TUGAS AKHIR SEMESTER DESAIN AR/VR

AR VR ANIMAL SOUND

Dosen Pengampu : Fizar Syafa'at, S.kom, M.kom

Disusun Oleh:

Kelompok 8

Muhammad Al-Faraby Moidady_F55123018 (Pengerjaan Laporan, QA/Tester)

Sofia Qatrunnada _F55123035 (AR/VR Specialist, Presenter/Demo)

Moh Maulana Yahya –-F55123037 (Lead Developer, Project Leader)

Nur Aisyah _F55123042 (AR/VR Specialist, Presenter/Demo)

Yogi Alamsyah_F55119053 (Pengerjaan Laporan, QA/Tester)

Kelas A

PROGRAM STUDI S1 TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

FAKULTAS TEKNIK

UNIVERSITAS TADULAKO

2025

ABSTRAK

Sistem ini merupakan media interaktif berbasis Augmented Reality (AR) yang memanfaatkan image tracking untuk memunculkan lingkungan 3D berupa pulau kecil berisi beberapa hewan. Ketika marker dipindai oleh kamera smartphone, sistem AR Foundation pada Unity melakukan pelacakan posisi dan rotasi marker, kemudian menampilkan pulau beserta objek hewan di atasnya. Setiap hewan dilengkapi collider, audio source, dan skrip interaksi sehingga dapat merespons sentuhan pengguna. Saat hewan diklik, aplikasi memicu pemutaran suara khas hewan sekaligus animasi gerakan sederhana sehingga tercipta ilusi hewan hidup yang "bergerak dan bersuara" di atas permukaan nyata. Implementasi ini menunjukkan bagaimana kombinasi AR, input sentuh, dan audio spasial dapat digunakan untuk membangun pengalaman interaktif yang imersif tanpa bergantung pada engine pihak ketiga seperti Vuforia. Secara potensial, sistem ini dapat dikembangkan lebih lanjut sebagai media edukasi pengenalan hewan atau diperluas ke mode VR untuk interaksi fisika yang lebih kompleks.

1. TUJUAN DAN SCOPE

a. Tujuan

Berdasarkan tema *"Virtual Buttons (AR) + Spatial Audio & Physics-Based Interaction (VR)"*, tujuan sistem yang dikembangkan adalah:

1) Menerapkan konsep virtual buttons pada media cetak (brosur/packaging) menggunakan AR Foundation, sehingga ketika marker dipindai kamera smartphone, tiap objek/hewan di atas pulau 3D dapat berfungsi sebagai tombol virtual yang memicu aksi tertentu.
2) Menghasilkan pengalaman interaksi yang lebih hidup melalui **s**uara spasial dan gerak fisika sederhana pada objek hewan ketika tombol virtual tersebut disentuh, sebagai representasi awal dari skenario interaksi VR berbasis fisika.
3) Menyusun arsitektur dan alur kerja (*flow*) AR→VR yang dapat menjadi dasar pengembangan lebih lanjut menuju lingkungan VR penuh dengan interaksi objek yang realistis.

b. Scope

Ruang lingkup tugas dibatasi pada:

1) Implementasi AR pada satu image target berupa brosur/packaging yang, ketika dipindai, memunculkan sebuah pulau kecil berisi beberapa hewan 3D; setiap hewan

berperan sebagai virtual button.

2) Interaksi yang diimplementasikan adalah: deteksi sentuhan pada hewan, pemutaran suara spasial (AudioSource 3D) dan animasi/gerakan fisika sederhana (misalnya hewan tersenggol lalu bergerak/lari).
3) Platform yang digunakan adalah smartphone Android berbasis ARCore dengan Unity 6, AR Foundation, dan alur desain yang disiapkan agar dapat diteruskan ke mode VR; integrasi ke perangkat VR dedicated dan fitur lanjutan (multi-marker, multi-user, UI kompleks) berada di luar scope prototipe ini dan hanya dibahas sebagai pengembangan ke depan.

2. METODE DAN TOOLS

a. Metode Teknis

Secara teknis, sistem dibangun dengan pendekatan *image-tracking–based AR* menggunakan AR Foundation. Ketika kamera smartphone diarahkan ke brosur/packaging, ARCore melakukan deteksi dan pelacakan *image target*, lalu mengirimkan data posisi–rotasi marker ke Unity sebagai *anchor* 3D. Di atas anchor tersebut di-*spawn* sebuah prefab pulau kecil yang berisi beberapa hewan 3D. Interaksi sentuhan diimplementasikan dengan membaca posisi tap pada layar, kemudian mengubahnya menjadi *raycast* dari kamera AR ke ruang 3D. Jika raycast mengenai *collider* salah satu hewan, skrip interaksi memicu dua aksi sekaligus: (1) memutar *AudioSource* 3D sebagai suara khas hewan (*spatial audio*), dan (2) menjalankan animasi gerak sederhana berbasis transformasi dan/atau *Rigidbody* (misalnya efek tersenggol lalu bergerak/lari). Alur ini sekaligus disusun agar dapat diperluas ke skenario VR dengan interaksi fisika yang lebih kompleks.

