DAFTAR LAMPIRAN

Lampiran - 1 Listing Program

VrLookWalk.Cs

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class VrLookWalk : MonoBehaviour

{

    public Transform vrCamera;

    public float toggleAngle = 30.0f;

    public float speed = 3.0f;

    public bool moveForward;



    private CharacterController cc;



    // Use this for initialization

    void Start()

    {

        cc = GetComponent<CharacterController>();

    }

    // Update is called once per frame

    void Update()

    {

        if (vrCamera.eulerAngles.x >= toggleAngle && vrCamera.eulerAngles.x < 90.0f)

        {

            moveForward = true;

        }

        else

        {

            moveForward = false;

        }

        if (moveForward)

        {

            Vector3 forward = vrCamera.TransformDirection(Vector3.forward);

            cc.SimpleMove(forward * speed);

        }

    }

}
```

MenuUTama.Cs

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;



public class MenuUTama : MonoBehaviour

{

    public AudioSource buttonsound;

    public Canvas MenuUtama;

    public Canvas Profil1;

    public Canvas Profil2;

    public Canvas petunjuk1;

    public Canvas
```

```
petunjuk2;

    public Canvas
petunjuk3;

    public Canvas
petunjuk4;

    public Canvas
petunjuk5;

    public Canvas
petunjuk6;

    public Canvas
petunjuk7;

    // Start is called before
the first frame update

    public void
Menuutama()

    {


buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= true;

        Profil1.enabled =
false;

        Profil2.enabled =
false;

        petunjuk1.enabled =
false;

        petunjuk2.enabled =
false;

        petunjuk3.enabled =
false;

        petunjuk4.enabled =
false;

        petunjuk5.enabled =
false;

        petunjuk6.enabled =
false;

        petunjuk7.enabled =
false;

    }

    public void profi1()

    {

buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= false;

        Profil1.enabled =
true;

        Profil2.enabled =
false;

    }

    public void profi2()

    {

buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= false;

        Profil1.enabled =
false;

        Profil2.enabled =
true;

    }

    public void Petunjuk1()

    {



buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= false;



        petunjuk1.enabled =
true;

        petunjuk2.enabled =
false;

        petunjuk3.enabled =
false;

        petunjuk4.enabled =
false;

        petunjuk5.enabled =
false;

        petunjuk6.enabled =
false;

        petunjuk7.enabled =
false;

    }

    public void Petunjuk2()

    {



buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= false;



        petunjuk1.enabled =
false;

        petunjuk2.enabled =
true;

        petunjuk3.enabled =
```

```
false;
        petunjuk4.enabled =
false;
        petunjuk5.enabled =
false;
        petunjuk6.enabled =
false;
        petunjuk7.enabled =
false;
    }
    public void Petunjuk3()
    {

buttonsound.PlayOneShot(
buttonsound.clip);
        MenuUtama.enabled
= false;

        petunjuk1.enabled =
false;
        petunjuk2.enabled =
false;
        petunjuk3.enabled =
true;
        petunjuk4.enabled =
false;
        petunjuk5.enabled =
false;
        petunjuk6.enabled =
false;
        petunjuk7.enabled =
false;
    }
    public void Petunjuk4()
    {

buttonsound.PlayOneShot(
buttonsound.clip);
        MenuUtama.enabled
= false;

        petunjuk1.enabled =
false;
        petunjuk2.enabled =
false;
        petunjuk3.enabled =
false;
        petunjuk4.enabled =
true;
        petunjuk5.enabled =
false;
        petunjuk6.enabled =
false;
        petunjuk7.enabled =
false;
    }
    public void Petunjuk5()
    {

buttonsound.PlayOneShot(
buttonsound.clip);
        MenuUtama.enabled
= false;

        petunjuk1.enabled =
false;
        petunjuk2.enabled =
false;
        petunjuk3.enabled =
false;
        petunjuk4.enabled =
false;
        petunjuk5.enabled =
true;
        petunjuk6.enabled =
false;
        petunjuk7.enabled =
false;
    }
    public void Petunjuk6()
    {

buttonsound.PlayOneShot(
buttonsound.clip);
        MenuUtama.enabled
= false;

        petunjuk1.enabled =
false;
        petunjuk2.enabled =
false;
        petunjuk3.enabled =
false;
        petunjuk4.enabled =
false;
```

```
        petunjuk5.enabled =
false;

        petunjuk6.enabled =
true;

        petunjuk7.enabled =
false;

    }

    public void Petunjuk7()

    {



buttonsound.PlayOneShot(
buttonsound.clip);

        MenuUtama.enabled
= false;



        petunjuk1.enabled =
false;

        petunjuk2.enabled =
false;

        petunjuk3.enabled =
false;

        petunjuk4.enabled =
false;

        petunjuk5.enabled =
false;

        petunjuk6.enabled =
false;

        petunjuk7.enabled =
true;

    }



    public void Modevr()

    {



SceneManager.LoadScene(
"Rumah Anggrek");

    }



    public void keluar()

    {

        Application.Quit();

        Debug.Log("Aplikasi
Keluar");

    }

}
```

