycastAll(evento, resultado);
8
9       return resultado.Count > 0;
10 }
```

Este método verifica si un toque se realiza sobre la interfaz de usuario.

```

1   private void SetPosition()
2   {
3       if (model3D != null)
4       {
5           model3D.transform.parent = null;
6           ARPointer.SetActive(false);
7           model3D = null;
8       }
9   }
```

Este método restablece la posición del modelo 3D al volver al menú principal.

```

1     private bool TapOverModel3D (Vector2 touchPosition){
2         Ray ray = ARCamera.ScreenPointToRay(touchPosition);
3
4         if (Physics.Raycast(ray, out RaycastHit hit3DModel))
5         {
6             if(hit3DModel.collider.CompareTag("Item"))
7             {
8                 SelectModel = hit3DModel.transform.gameObject;
9                 return true;
10            }
11        }
12
13        return false;
14    }

```

En este método verifica si un toque se realiza sobre un modelo 3D.

```

1     public void DeleteModel()
2     {
3         Destroy(model3D);
4         ARPointer.SetActive(false);
5         GameManager.instance.MainMenu();
6     }
7

```

En esta parte elimina el modelo 3D actual y vuelve al menú principal.

7. Casos de Uso y Ejemplos prácticos

La AR enfocada en anatomía ofrece aplicaciones diversas y transformadoras en campos como la educación médica, donde facilita la comprensión tridimensional del cuerpo humano y mejora la interactividad en el aprendizaje. En el ámbito médico, la AR se emplea para simular procedimientos y entrenar a profesionales, así como para proporcionar asistencia visual durante cirugías en tiempo real. Además, la tecnología tiene aplicaciones en consultas médicas, investigación, rehabilitación, desarrollo de fármacos, diseño de dispositivos médicos y prevención de enfermedades. Desde la visualización de datos complejos hasta la creación de experiencias educativas interactivas, la AR anatomy tiene el potencial de transformar la práctica médica y la comprensión del cuerpo humano en diversos contextos.

8. Observaciones

- ❖ La arquitectura basada en eventos es acertada para una comunicación eficiente en la aplicación de realidad aumentada. Se recomienda ejemplificar cómo mejora la experiencia del usuario y la eficiencia del desarrollo.
- ❖ La elección de AR Foundation en Unity asegura compatibilidad con iOS y Android, beneficiando los objetivos y la enseñanza de anatomía. Detallar cómo esta elección beneficia la aplicación sería útil.
- ❖ AR Core mejora la experiencia en dispositivos Android; proporcionar ejemplos específicos de su integración sería beneficioso para entender la anatomía tridimensional.
- ❖ DoTween para animación es una elección positiva; describir cómo contribuye a la experiencia del usuario en contextos específicos sería útil.

- ❖ Recomendación de incluir desafíos enfrentados, cómo se abordaron y lecciones aprendidas para futuros proyectos de realidad aumentada en el ámbito médico y educativo.
- ❖ Considerar pruebas piloto con estudiantes de medicina para evaluar la efectividad de la aplicación en aprendizaje y práctica médica.
- ❖ Explorar oportunidades futuras, como la integración de nuevas tecnologías o colaboraciones con instituciones médicas, para validar y mejorar la aplicación en entornos educativos y clínicos.

9. Conclusiones

- La RA ha demostrado mejorar el desempeño académico en anatomía al ayudar a los estudiantes a identificar, ubicar y comprender las relaciones entre estructuras anatómicas.
- En entornos médicos, la RA puede ser utilizada para simular procedimientos médicos y prácticas quirúrgicas, reduciendo los riesgos asociados con la práctica en pacientes reales y mejorando la destreza de los profesionales de la salud.
- La tecnología de realidad aumentada puede adaptarse a diversos niveles educativos, desde la enseñanza básica hasta la formación avanzada en medicina, brindando una herramienta versátil para educadores y profesionales de la salud.

10. Referencias

- 10.1. Technologies, U. (s. f.). *Marco de trabajo AR Foundation de Unity*. Unity. <https://unity.com/es/unity/features/arfoundation>
- 10.2. KhronosGroup. (s. f.). *GitHub - KhronosGroup/UnityGLTF: Runtime GLTF 2.0 Loader for Unity3D*. GitHub. <https://github.com/KhronosGroup/UnityGLTF>
- 10.3. Sketchfab. (s. f.). *Log in to your Sketchfab account*. <https://sketchfab.com/feed>
- 10.4. *DOTween (HOTween v2)*. (s. f.). <http://dotween.demigiant.com/>

11. Repositorio

Para acceder al proyecto AR-Anatomy realizado en Unity, el archivo instalador de la aplicación, además de este informe de implementación y el informe teórico anterior, puede visitar el repositorio de GitHub:

“<https://github.com/Jharvichu/AR-Anatomy>”

La rama en la cual realizamos el proyecto inicialmente fue la *main*, pero luego se continuó a la rama *RamaPrincipal* dado que se había creado una rama diferente llamada *respaldo* en caso ocurriera un error en el proyecto.