b. Tools / Software

Tools utama yang digunakan adalah:

1) Unity 6.2 (editor utama pengembangan aplikasi).
2) AR Foundation dan ARCore XR Plugin sebagai kerangka kerja AR berbasis *image tracking* pada Android.
3) Android SDk dan Android Build Support untuk proses build ke perangkat Android
4) Unity Input System / Touch Input untuk membaca interaksi sentuhan
5) Unity Asset Store (atau sumber aset bebas lain) untuk model 3D hewan dan aset pendukung

6) Editor kode (Visual Studio Code) untuk penulisan skrip C#

c. Lingkungan Pengembangan

Lingkungan pengembangan yang digunakan meliputi:

1) Hardware: laptop kelas menengah dengan prosesor multi–core, RAM minimal 8 GB, dan GPU terintegrasi/dedicated yang mendukung Unity 3D; serta satu smartphone Android yang kompatibel dengan ARCore sebagai perangkat uji.
2) Sistem operasi: Windows 10/11 64-bit sebagai OS utama pengembangan.
3) Software pengembang: Unity 6.2 dengan modul Android Build Support, AR Foundation + ARCore XR Plugin, Android SDK/ADB untuk *deploy* dan debugging ke perangkat fisik.

3. ARSITEKTUR SISTEM DAN FLOW AR→VR

a. Arsitektur Sistem

1) Lapisan Input dan Sensing

a) Kamera Smartphone (AR Foundation)
   • Menangkap gambar dunia nyata.
   • ARCore/ARKit melakukan tracking marker (image target).
   • Memberikan posisi & rotasi marker ke Unity
b) Input Sentuhan Layar (Touch Input System)
   • Mengambil posisi sentuhan (screen point).
   • Mengubah sentuhan menjadi raycast ke objek AR.
c) Sensor Gerak Perangkat (IMU)
   • Gyroscope dan accelerometer untuk tracking kamera AR.

2) Lapisan Proses dan Logika Aplikasi

a) AR Subsystem (AR Foundation)

   • Mengelola *image tracking.*

• Mengubah marker menjadi anchor 3D di ruang AR.

b) *Object Spawning System*

• Prefab ayam dimunculkan pada posisi image targer.

• Mengatur orientasi dan skala objek.

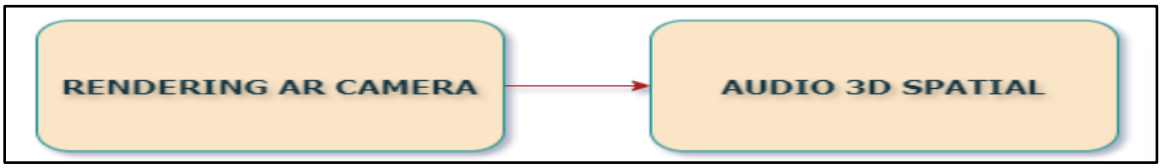
c) *Physics & Interaction Engine*

• Collider pada objek ayam menerima raycast.

• Saat collider kena raycast maka event “OnTouch” terjadi.

• AudioSource memutar suara

3) Lapisan Output dan *Rendering*



a) AR Subsysem (AR Foundation

• Mengelola image tracking.

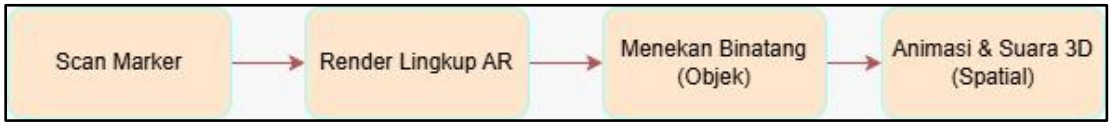
• Mengubah marker menjadi anchor 3D di ruang AR

b) Object Spawning System

• Prefab ayam dimunculkan pada posisi image targer.