Mainmenu.Cs

```
using System.Collections;

using
System.Collections.Generi
c;

using UnityEngine;

using
UnityEngine.SceneManage
ment;

using
UnityEngine.UIElements;

public class MainMenu2 :
MonoBehaviour

{

    public AudioSource
buttonsound;

    public Canvas
BukaMenu;

    public Canvas Menu_1;

    public Canvas Menu_2;

    public Canvas Menu_3;

    public GameObject[]
images;

    private int currentIndex
= 0;

    void Start()

    {

        BukaMenu.enabled =
true;

        Menu_1.enabled =
false;

        Menu_2.enabled =
false;

        Menu_3.enabled =
false;



        for (int i = 0; i <
images.Length; i++)

        {


images[i].SetActive(i ==
currentIndex);

        }

    }

    public void menu_1()

    {


buttonsound.PlayOneShot(
buttonsound.clip);
```

```
        BukaMenu.enabled =
false;

        Menu_1.enabled =
true;

        Menu_2.enabled =
false;

        Menu_3.enabled =
false;

    }

    public void menu_2()

    {


buttonsound.PlayOneShot(
buttonsound.clip);

        BukaMenu.enabled =
false;

        Menu_1.enabled =
false;

        Menu_2.enabled =
true;

        Menu_3.enabled =
false;

    }

    public void menu_3()

    {


buttonsound.PlayOneShot(
buttonsound.clip);

        BukaMenu.enabled =
false;

        Menu_1.enabled =
false;

        Menu_2.enabled =
false;

        Menu_3.enabled =
true;

    }

    public void bukamenu()

    {


buttonsound.PlayOneShot(
buttonsound.clip);

        BukaMenu.enabled =
true;

        Menu_1.enabled =
false;

        Menu_2.enabled =
false;

        Menu_3.enabled =
false;

    }

    public void NextImage()

    {


images[currentIndex].SetA
ctive(false);

        currentIndex =
(currentIndex + 1) %
images.Length;


images[currentIndex].SetA
ctive(true);

    }

    public void
PreviousImage()

    {


images[currentIndex].SetA
ctive(false);

        currentIndex =
(currentIndex - 1 +
images.Length) %
images.Length;


images[currentIndex].SetA
ctive(true);

    }

    public void Menuutama(

    {


buttonsound.PlayOneShot(
buttonsound.clip);


SceneManager.LoadScene(
"MenuUtama");

    }

}
```

ButtonHoverText.Cs

```
using UnityEngine;

using TMPro;

using
UnityEngine.EventSystem
s;


public class
ButtonHoverText :
MonoBehaviour,
IPointerEnterHandler,
IPointerExitHandler

{

    public
TextMeshProUGUI
hoverText; // Referensi ke
```

```
TextMeshProUGUI yang
ingin ditampilkan


    void Start()

    {

        if (hoverText != null)

        {


hoverText.gameObject.Set
Active(false); //
Sembunyikan teks di awal

        }

    }

    public void
OnPointerEnter(PointerEv
entData eventData)

    {

        if (hoverText != null)

        {


hoverText.gameObject.Set
Active(true); // Tampilkan
teks saat tombol disorot

        }

    }

    public void
OnPointerExit(PointerEve
ntData eventData)

    {

        if (hoverText != null)

        {


hoverText.gameObject.Set
Active(false); //
Sembunyikan teks saat
tidak disorot

        }

    }

}
```

