LAMPIRAN

A. *SCRIPT*

1. Splash Screen

a. Kontroler *Splash Screen*

```
public class splash : MonoBehaviour {

    void Start() {
        StartCoroutine(Example());
    }

    IEnumerator Example() {
        yield return new WaitForSeconds(2);
        Application.LoadLevel ("loading");
    }
}
```

b. Kontroler *Loading*

```
public class loading : MonoBehaviour {

    public Transform LoadingBar;

    [SerializeField] private float currentAmount;
    [SerializeField] private float speed;

        void Update () {
        if (currentAmount < 100) {
            currentAmount += speed * Time.deltaTime;
            Debug.Log ((int)currentAmount);
        } else {
            Application.LoadLevel ("menu_utama");
        }

        LoadingBar.GetComponent<Image> ().fillAmount = currentAmount
/ 100;
    }
}
```

2. Menu Utama

```
public class menu : MonoBehaviour {

    public void GoToMainMenu(){
        Application.LoadLevel("menu_utama");
    }

    public void keARkamerakegiatanku(){
        Application.LoadLevel("animasi");
    }

    public void ExitApplication(){
        Application.Quit ();
    }
}
```

3. AR Kamera

a. Kontroler Sentuhan

```
Namespace Lean.Touch
{
    public class LeanScale : MonoBehaviour
    {
        [Tooltip("Ignore fingers with StartedOverGui?")]
        public bool IgnoreStartedOverGui = true;

        [Tooltip("Ignore fingers with IsOverGui?")]
        public bool IgnoreIsOverGui;

        [Tooltip("Allows you to force rotation with a specific amount
 of fingers (0 = any)")]
        public int RequiredFingerCount;

        [Tooltip("Does scaling require an object to be selected?")]
        public LeanSelectable RequiredSelectable;

        [Tooltip("The camera that will be used to calculate the zoom
(None = MainCamera)")]
        public Camera Camera;

        [Tooltip("If you want the mouse wheel to simulate pinching th
en set the strength of it here")]
        [Range(-1.0f, 1.0f)]
        public float WheelSensitivity;

        [Tooltip("Should the scaling be performanced relative to the
finger center?")]
        public bool Relative;

        [Tooltip("Should the scale value be clamped?")]
        public bool ScaleClamp;

        [Tooltip("The minimum scale value on all axes")]
```

```
        public Vector3 ScaleMin;

        [Tooltip("The maximum scale value on all axes")]
        public Vector3 ScaleMax;

#if UNITY_EDITOR
        protected virtual void Reset()
        {
            Start();
        }
#endif

        protected virtual void Start()
        {
            if (RequiredSelectable == null)
            {
                RequiredSelectable = GetComponent<LeanSelectable>();
            }
        }

        protected virtual void Update()
        {
            var fingers = LeanSelectable.GetFingersOrClear(IgnoreStar
tedOverGui, IgnoreIsOverGui, RequiredFingerCount, RequiredSelectable)
;

                    var pinchScale = LeanGesture.GetPinchScale(finge
rs, WheelSensitivity);

            if (pinchScale > 0.0f)
            {
                if (Relative == true)
                {
                    var pinchScreenCenter = LeanGesture.GetScreenCent
er(fingers);

                    if (transform is RectTransform)
                    {
                        TranslateUI(pinchScale, pinchScreenCenter);
                    }
                    else
                    {
                        Translate(pinchScale, pinchScreenCenter);
                    }
                }

                Scale(transform.localScale * pinchScale);
            }
        }

        protected virtual void TranslateUI(float pinchScale, Vector2
pinchScreenCenter)
        {            var screenPoint = RectTransformUtility.WorldToSc
reenPoint(Camera, transform.position);

            screenPoint.x = pinchScreenCenter.x + (screenPoint.x - pi
nchScreenCenter.x) * pinchScale;
            screenPoint.y = pinchScreenCenter.y + (screenPoint.y - pi
nchScreenCenter.y) * pinchScale;

            var worldPoint = default(Vector3);
```

```
            if (RectTransformUtility.ScreenPointToWorldPointInRectang
le(transform.parent as RectTransform, screenPoint, Camera, out worldP
oint) == true)
            {
                transform.position = worldPoint;
            }
        }

        protected virtual void Translate(float pinchScale, Vector2 sc
reenCenter)
        {
            var camera = LeanTouch.GetCamera(Camera, gameObject);

            if (camera != null)
            {
                var screenPosition = camera.WorldToScreenPoint(transf
orm.position);

                screenPosition.x = screenCenter.x + (screenPosition.x
 - screenCenter.x) * pinchScale;
                screenPosition.y = screenCenter.y + (screenPosition.y
 - screenCenter.y) * pinchScale;

                transform.position = camera.ScreenToWorldPoint(screen
Position);
            }
        }

        protected virtual void Scale(Vector3 scale)
        {
            if (ScaleClamp == true)
            {
                scale.x = Mathf.Clamp(scale.x, ScaleMin.x, ScaleMax.x
);
                scale.y = Mathf.Clamp(scale.y, ScaleMin.y, ScaleMax.y
);
                scale.z = Mathf.Clamp(scale.z, ScaleMin.z, ScaleMax.z
);
            }

            transform.localScale = scale;
        }
    }
}
```