• Mengatur orientasi dan skala objek.

b. Flow AR→ VR



c. Modul Sistem

1) Unity Engine

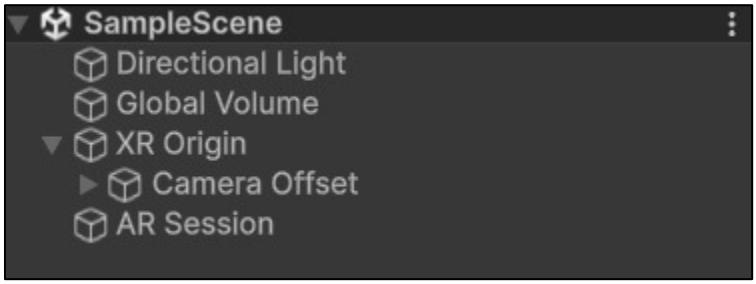
2) AR Foundation

3) AR Core (Android

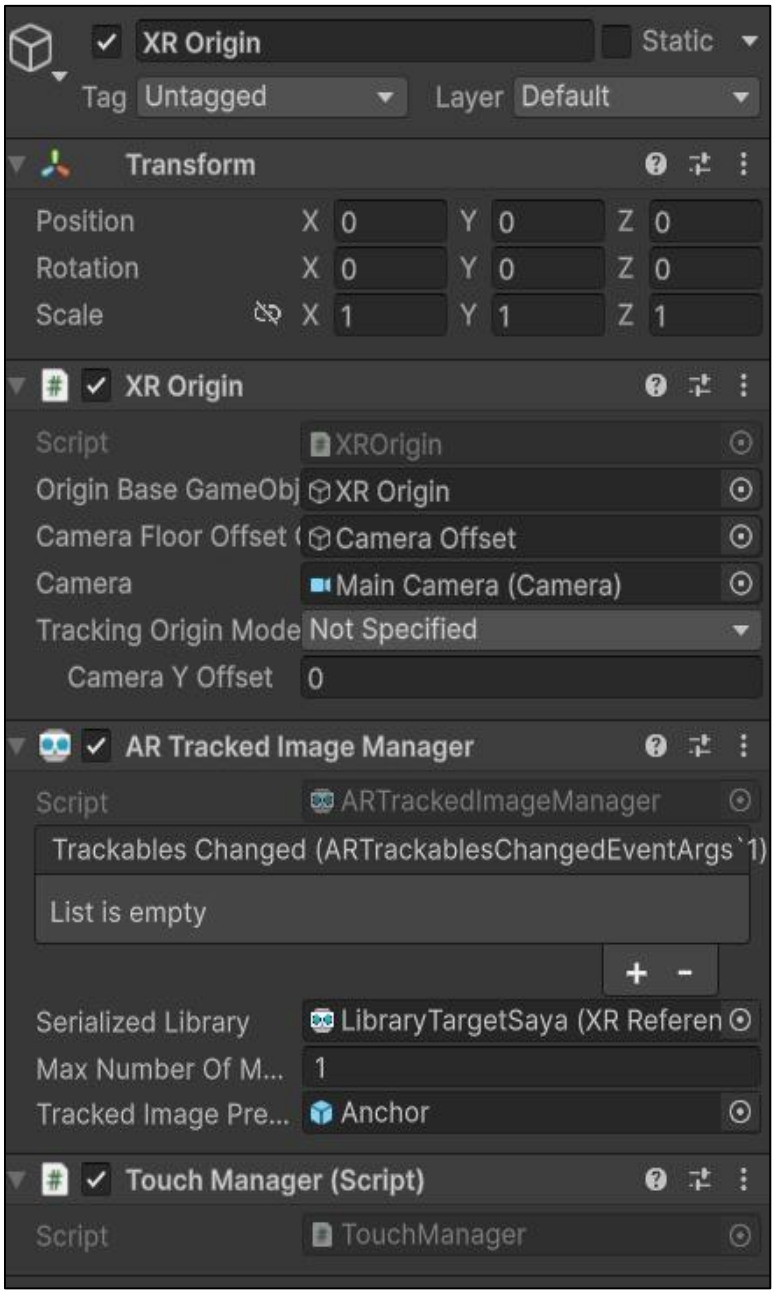
4. IMPLEMENTASI

a. Struktur Proyek

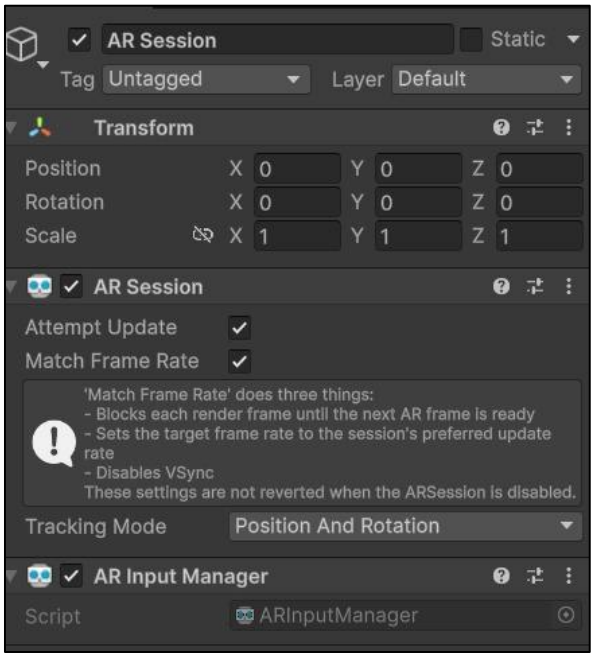
1) Hirarki



2) XR Origin



3) Ar Session



b. Kode

1) Image Tracking

```
using System;
using System.Collections.Generic;
using Unity.Jobs;
using UnityEngine.Serialization;
using UnityEngine.XR.ARSubsystems;
using Unity.XR.CoreUtils;

namespace UnityEngine.XR.ARFoundation
{
    /// <summary>
    /// A manager for <see cref="ARTrackedImage"/>s. Uses the <c>XRImageTrackingSubsystem</c>
    /// to recognize and track 2D images in the physical environment.
    /// </summary>
    /// <remarks>
    /// Related information: <a href="xref:arfoundation-image-tracking">AR Tracked Image Manager component</a>
    /// </remarks>
    [DefaultExecutionOrder(ARUpdateOrder.k_TrackedImageManager)]
    [RequireComponent(typeof(XROrigin))]
    [AddComponentMenu("XR/AR Foundation/AR Tracked Image Manager")]
    [HelpURL("features/image-tracking")]
    public sealed class ARTrackedImageManager : ARTrackableManager<
        XRImageTrackingSubsystem,
        XRImageTrackingSubsystemDescriptor,
        XRImageTrackingSubsystem.Provider,
        XRTrackedImage,
        ARTrackedImage>
    {
        [SerializeField]
        [FormerlySerializedAs("m_ReferenceLibrary")]
        [Tooltip("The library of images which will be detected and/or tracked in the physical environment.")]
        XRReferenceImageLibrary m_SerializedLibrary;

        /// <summary>
        /// Get or set the reference image library (that is, the set of images to search for in the physical environment).
        /// </summary>
        /// <remarks>
        /// An <c>IReferenceImageLibrary</c> can be either an <c>XRReferenceImageLibrary</c>
        /// or a <c>RuntimeReferenceImageLibrary</c>. <c>XRReferenceImageLibrary</c>s can only be
        /// constructed in the Editor and are immutable at runtime. A <c>RuntimeReferenceImageLibrary</c>
        /// is the runtime representation of a <c>XRReferenceImageLibrary</c> and can be mutable
        /// at runtime (see <c>MutableRuntimeReferenceImageLibrary</c>).
        /// </remarks>
```

```
43      /// <exception cref="System.InvalidOperationException">Thrown if the <see cref="referenceLibrary"/> is set to <c>null</c> while image tracking is enabled.</exception>
44      public IReferenceImageLibrary referenceLibrary
45      {
46          get
47          {
48              if (subsystem != null)
49              {
50                  return subsystem.imageLibrary;
51              }
52              else
53              {
54                  return m_SerializedLibrary;
55              }
56          }
57
58          set
59          {
60              if (value == null && subsystem != null && subsystem.running)
61                  throw new InvalidOperationException("Cannot set a null reference library while image tracking is enabled.");
62
63              if (value is XRReferenceImageLibrary serializedLibrary)
64              {
65                  m_SerializedLibrary = serializedLibrary;
66                  if (subsystem != null)
67                      subsystem.imageLibrary = subsystem.CreateRuntimeLibrary(serializedLibrary);
68              }
69              else if (value is RuntimeReferenceImageLibrary runtimeLibrary)
70              {
71                  m_SerializedLibrary = null;
72                  EnsureSubsystemInstanceSet();
73
74                  if (subsystem != null)
75                      subsystem.imageLibrary = runtimeLibrary;
76              }
77
78              if (subsystem != null)
79                  UpdateReferenceImages(subsystem.imageLibrary);
80          }
81      }
```

```
83      /// <summary>
84      /// Creates a <c>UnityEngine.XR.ARSubsystems.RuntimeReferenceImageLibrary</c> from an existing
85      /// <c>UnityEngine.XR.ARSubsystems.XRReferenceImageLibrary</c>
86      /// or an empty library if <paramref name="serializedLibrary"/> is <c>null</c>.
87      /// Use this to construct reference image libraries at runtime. If the library is of type
88      /// <c>MutableRuntimeReferenceImageLibrary</c>, it is modifiable at runtime.
89      /// </summary>