KetTumbuhan.Cs

```
using System.Collections;
using
System.Collections.Generi
c;
using UnityEngine;
using
UnityEngine.SceneManage
ment;
using UnityEngine.UI;


public class KetTumbuhan
: MonoBehaviour
{
    public AudioSource
buttonsound;
    public AudioSource
Dendrobium_Rantii;
    public AudioSource
Coelogyne_celebensis;
    public AudioSource
Phalaenopsis_amabilis;
    public AudioSource
Ascocentrum_miniatum;
    public AudioSource
Calanthe_triplicata;
    public Canvas Ket1;
    public Canvas Ket2;
    public Canvas Ket3;
    public Canvas Ket4;
    public Canvas Ket5;
    public Canvas Info1;
    public Canvas Info2;
    public Canvas Info3;
    public Canvas Info4;
    public Canvas Info5;
    // Start is called before
the first frame update
    void Start()
    {
        Ket1.enabled = false;
        Ket2.enabled = false;
        Ket3.enabled = false;
        Ket4.enabled = false;
        Ket5.enabled = false;
        Info1.enabled = true;
        Info2.enabled = true;
        Info3.enabled = true;
        Info4.enabled = true;
        Info5.enabled = true;
    }

    public void ket_1()
    {

Dendrobium_Rantii.PlayO
neShot(Dendrobium_Ranti
i.clip);
        Info1.enabled = false;
        Ket1.enabled = true;
    }
    public void info_1()
    {

buttonsound.PlayOneShot(
buttonsound.clip);

Dendrobium_Rantii.Stop()
;
        Info1.enabled = true;
        Ket1.enabled = false;
    }
    public void ket_2()
    {

Coelogyne_celebensis.Play
OneShot(Coelogyne_celeb
ensis.clip);
        Info2.enabled = false;
        Ket2.enabled = true;
    }
    public void info_2()
    {

buttonsound.PlayOneShot(
buttonsound.clip);

Coelogyne_celebensis.Sto
p();
        Info2.enabled = true;
        Ket2.enabled = false;
    }
    public void ket_3()
    {

Phalaenopsis_amabilis.Pla
```

```
yOneShot(Phalaenopsis_amabilis.clip);
        Info3.enabled = false;
        Ket3.enabled = true;
    }
    public void info_3()
    {

buttonsound.PlayOneShot(buttonsound.clip);

Phalaenopsis_amabilis.Stop();
        Info3.enabled = true;
        Ket3.enabled = false;
    }
    public void ket_4()
    {

Ascocentrum_miniatum.PlayOneShot(Ascocentrum_miniatum.clip);
        Info4.enabled = false;
        Ket4.enabled = true;
    }
    public void info_4()
    {

buttonsound.PlayOneShot(buttonsound.clip);

Ascocentrum_miniatum.Stop();
        Info4.enabled = true;
        Ket4.enabled = false;
    }
    public void ket_5()
    {

Calanthe_triplicata.PlayOneShot(Calanthe_triplicata.clip);
        Info5.enabled = false;
        Ket5.enabled = true;
    }
    public void info_5()
    {

buttonsound.PlayOneShot(buttonsound.clip);

Calanthe_triplicata.Stop();
        Info5.enabled = true;
        Ket5.enabled = false;
    }
}
```

MiniMap.Cs

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

public class MiniMap : MonoBehaviour
{
    public Transform Player;

    private void LateUpdate()
    {
        Vector3 newPos = transform.position;
        newPos.y = transform.position.y;
        transform.position = newPos;


        transform.rotation = Quaternion.Euler(90, Player.eulerAngles.y, 0);
    }
}
```

PintuOtomatis.Cs

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;



public class PintuOtomatis : MonoBehaviour

{

    public Animator doorAnimator; // Referensi ke komponen Animator pada pintu

    void Start()

    {

        if (doorAnimator == null)

        {


Debug.LogError("Animator is not assigned!");

        }

    }

    void OnTriggerEnter(Collider other)

    {

        if (other.CompareTag("Player"))

        {

            Debug.Log("Player entered the trigger zone");


doorAnimator.SetBool("isOpen", true);

        }

    }



    void OnTriggerExit(Collider other)

    {

        if (other.CompareTag("Player"))

        {
```

```
        Debug.Log("Player exited the trigger zone");


doorAnimator.SetBool("isOpen", false);

        }

    }

}
```