b. Panel Deskripsi

```
public class targetData : MonoBehaviour
    {

        public Transform Deskripsi;
        public Transform PanelDeskripsi;

        void Update()
        {
            StateManager sm = TrackerManager.Instance.GetStateManager
();
```

```
            IEnumerable<TrackableBehaviour> tbs = sm.GetActiveTrackab
leBehaviours();

            foreach (TrackableBehaviour tb in tbs)
            {
                string name = tb.TrackableName;
                ImageTarget it = tb.Trackable as ImageTarget;
                Vector2 size = it.GetSize();

                Debug.Log("Active image target:" + name + "  -
size: " + size.x + ", " + size.y);

                Deskripsi.gameObject.SetActive(true);
                PanelDeskripsi.gameObject.SetActive(true);



                if(name == "tendangan_luar")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Tendang
an Dalam,</i> menggunakan kaki bagian dalam. Teknik menendang bola in
i berguna untuk memberikan umpan kepemain lain dengan jarak yang cuku
p jauh atau long passing.";
                }


                if (name == "tendangan_dalam")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Tendang
an ujung kaki,</i> teknik ini tidak terlalu sering digunakan oleh par
a pemain hanya dalam kondisi tertentu saja seperti saat terjepit dan
tertekan oleh lawan, serta juga bisa dilakukan saat berhadapan satu l
awan satu dengan seorang kiper dari tim lawan.";
                }

                if (name == "tendangan_punggung")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Tendang
an punggung kaki,</i> Pada umumnya menendang dengan punggung kaki dig
unakan untuk menembak ke gawang atau shooting.";
                }

                if (name == "baseball")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Memukul
 bola,</i> ayunkan pemukul bolamu tepat ke arah bola yang dilempar ol
eh temanmu. Pegang kuat-
kuat, jangan sampai pemukul bolanya terlepas.";
                }

                if (name == "drible_basket")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Memantu
lkan bola,</i> untuk memantulkan bola yang perlu kamu lakukan adalah
fokus kepada langkah kakimu serta bolanya sehingga bolanya tidak mema
ntul ke arah yang lain.";
                }

                if (name == "melempar_bola")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Melempa
r bola,</i> fokuslah ke arah temanmu dan lempar bola sekuat-
```

```
kuatnya. Lemparlah bola dengan terarah agar tidak membahayakan dirimu
 dan sekitarmu.";
                }
                if (name == "Meroda")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Meroda,
</i> Bertumpulah dengan tangan kananmu lalu ayunkan kakimu kearah kan
an. Ajaklah salah satu temanmu untuk membantumu meroda. Ingat ketika
kamu mempraktikannya harus diawasi oleh guru.";
                }

                if (name == "melompat")
                {
                    Deskripsi.GetComponent<Text>().text = "<i>Melompa
t,</i> ambilah ancang-
ancang yang cukup kemudian melompatlah. Ketika kamu mendarat gunakanl
ah kedua kakimu sebagai tumpuan.";
                }

            }
        }
    }
```

c. Kontroler utama

```
namespace Vuforia
{
    /// <summary>
    /// A custom handler that implements the ITrackableEventHandler
interface.
    /// </summary>
    public class DefaultTrackableEventHandler : MonoBehaviour,
                                                ITrackableEventHandler
    {

        //------------Begin Sound----------
        public AudioSource soundTarget;
        public AudioClip clipTarget;
        private AudioSource[] allAudioSources;

        //function to stop all sounds
        void StopAllAudio()
        {
            allAudioSources = FindObjectsOfType(typeof(AudioSource))
as AudioSource[];
            foreach (AudioSource audioS in allAudioSources)
            {
                audioS.Stop();
            }
        }

        //function to play sound
        void playSound(string ss)
        {
            clipTarget = (AudioClip)Resources.Load(ss);
            soundTarget.clip = clipTarget;
            soundTarget.loop = false;
            soundTarget.playOnAwake = false;
```

```
            soundTarget.Play();
        }

        //-----------End Sound------------


        #region PRIVATE_MEMBER_VARIABLES

        private TrackableBehaviour mTrackableBehaviour;

        #endregion // PRIVATE_MEMBER_VARIABLES



        #region UNTIY_MONOBEHAVIOUR_METHODS

        void Start()
        {
            mTrackableBehaviour = GetComponent<TrackableBehaviour>();
            if (mTrackableBehaviour)
            {
mTrackableBehaviour.RegisterTrackableEventHandler(this);
            }

            //Register / add the AudioSource as object
            soundTarget =
(AudioSource)gameObject.AddComponent<AudioSource>();
        }

        #endregion // UNTIY_MONOBEHAVIOUR_METHODS



        #region PUBLIC_METHODS

        /// <summary>
        /// Implementation of the ITrackableEventHandler function
called when the
        /// tracking state changes.
        /// </summary>
        public void OnTrackableStateChanged(
                                        TrackableBehaviour.Status
previousStatus,
                                        TrackableBehaviour.Status
newStatus)
        {
            if (newStatus == TrackableBehaviour.Status.DETECTED ||
                newStatus == TrackableBehaviour.Status.TRACKED ||
                newStatus ==
TrackableBehaviour.Status.EXTENDED_TRACKED)
            {
                OnTrackingFound();
            }
            else
            {
                OnTrackingLost();
            }
        }

        #endregion // PUBLIC_METHODS
```

```
        #region PRIVATE_METHODS


        private void OnTrackingFound()
        {
            Renderer[] rendererComponents =
GetComponentsInChildren<Renderer>(true);
            Collider[] colliderComponents =
GetComponentsInChildren<Collider>(true);

            // Enable rendering:
            foreach (Renderer component in rendererComponents)
            {
                component.enabled = true;
            }

            // Enable colliders:
            foreach (Collider component in colliderComponents)
            {
                component.enabled = true;
            }

            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName
+ " found");
```