90      /// <param name="serializedLibrary">An existing <c>XRReferenceImageLibrary</c>, or <c>null</c> to create an empty mutable image library.</param>
91      /// <returns>A new <c>RuntimeReferenceImageLibrary</c> representing the deserialized version of <paramref name="serializedLibrary"/> or an empty library if <paramref name="serializedLibrary"/> is <c>null</c>.</returns>
92      /// <exception cref="System.NotSupportedException">Thrown if there is no subsystem. This usually means image tracking is not supported.</exception>
93      public RuntimeReferenceImageLibrary CreateRuntimeLibrary(XRReferenceImageLibrary serializedLibrary = null)
94      {
95          EnsureSubsystemInstanceSet();
96
97          if (subsystem == null)
98              throw new NotSupportedException("No image tracking subsystem found. This usually means image tracking is not supported.");
99
100         return subsystem.CreateRuntimeLibrary(serializedLibrary);
101     }
102
103     [SerializeField]
104     [Tooltip("The maximum number of moving images to track in realtime. Not all implementations support this feature.")]
105     int m_MaxNumberOfMovingImages;
106
107     bool supportsMovingImages => descriptor?.supportsMovingImages == true;
108
109     /// <summary>
110     /// The requested maximum number of moving images to track in real time. Support can vary between devices and providers. Check
111     /// for support at runtime with <see cref="SubsystemLifecycleManager{TSubsystem,TSubsystemDescriptor,TProvider}.descriptor"/>'s
112     /// `supportsMovingImages` property.
113     /// </summary>
114     public int requestedMaxNumberOfMovingImages
115     {
116         get => supportsMovingImages ? subsystem.requestedMaxNumberOfMovingImages : m_MaxNumberOfMovingImages;
117         set
118         {
119             m_MaxNumberOfMovingImages = value;
120             if (enabled && (descriptor?.supportsMovingImages == true))
121             {
122                 subsystem.requestedMaxNumberOfMovingImages = value;
123             }
124         }
125     }
126
127     /// <summary>
128     /// Get the maximum number of moving images to track in real time that is currently in use by the subsystem.
129     /// </summary>
130     public int currentMaxNumberOfMovingImages => supportsMovingImages ? subsystem.currentMaxNumberOfMovingImages : 0;
```

```
        [SerializeField]
        [Tooltip("If not null, instantiates this prefab for each detected image.")]
        GameObject m_TrackedImagePrefab;

        /// <summary>
        /// If not null, instantiates this Prefab for each detected image.
        /// </summary>
        /// <remarks>
        /// The purpose of this property is to extend the functionality of <see cref="ARTrackedImage"/>s.
        /// It is not the recommended way to instantiate content associated with an <see cref="ARTrackedImage"/>.
        /// </remarks>
        public GameObject trackedImagePrefab
        {
            get => m_TrackedImagePrefab;
            set => m_TrackedImagePrefab = value;
        }

        /// <summary>
        /// Get the Prefab that will be instantiated for each <see cref="ARTrackedImage"/>.
        /// </summary>
        /// <returns>The Prefab that will be instantiated for each <see cref="ARTrackedImage"/>.</returns>
        protected override GameObject GetPrefab() => m_TrackedImagePrefab;

        /// <summary>
        /// Invoked once per frame with information about the <see cref="ARTrackedImage"/>s that have changed (that is, been added, updated, or removed).
        /// This happens just before <see cref="ARTrackedImage"/>s are destroyed, so you can set <c>ARTrackedImage.destroyOnRemoval</c> to <c>false</c>
        /// from this event to suppress this behavior.
        /// </summary>
        [Obsolete("trackedImagesChanged has been deprecated in AR Foundation version 6.0. Use trackablesChanged instead.", false)]
        public event Action<ARTrackedImagesChangedEventArgs> trackedImagesChanged;

        /// <summary>
        /// The name to be used for the <c>GameObject</c> whenever a new image is detected.
        /// </summary>
        protected override string gameObjectName => nameof(ARTrackedImage);

        /// <summary>
        /// Sets the image library on the subsystem before Start() is called on the <c>XRImageTrackingSubsystem</c>.