GvrReticlePointer.Cs

```
//-----------------------------------------------------------------------
// <copyright file="GvrReticlePointer.cs" company="Google Inc.">
// Copyright 2017 Google Inc. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
// http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
// </copyright>
//-----------------------------------------------------------------------

using UnityEngine;
using UnityEngine.EventSystems;

/// <summary>Draws a circular reticle in front of any object that the user points at.</summary>
/// <remarks>The circle dilates if the object is clickable.</remarks>
[HelpURL("https://developers.google.com/vr/unity/reference/class/GvrReticlePointer")]
public class GvrReticlePointer : GvrBasePointer
{
  /// <summary>
  /// The constants below are expsed for testing. Minimum inner angle of the reticle (in degrees).
  /// </summary>
  public const float RETICLE_MIN_INNER_ANGLE = 0.0f;

  /// <summary>Minimum outer angle of the reticle (in degrees).</summary>
  public const float RETICLE_MIN_OUTER_ANGLE = 0.5f;

  /// <summary>
  /// Angle at which to expand the reticle when intersecting with an object (in degrees).
  /// </summary>
  public const float RETICLE_GROWTH_ANGLE = 1.5f;

  /// <summary>Minimum distance of the reticle (in meters).</summary>
  public const float RETICLE_DISTANCE_MIN = 0.45f;

  /// <summary>Maximum distance of the reticle (in meters).</summary>
  public float maxReticleDistance = 20.0f;

  /// <summary>Number of segments making the reticle circle.</summary>
  public int reticleSegments = 20;

  /// <summary>Growth speed multiplier for the reticle.</summary>
  public float reticleGrowthSpeed = 8.0f;

  /// <summary>Sorting order to use for the reticle's renderer.</summary>
  /// <remarks><para>
  /// Range values come from https://docs.unity3d.com/ScriptReference/Renderer-sortingOrder.html.
  /// </para><para>
  /// Default value 32767 ensures gaze reticle is always rendered on top.
  /// </para></remarks>
  [Range(-32767, 32767)]
  public int reticleSortingOrder = 32767;

  /// <summary>Gets or sets the material used to render the reticle.</summary>
  /// <value>The material used to render the reticle.</value>
  public Material MaterialComp { private get; set; }

  /// <summary>Gets the current inner angle of the reticle (in degrees).</summary>
  /// <remarks>Exposed for testing.</remarks>
  /// <value>The current
```

```
inner angle of the reticle (in degrees).</value>
    public float ReticleInnerAngle { get; private set; }

    /// <summary>Gets the current outer angle of the reticle (in degrees).</summary>
    /// <remarks>Exposed for testing.</remarks>
    /// <value>The current outer angle of the reticle (in degrees).</value>
    public float ReticleOuterAngle { get; private set; }

    /// <summary>Gets the current distance of the reticle (in meters).</summary>
    /// <remarks>Getter exposed for testing.</remarks>
    /// <value>The current distance of the reticle (in meters).</value>
    public float ReticleDistanceInMeters { get; private set; }

    /// <summary>
    /// Gets the current inner and outer diameters of the reticle, before distance multiplication.
    /// </summary>
    /// <remarks>Getters exposed for testing.</remarks>
    /// <value>
    /// The current inner and outer diameters of the reticle, before distance multiplication.
    /// </value>
    public float ReticleInnerDiameter { get; private set; }

    /// <summary>Gets the current outer diameter of the reticle (in meters).</summary>
    /// <value>The current outer diameter of the reticle (in meters).</value>
    public float ReticleOuterDiameter { get; private set; }

    /// <inheritdoc/>
    public override float MaxPointerDistance
    {
        get { return maxReticleDistance; }
    }

    /// <inheritdoc/>
    public override void OnPointerEnter(RaycastResult raycastResultResult, bool isInteractive)
    {

SetPointerTarget(raycastResultResult.worldPosition, isInteractive);
    }

    /// <inheritdoc/>
    public override void OnPointerHover(RaycastResult raycastResultResult, bool isInteractive)
    {

SetPointerTarget(raycastResultResult.worldPosition, isInteractive);
    }

    /// <inheritdoc/>
    public override void OnPointerExit(GameObject previousObject)
    {

ReticleDistanceInMeters = maxReticleDistance;
        ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
    }

    /// <inheritdoc/>
    public override void OnPointerClickDown()
    {
    }

    /// <inheritdoc/>
    public override void OnPointerClickUp()
    {
    }

    /// <inheritdoc/>
    public override void GetPointerRadius(out float enterRadius, out float exitRadius)
    {
        float min_inner_angle_radians = Mathf.Deg2Rad * RETICLE_MIN_INNER_ANGLE;

        float max_inner_angle_radians =
            Mathf.Deg2Rad * (RETICLE_MIN_INNER_ANGLE + RETICLE_GROWTH_ANGLE);

        enterRadius = 2.0f * Mathf.Tan(min_inner_angle_radians);
        exitRadius = 2.0f * Mathf.Tan(max_inner_angle_radians);
    }

    /// <summary>Updates the material based on the reticle properties.</summary>
    public void UpdateDiameters()
    {

ReticleDistanceInMeters =

Mathf.Clamp(ReticleDistanceInMeters, RETICLE_DISTANCE_MIN, maxReticleDistance);

        if (ReticleInnerAngle
```

```
< RETICLE_MIN_INNER_ANGLE)
        {
            ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        }

        if (ReticleOuterAngle < RETICLE_MIN_OUTER_ANGLE)
        {
            ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
        }

        float inner_half_angle_radians = Mathf.Deg2Rad * ReticleInnerAngle * 0.5f;
        float outer_half_angle_radians = Mathf.Deg2Rad * ReticleOuterAngle * 0.5f;

        float inner_diameter = 2.0f * Mathf.Tan(inner_half_angle_radians);
        float outer_diameter = 2.0f * Mathf.Tan(outer_half_angle_radians);

        ReticleInnerDiameter = 

Mathf.Lerp(ReticleInnerDiameter, inner_diameter, Time.unscaledDeltaTime * reticleGrowthSpeed);
        ReticleOuterDiameter = 

Mathf.Lerp(ReticleOuterDiameter, outer_diameter, Time.unscaledDeltaTime * reticleGrowthSpeed);


MaterialComp.SetFloat("_InnerDiameter", ReticleInnerDiameter * ReticleDistanceInMeters);

MaterialComp.SetFloat("_OuterDiameter", ReticleOuterDiameter * ReticleDistanceInMeters);

MaterialComp.SetFloat("_DistanceInMeters", ReticleDistanceInMeters);
    }

    /// @cond
    /// <inheritdoc/>
    protected override void Start()
    {
        base.Start();

        Renderer rendererComponent = GetComponent<Renderer>();

rendererComponent.sortingOrder = reticleSortingOrder;

        MaterialComp = rendererComponent.material;


CreateReticleVertices();
    }

    /// @endcond
    /// <summary>This MonoBehavior's Awake behavior.</summary>
    private void Awake()
    {
        ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE;
        ReticleOuterAngle = RETICLE_MIN_OUTER_ANGLE;
    }

    /// @cond
    /// <summary>This MonoBehavior's `Update` method.</summary>
    private void Update()
    {
        UpdateDiameters();
    }

    /// @endcond
    /// <summary>Sets the reticle pointer's target.</summary>
    /// <param name="target">The target location.</param>
    /// <param name="interactive">Whether the pointer is pointing at an interactive object.</param>
    /// <returns>Returns `true` if the target is set successfully.</returns>
    private bool SetPointerTarget(Vector3 target, bool interactive)
    {
        if (PointerTransform == null)
        {

Debug.LogWarning("Cannot operate on a null pointer transform");
            return false;
        }

        Vector3 targetLocalPosition = PointerTransform.InverseTransformPoint(target);


ReticleDistanceInMeters = Mathf.Clamp(targetLocalPosition.z,

RETICLE_DISTANCE_MIN,

maxReticleDistance);
        if (interactive)
        {
            ReticleInnerAngle = RETICLE_MIN_INNER_ANGLE + RETICLE_GROWTH_ANGLE;
```

```
            ReticleOuterAngle
=
RETICLE_MIN_OUTER_
ANGLE +
RETICLE_GROWTH_AN
GLE;
        }
        else
        {
            ReticleInnerAngle
=
RETICLE_MIN_INNER_
ANGLE;
            ReticleOuterAngle
=
RETICLE_MIN_OUTER_
ANGLE;
        }

        return true;
    }

    private void
CreateReticleVertices()
    {
        Mesh mesh = new
Mesh();

gameObject.AddCompone
nt<MeshFilter>();

GetComponent<MeshFilte
r>().mesh = mesh;

        int segments_count =
reticleSegments;
        int vertex_count =
(segments_count + 1) * 2;

#region Vertices

        Vector3[] vertices =
new
Vector3[vertex_count];

        const float kTwoPi =
Mathf.PI * 2.0f;
        int vi = 0;
        for (int si = 0; si <=
segments_count; ++si)
        {
            // Add two vertices
for every circle segment:
one at the beginning of the
            // prism, and one at
the end of the prism.
            float angle =
                (float)si /
                (float)segments_co
                unt * kTwoPi;

            float x =
Mathf.Sin(angle);
            float y =
Mathf.Cos(angle);

            vertices[vi++] =
new Vector3(x, y, 0.0f); //
Outer vertex.
            vertices[vi++] =
new Vector3(x, y, 1.0f); //
Inner vertex.
        }
#endregion

#region Triangles
        int indices_count =
(segments_count + 1) * 3 *
2;
        int[] indices = new
int[indices_count];

        int vert = 0;
        int idx = 0;
        for (int si = 0; si <
segments_count; ++si)
        {
            indices[idx++] =
vert + 1;
            indices[idx++] =
vert;
            indices[idx++] =
vert + 2;

            indices[idx++] =
vert + 1;
            indices[idx++] =
vert + 2;
            indices[idx++] =
vert + 3;

            vert += 2;
        }
#endregion

        mesh.vertices =
vertices;
        mesh.triangles =
indices;

mesh.RecalculateBounds()
;
#if
!UNITY_5_5_OR_NEWE
R
        // Optimize() is
deprecated as of Unity
5.5.0p1.
        mesh.Optimize();
#endif  //
!UNITY_5_5_OR_NEWE
R
    }
}
```

SuaraGanti.Cs

```
using System.Collections;
using UnityEngine;

public class SuaraGanti :
MonoBehaviour
{
    public AudioSource
audioSource1; //
AudioSource untuk audio
pertama
    public AudioSource
audioSource2; //
AudioSource untuk audio
kedua

    public AudioClip
audioClip1; // Audio clip
pertama
    public AudioClip
audioClip2; // Audio clip
kedua

    void Start()
    {
        // Pastikan audio
sources dan audio clips
sudah diassign
        if (audioSource1 !=
null && audioSource2 !=
null && audioClip1 !=
null && audioClip2 !=
null)
        {

StartCoroutine(PlaySequen
tialAudio());
        }
        else
        {

Debug.LogError("AudioSo
urce atau AudioClip tidak
```

```
diassign dengan benar
pada " +
gameObject.name);
        }
    }

    private IEnumerator
PlaySequentialAudio()
    {
        // Set clip dan play
audio pertama
        audioSource1.clip =
audioClip1;
        audioSource1.Play();

        // Tunggu hingga
audio pertama selesai
        yield return new
WaitForSeconds(audioClip
1.length);

        // Set clip dan play
audio kedua
        audioSource2.clip =
audioClip2;
        audioSource2.Play();
    }
}
```

GazeInteraksi.Cs

```
using System.Collections;
using
System.Collections.Generi
c;
using UnityEngine;
using
UnityEngine.EventSystem
s;

public class GazeInteraksi
: MonoBehaviour
{
    public float waktulihat =
2f;
    private float waktu;
    private bool lihatobjek;

    // Use this for
initialization
    void Start()
    {

    }

    // Update is called once
per frame
    void Update()
    {

        if (lihatobjek)
        {
            waktu +=
Time.deltaTime;

            if (waktu >=
waktulihat)
            {
                // execute
pointerdown handler

ExecuteEvents.Execute(ga
meObject, new
PointerEventData(EventSy
stem.current),
ExecuteEvents.pointerDow
nHandler);
                waktu = 0f;
            }
        }

    }

    public void
PointerEnter()
    {
        lihatobjek = true;

Debug.Log("PointerEnter"
);
    }

    public void
PointerExit()
    {
        lihatobjek = false;

Debug.Log("PointerExit");
    }

    public void
PointerDown()
    {

Debug.Log("PointerDown
");
    }
}
```