        /// </summary>
```

```
        protected override void OnBeforeStart()
        {
            if (subsystem.imageLibrary == null && m_SerializedLibrary != null)
            {
                subsystem.imageLibrary = subsystem.CreateRuntimeLibrary(m_SerializedLibrary);
                m_SerializedLibrary = null;
            }

            UpdateReferenceImages(subsystem.imageLibrary);
            if (supportsMovingImages)
            {
                subsystem.requestedMaxNumberOfMovingImages = m_MaxNumberOfMovingImages;
            }

            enabled = (subsystem.imageLibrary != null);
#if DEVELOPMENT_BUILD
            if (subsystem.imageLibrary == null)
            {
                Debug.LogWarning($"{nameof(ARTrackedImageManager)} '{name}' was enabled but no reference image library is specified. To enable, set a valid reference image library and then re-enable this component.");
            }
#endif
        }

        bool FindReferenceImage(Guid guid, out XRReferenceImage referenceImage)
        {
            if (m_ReferenceImages.TryGetValue(guid, out referenceImage))
                return true;

            // If we are using a mutable library, then it's possible an image
            // has been added that we don't yet know about, so search the library.
            if (referenceLibrary is MutableRuntimeReferenceImageLibrary mutableLibrary)
            {
                foreach (var candidateImage in mutableLibrary)
                {
                    if (candidateImage.guid.Equals(guid))
                    {
                        referenceImage = candidateImage;
                        m_ReferenceImages.Add(referenceImage.guid, referenceImage);
                        return true;
                    }
                }
            }

            return false;
        }
```

```
        /// <summary>
        /// Invoked just after updating each <see cref="ARTrackedImage"/>. Used to update the <see cref="ARTrackedImage.referenceImage"/>.
        /// </summary>
        /// <param name="image">The tracked image being updated.</param>
        /// <param name="sessionRelativeData">New data associated with the tracked image. Spatial data is
        /// relative to the <see cref="XROrigin"/>.</param>
        protected override void OnAfterSetSessionRelativeData(
            ARTrackedImage image,
            XRTrackedImage sessionRelativeData)
        {
            if (FindReferenceImage(sessionRelativeData.sourceImageId, out XRReferenceImage referenceImage))
            {
                image.referenceImage = referenceImage;
            }
#if DEVELOPMENT_BUILD
            else
            {
                Debug.LogError($"Could not find reference image with guid {sessionRelativeData.sourceImageId}");
            }
#endif
        }

        /// <summary>
        /// Invokes the <see cref="trackedImagesChanged"/> event.
        /// </summary>
        /// <param name="added">A list of images added this frame.</param>
        /// <param name="updated">A list of images updated this frame.</param>
        /// <param name="removed">A list of images removed this frame.</param>
        [Obsolete("OnTrackablesChanged() has been deprecated in AR Foundation version 6.0.", false)]
        protected override void OnTrackablesChanged(
            List<ARTrackedImage> added,
            List<ARTrackedImage> updated,
            List<ARTrackedImage> removed)
        {
            if (trackedImagesChanged != null)
            {
                using (new ScopedProfiler("OnTrackedImagesChanged"))
                trackedImagesChanged?.Invoke(
                    new ARTrackedImagesChangedEventArgs(
                        added,
                        updated,
                        removed));
            }
        }
```

```
            void UpdateReferenceImages(RuntimeReferenceImageLibrary library)
            {
                if (library == null)
                    return;

                int count = library.count;
                for (int i = 0; i < count; ++i)
                {
                    var referenceImage = library[i];
                    m_ReferenceImages[referenceImage.guid] = referenceImage;
                }
            }

            Dictionary<Guid, XRReferenceImage> m_ReferenceImages = new Dictionary<Guid, XRReferenceImage>();
        }
    }
```