GvrHeadset.Cs

```
//-----------------------------
-----------------------------
--------
// <copyright
file="GvrHeadset.cs"
company="Google Inc.">
// Copyright 2017 Google
Inc. All rights reserved.
//
// Licensed under the
Apache License, Version
2.0 (the "License");
// you may not use this file
except in compliance with
the License.
// You may obtain a copy
of the License at
//
//
http://www.apache.org/lice
nses/LICENSE-2.0
//
// Unless required by
applicable law or agreed to
in writing, software
// distributed under the
License is distributed on
an "AS IS" BASIS,
// WITHOUT
WARRANTIES OR
CONDITIONS OF ANY
KIND, either express or
implied.
// See the License for the
specific language
governing permissions and
// limitations under the
License.
// </copyright>
//-----------------------------
-----------------------------
--------

using System;
using System.Collections;
using
System.ComponentModel;
using Gvr.Internal;
using UnityEngine;

/// <summary>Main entry
point for Standalone
headset APIs.</summary>
/// <remarks>
/// To use this API, use the
GvrHeadset prefab. There
```

```
can be only one such
prefab in a scene, since
/// this is a singleton
object.
/// </remarks>
[HelpURL("https://develop
ers.google.com/vr/unity/ref
erence/class/GvrHeadset")]
public class GvrHeadset :
MonoBehaviour
{
#if UNITY_EDITOR
    /// <summary>Whether
this app supports
Positional Head Tracking
in editor play
mode.</summary>
    /// <remarks>
    /// This is a user-
controlled field which can
be toggled in the inspector
for the GvrHeadset.
    /// Its value is ignored if
there is a connected device
running Instant Preview.
    /// </remarks>
    public static bool
editorSupportsPositionalH
eadTracking = false;
#endif // UNITY_EDITOR

    private static
GvrHeadset instance;

    private
IHeadsetProvider
headsetProvider;
    private HeadsetState
headsetState;
    private IEnumerator
headsetUpdate;
    private
WaitForEndOfFrame
waitForEndOfFrame =
new
WaitForEndOfFrame();

    // Delegates for GVR
events.
    private
OnSafetyRegionEvent
safetyRegionDelegate;
    private
OnRecenterEvent
recenterDelegate;

    /// <summary>Initializes
a new instance of the <see
cref="GvrHeadset" />
class.</summary>
    protected GvrHeadset()
    {

headsetState.Initialize();
    }

    // Delegate definitions.

    /// <summary>
    /// This delegate is
called when the headset
crosses the safety region
boundary.
    /// </summary>
    /// <param
name="enter">
    /// Set to `true` if the
safety region is being
entered, or `false` if the
safety region is
    /// being exited.
    /// </param>
    public delegate void
OnSafetyRegionEvent(boo
l enter);

    /// <summary>This
delegate is called after the
headset is
recentered.</summary>
    /// <param
name="recenterType">Ind
icates the reason
recentering
occurred.</param>
    /// <param
name="recenterFlags">Fla
gs related to recentering.
See
|GvrRecenterFlags|.</para
m>
    /// <param
name="recenteredPosition
">The positional offset
from the session start
pose.</param>
    /// <param
name="recenteredOrientati
on">
    /// The rotational offset
from the session start pose.
    /// </param>
    public delegate void
OnRecenterEvent(GvrRec
enterEventType
recenterType,

GvrRecenterFlags
recenterFlags,

Vector3
recenteredPosition,

Quaternion
recenteredOrientation);

#region
DELEGATE_HANDLER
S
    /// <summary>Event
handlers for
`OnSafetyRegionChange`.
</summary>
    /// <remarks>Triggered
when the safety region has
been entered or
exited.</remarks>
    public static event
OnSafetyRegionEvent
OnSafetyRegionChange
    {
        add
        {
            if (instance != null)
            {

instance.safetyRegionDele
gate += value;
            }
        }

        remove
        {
            if (instance != null)
            {

instance.safetyRegionDele
gate -= value;
            }
        }
    }

    /// <summary>Event
handlers for
`OnRecenter`.</summary>
    /// <remarks>Triggered
when a recenter command
has been issued by the
```

```
user.</remarks>
    public static event
OnRecenterEvent
OnRecenter
    {
        add
        {
            if (instance != null)
            {

instance.recenterDelegate
+= value;
            }
        }

        remove
        {
            if (instance != null)
            {

instance.recenterDelegate -
= value;
            }
        }
    }
#endregion  //
DELEGATE_HANDLER
S

#region
GVR_HEADSET_PROPE
RTIES
    /// <summary>
    /// Gets a value
indicating whether this
headset supports 6DoF
positional tracking.
    /// </summary>
    /// <value>
    /// Value `true` if this
headset supports 6DoF
positional tracking, or
`false` if only 3DoF
    /// rotation-based head
tracking is supported.
    /// </value>
    public static bool
SupportsPositionalTrackin
g
    {
        get
        {
            if (instance ==
null)
            {
                return false;
            }

            try
            {
                return
instance.headsetProvider.S
upportsPositionalTracking;
            }
            catch (Exception e)
            {

Debug.LogError("Error
reading
SupportsPositionalTrackin
g: " + e.Message);
                return false;
            }
        }
    }

    /// <summary>
    /// Gets a value
indicating whether this
headset provides an Editor
Emulator.
    /// </summary>
    /// <value>
    /// Value `true` if this
headset provides an Editor
Emulator, or `false`
otherwise.
    /// </value>
    public bool
ProvidesEditorEmulator
    {
        get
        {
            return
headsetProvider as
EditorHeadsetProvider !=
null;
        }
    }

    /// <summary>Populates
`floorHeight` with the
detected height, if one is
available.</summary>
    /// <remarks>This may
be unavailable if the
underlying GVR API call
fails.</remarks>
    /// <returns>
    /// Returns `true` if value
retrieval was successful,
`false` otherwise (depends
on tracking
    /// state).
    /// </returns>
    /// <param
name="floorHeight">
    /// If this call returns
`true`, this value is set to
the retrieved `floorHeight`.
Otherwise
    /// leaves the value
unchanged.
    /// </param>

[SuppressMemoryAllocati
onError(
        IsWarning = true,
Reason = "A getter for a
float should not allocate.")]
    public static bool
TryGetFloorHeight(ref
float floorHeight)
    {
        if (instance == null)
        {
            return false;
        }

        return
instance.headsetProvider.T
ryGetFloorHeight(ref