2) Touch Detection

Assets > Scripts > TouchManager.cs

```
using UnityEngine;
using UnityEngine.XR.ARFoundation; // Wajib untuk AR

public class TouchManager : MonoBehaviour
{
    private Camera arCamera;

    void Start()
    {
        arCamera = Camera.main; // Mengambil kamera utama
    }

    void Update()
    {
        // Cek apakah ada sentuhan jari di layar
        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began)
        {
            // Tembakkan Ray (Garis invisible) dari posisi sentuhan
            Ray ray = arCamera.ScreenPointToRay(Input.GetTouch(0).position);
            RaycastHit hit;

            // Jika Ray mengenai sesuatu yang punya Collider
            if (Physics.Raycast(ray, out hit))
            {
                // Cek apakah yang kena itu Hewan?
                if (hit.transform.CompareTag("Animal"))
                {
                    // Panggil fungsi Interact yang ada di hewan tersebut
                    AnimalBehavior animalScript = hit.transform.GetComponent<AnimalBehavior>();
                    if (animalScript != null)
                    {
                        animalScript.Interact();
                    }
                }
            }
        }
    }
}
```

3) Resources Behaviour

```
using UnityEngine;

public class AnimalBehavior : MonoBehaviour
{
    private Animator anim;
    private AudioSource audioSource;

    void Start()
    {
        anim = GetComponent<Animator>();
        audioSource = GetComponent<AudioSource>();
    }

    // Fungsi ini akan dipanggil saat hewan disentuh
    public void Interact()
    {
        // 1. Mainkan Animasi (Pastikan di Animator ada Parameter Trigger bernama "Action")
        if(anim != null) anim.SetTrigger("Action");

        // 2. Mainkan Suara
        if(audioSource != null) audioSource.Play();

        // 3. Efek Fisika (Loncat dikit biar kerasa 'Physics'-nya)
        Rigidbody rb = GetComponent<Rigidbody>();
        if(rb != null)
        {
            rb.AddForce(Vector3.up * 100f); // Dorong ke atas sedikit
        }
    }
}
```

c. Algoritma

4) Image Tracking

ARTrackedImageManager adalah komponen utama AR Foundation yang digunakan untuk mendeteksi dan melacak gambar 2D di dunia nyata menggunakan XRImageTrackingSubsystem. Komponen ini bekerja dengan memuat reference image library, yaitu kumpulan gambar yang telah ditentukan sebelumnya sebagai target deteksi. Library ini bisa berupa XRReferenceImageLibrary yang dibuat lewat Unity Editor atau RuntimeReferenceImageLibrary yang dapat dimodifikasi saat aplikasi sudah berjalan. Ketika aplikasi dimulai, manager ini memastikan library dimasukkan ke subsystem image tracking dan menyiapkan semua referensi gambar yang akan dipakai selama proses deteksi. Selama sesi AR berlangsung, kamera akan mencari kecocokan antara gambar nyata dan gambar yang ada di library. Jika sebuah gambar terdeteksi, sistem akan membuat atau memperbarui objek ARTrackedImage yang mewakili gambar tersebut di ruang AR. Jika sebuah prefab telah ditentukan, prefab tersebut akan diinstansiasi lalu ditempatkan tepat di atas gambar yang terdeteksi. Manager ini juga menangani pembaruan posisi, rotasi, atau keadaan tracking dari setiap

gambar yang sedang terdeteksi. Selain itu, kelas ini menyediakan dukungan untuk perangkat yang mampu melacak gambar bergerak, dengan memberikan pengaturan jumlah maksimum gambar bergerak yang boleh dilacak secara bersamaan. Seluruh referensi gambar secara internal disimpan dalam dictionary untuk memastikan pencarian gambar lebih cepat, terutama ketika subsystem mengirimkan data baru mengenai gambar yang dilacak. Setiap kali data tracking diperbarui, ARTrackedImageManager akan mencocokkan ID gambar yang dilaporkan subsystem dengan gambar di library, kemudian memperbarui objek ARTrackedImage yang sesuai. Meskipun event lama seperti trackedImagesChanged sudah ditandai sebagai deprecated, mekanisme notifikasi ini sebelumnya digunakan untuk memberi tahu perubahan gambar yang baru terdeteksi, diperbarui, atau hilang. Secara keseluruhan, komponen ini menjadi pusat proses image tracking pada AR Foundation, mulai dari memuat library, mendeteksi marker, memperbarui trackable, hingga menampilkan konten AR di atas gambar target.

5) Touch Detection

Kode TouchManager ini berfungsi untuk mendeteksi sentuhan pengguna pada layar dalam aplikasi AR, lalu menerjemahkannya menjadi interaksi dengan objek 3D di dalam scene. Pada awalnya, *script* mengambil kamera utama (Camera.main), yang biasanya merupakan kamera AR bawaan AR Foundation. Setiap frame, kode memeriksa apakah ada sentuhan yang baru dimulai di layar. Jika ada, *script* membuat sebuah Ray dari posisi sentuhan tersebut menuju ke dalam dunia 3D. Ray ini bekerja sebagai garis tak terlihat yang digunakan untuk mengecek apakah pengguna menyentuh sebuah objek tertentu. Jika raycast mengenai objek yang memiliki collider, *script* kemudian memeriksa apakah objek tersebut memiliki tag "Animal", yang menandakan bahwa objek tersebut adalah hewan yang bisa diinteraksikan. Jika tag-nya sesuai, *script* mengambil komponen AnimalBehavior dari objek tersebut. Jika komponen itu ditemukan, fungsi Interact() dipanggil untuk menjalankan aksi yang sudah didefinisikan pada hewan, seperti animasi, suara, atau perilaku tertentu. Kode ini menjadi dasar sistem interaksi berbasis sentuhan dalam aplikasi AR yang menggunakan AR Foundation.

6) Resources Behaviour

Script AnimalBehavior ini digunakan untuk mengatur bagaimana sebuah hewan bereaksi ketika disentuh dalam aplikasi AR. Pada saat objek hewan mulai aktif, *script*

mengambil komponen Animator dan AudioSource yang menempel pada objek tersebut. Ketika fungsi Interact() dipanggil, biasanya setelah pengguna menyentuh hewan melalui raycast, *script* akan menjalankan tiga respons sekaligus. Pertama, jika objek memiliki animator, *script* memicu parameter trigger bernama "Action" sehingga hewan menampilkan animasi tertentu seperti melompat atau bergerak. Kedua, jika terdapat audio source, *script* akan memutar suara hewan untuk memberikan efek interaksi yang lebih hidup. Ketiga, *script* menambahkan sedikit gaya ke arah atas menggunakan Rigidbody agar hewan terlihat seperti bergerak secara fisik sehingga interaksinya terasa lebih natural dan responsif. Secara keseluruhan, *script* ini menggabungkan animasi, suara, dan efek fisika untuk menciptakan reaksi hewan yang menarik ketika disentuh di lingkungan AR.