floorHeight);
    }

    /// <summary>
    /// Populates position
and rotation with the last
recenter transform, if one
is available.
    /// </summary>
    /// <remarks>This may
be unavailable if the
underlying GVR API call
fails.</remarks>
    /// <returns>Returns
`true` if value retrieval was
successful, `false`
otherwise.</returns>
    /// <param
name="position">
    /// If this call returns
`true`, this value is set to
the retrieved position.
    /// </param>
    /// <param
name="rotation">
    /// If this call returns
```

```
`true`, this value is set to
the retrieved rotation.
    /// </param>
    public static bool
TryGetRecenterTransform
(ref Vector3 position, ref
Quaternion rotation)
    {
        if (instance == null)
        {
            return false;
        }

        return
instance.headsetProvider.T
ryGetRecenterTransform(r
ef position, ref rotation);
    }

    /// <summary>Populates
`safetyType` with the
safety region type, if one is
available.</summary>
    /// <remarks>
    /// Populates
`safetyType` with the
available safety region
feature on the currently-
running
    /// device.  This may be
unavailable if the
underlying GVR API call
fails.
    /// </remarks>
    /// <returns>Returns
`true` if value retrieval was
successful, `false`
otherwise.</returns>
    /// <param
name="safetyType">
    /// If this call returns
`true`, this value is set to
the retrieved `safetyType`.
    /// </param>
    public static bool
TryGetSafetyRegionType(
ref GvrSafetyRegionType
safetyType)
    {
        if (instance == null)
        {
            return false;
        }

        return
instance.headsetProvider.T
ryGetSafetyRegionType(re
f safetyType);
    }

    /// <summary>
    /// Populates
`innerRadius` with the
safety cylinder inner
radius, if one is available.
    /// </summary>
    /// <remarks>
    /// This is the radius at
which safety management
(e.g. safety fog) may cease
taking effect.
    /// <para>
    /// If the safety region is
of type
`GvrSafetyRegionType.Cy
linder`, populates
`innerRadius` with
    /// the inner radius size
of the safety cylinder in
meters.  Before using,
confirm that the
    /// safety region type is
`GvrSafetyRegionType.Cy
linder`.  This may be
unavailable if the
    /// underlying GVR API
call fails.
    /// </para></remarks>
    /// <returns>Returns
`true` if value retrieval was
successful, `false`
otherwise.</returns>
    /// <param
name="innerRadius">
    /// If this call returns
`true`, this value is set to
the retrieved
`innerRadius`.
    /// </param>
    public static bool
TryGetSafetyCylinderInne
rRadius(ref float
innerRadius)
    {
        if (instance == null)
        {
            return false;
        }

        return
instance.headsetProvider.T
ryGetSafetyCylinderInner
Radius(ref innerRadius);
    }

    /// <summary>
    /// Populates
`outerRadius` with the
safety cylinder outer
radius, if one is available.
    /// </summary>
    /// <remarks>
    /// If the safety region is
of type
`GvrSafetyRegionType.Cy
linder`, populates
`outerRadius` with
    /// the outer radius size
of the safety cylinder in
meters.  Before using,
confirm that the
    /// safety region type is
`GvrSafetyRegionType.Cy
linder`.  This may be
unavailable if the
    /// underlying GVR API
call fails.
    /// <para>
    /// This is the radius at
which safety management
(e.g. safety fog) may start
to take effect.
    /// </para></remarks>
    /// <returns>Returns
`true` if value retrieval was
successful, `false`
otherwise.</returns>
    /// <param
name="outerRadius">
    /// If this call returns
`true`, this value is set to
the retrieved
`outerRadius`.
    /// </param>
    public static bool
TryGetSafetyCylinderOute
rRadius(ref float
outerRadius)
    {
        if (instance == null)
        {
            return false;
        }

        return
instance.headsetProvider.T
ryGetSafetyCylinderOuter
Radius(ref outerRadius);
```

```
  }

#endregion  // GVR_HEADSET_PROPERTIES
  private void Awake()
  {
    if (instance != null)
    {
      Debug.LogError("More than one GvrHeadset instance was found in your scene. "
          + "Ensure that there is only one GvrHeadset.");
      this.enabled = false;
      return;
    }

    instance = this;
    if (headsetProvider == null)
    {
      headsetProvider = HeadsetProviderFactory.CreateProvider();
    }
  }

  private void OnEnable()
  {
    if (!SupportsPositionalTracking)
    {
      return;
    }

    headsetUpdate = EndOfFrame();
    StartCoroutine(headsetUpdate);
  }

  private void OnDisable()
  {
    if (!SupportsPositionalTracking)
    {
      return;
    }

    if (headsetUpdate != null)
    {
      StopCoroutine(headsetUpdate);
    }
  }

  private void OnDestroy()
  {
    if (!SupportsPositionalTracking)
    {
      return;
    }

    instance = null;
  }

  private void UpdateStandalone()
  {
    // Events are stored in a queue, so poll until we get Invalid.
    headsetProvider.PollEventState(ref headsetState);
    while (headsetState.eventType != GvrEventType.Invalid)
    {
      switch (headsetState.eventType)
      {
        case GvrEventType.Recenter:
          if (recenterDelegate != null)
          {
            recenterDelegate(headsetState.recenterEventType,
                (GvrRecenterFlags)headsetState.recenterEventFlags,
                headsetState.recenteredPosition,
                headsetState.recenteredRotation);
          }

          break;
        case GvrEventType.SafetyRegionEnter:
          if (safetyRegionDelegate != null)
          {
            safetyRegionDelegate(true);
          }

          break;
        case GvrEventType.SafetyRegionExit:
          if (safetyRegionDelegate != null)
          {
            safetyRegionDelegate(false);
          }

          break;
        case GvrEventType.Invalid:
          throw new InvalidEnumArgumentException(
              "Invalid headset event: " + headsetState.eventType);
        default:  // Fallthrough, should never get here.
          break;
      }

      headsetProvider.PollEventState(ref headsetState);
    }
  }

  private IEnumerator EndOfFrame()
  {
    while (true)
    {
      // This must be
```

```
done at the end of the
frame to ensure that all
GameObjects had a chance
            // to read transient
state (e.g. events, etc) for
the current frame before it
gets
            // reset.
            yield return
waitForEndOfFrame;

UpdateStandalone();
        }
    }
}
```