5. PENGUJIAN DAN HASIL

a. Metode Pengujian

Functional Testing yang memuat *Marker Scanning Test, Screen Toucing Test,* dan *Sound Effect Test*

b. Hasil SSQ

Tanggapan dari para pengguna yang telah mencoba aplikasi ini menunjukkan bahwa tingkat ketidaknyamanan yang dirasakan berada pada kategori rendah. Secara umum, sebagian besar pengguna yang telah mencoba aplikasi ini menyatakan bahwa mereka tidak mengalami kebingungan ataupun gangguan selama proses penggunaan. Alur interaksi, tampilan antarmuka, serta respons sistem dinilai cukup jelas sehingga pengguna dapat menavigasi fitur-fitur yang tersedia tanpa hambatan berarti. Hasil ini mengindikasikan bahwa aspek kenyamanan dan kemudahan penggunaan pada aplikasi telah memenuhi ekspektasi awal.

c. Performa (FPS dan *Latency*)

Hasil pengujian performa menunjukkan bahwa aplikasi ini mampu berjalan secara stabil pada berbagai perangkat yang mendukung fitur AR, baik yang memiliki spesifikasi rendah maupun tinggi. Proses pendeteksian marker oleh kamera berlangsung dengan baik, yang terlihat dari waktu respons yang cepat dalam menampilkan objek AR. Selain itu, kinerja aplikasi juga terjaga stabil, ditunjukkan dengan pencapaian frame rate yang konsisten pada 60 FPS (*Frame Per Second*) tanpa mengalami penurunan performa maupun memberikan beban berlebih pada perangkat yang digunakan

6. MASALAH DAN SOLUSI

Selama proses pengembangan, salah satu kendala yang muncul adalah kegagalan model naga untuk merespons interaksi tap, sementara model hewan lain seperti serigala dan laba-laba berfungsi dengan normal. Pemeriksaan telah dilakukan pada bagian animator, collider, dan struktur prefab, namun tidak ditemukan penyebab yang jelas. Permasalahan akhirnya teridentifikasi berasal dari perbedaan kategori atau *tag* yang digunakan pada objek naga, sehingga sistem deteksi interaksi tidak mengenalinya sebagai bagian dari kelompok hewan lain yang telah berfungsi. Solusi dicapai dengan menyamakan *tag* objek naga menjadi 'Animal', sama seperti *resource* lainnya. Setelah penyesuaian tersebut dilakukan, interaksi kembali berjalan dengan normal: animasi dapat terpicu, suara muncul, dan model merespons tap sesuai yang diharapkan. Temuan ini menunjukkan pentingnya konsistensi pengaturan kategori objek dalam sistem interaksi AR

7. KESIMPULAN DAN PENGEMBANGAN LANJUTAN

a. Kesimpulan

Berdasarkan hasil pengujian menggunakan SSQ dan evaluasi performa, dapat disimpulkan bahwa aplikasi AR yang dikembangkan telah memberikan pengalaman penggunaan yang nyaman dan stabil. Tingkat ketidaknyamanan pengguna berada pada kategori rendah, menunjukkan bahwa fitur AR, animasi, dan interaksi yang diterapkan tidak menimbulkan disorientasi, kebingungan, maupun gejala simulator sickness lainnya. Selain itu, performa aplikasi terbukti optimal pada berbagai perangkat, termasuk perangkat dengan spesifikasi rendah. Proses pendeteksian marker berlangsung cepat, objek dapat ditampilkan tanpa jeda yang berarti, dan aplikasi mampu mempertahankan frame rate stabil pada kisaran 60 FPS tanpa menurunkan performa perangkat. Secara keseluruhan, aplikasi AR ini layak digunakan karena memberikan pengalaman yang responsif, nyaman, dan bebas gangguan pada pengguna.

b. Pengembangan Lanjutan

Pengembangan lanjutan dari aplikasi AR hewan ini dapat difokuskan pada peningkatan interaktivitas dan pengalaman imersif pengguna. Fitur gestur tambahan seperti memutar dan memperbesar objek dapat diterapkan untuk memberikan kendali yang lebih natural. Selain itu, aplikasi dapat dikembangkan dengan menambahkan lebih banyak hewan, variasi animasi, serta panel informasi interaktif untuk tujuan edukasi.

DAFTAR PUSTAKA

Machała, S., Chamier-Gliszczyński, N. and Królikowski, T., 2022. Application of AR/VR Technology in Industry 4.0. Procedia Computer Science, 207, pp.2990-299

LISENSI ASET 3D/AUDIO

https://free3d.com/3d-model/wolf-rigged-and-game-ready-42808.html?dd_referrer=

https://free3d.com/3d-model/spider-animated-low-poly-and-game-ready-87147.html

https://free3d.com/3d-model/black-dragon-rigged-and-game-ready-92023.html

LAMPIRAN

Link Github : https://github.com/MulYahya/Kelompok_8

Link Video Demo : https://youtu.be/SwcuL7VQCs0?si=QWF9NXwaly-X8JAd