GvrEditorEmulator.Cs

```
//-----------------------------
-----------------------------
--------
// <copyright
file="GvrEditorEmulator.c
s" company="Google
Inc.">
// Copyright 2017 Google
Inc. All rights reserved.
//
// Licensed under the
Apache License, Version
2.0 (the "License");
// you may not use this file
except in compliance with
the License.
// You may obtain a copy
of the License at
//
//
http://www.apache.org/lice
nses/LICENSE-2.0
//
// Unless required by
applicable law or agreed to
in writing, software
// distributed under the
License is distributed on
an "AS IS" BASIS,
// WITHOUT
WARRANTIES OR
CONDITIONS OF ANY
KIND, either express or
implied.
// See the License for the
specific language
governing permissions and
// limitations under the
License.
// </copyright>
//-----------------------------
-----------------------------
--------

using System;
using
System.Collections.Generi
c;
using Gvr.Internal;
using UnityEngine;

/// <summary>Provides
mouse-controlled head
tracking emulation in the
Unity editor.</summary>
[HelpURL("https://develop
ers.google.com/vr/unity/ref
erence/class/GvrEditorEm
ulator")]
public class
GvrEditorEmulator :
MonoBehaviour
{
    // GvrEditorEmulator
should only be compiled in
the Editor.
    //
    // Otherwise, it will
override the camera pose
every frame on device
which causes the
    // following behaviour:
    //
    // The rendered camera
pose will still be correct
because the
VR.InputTracking pose
    // gets applied after
LateUpdate has occured.
However, any functionality
that
    // queries the camera
pose during Update or
LateUpdate after
GvrEditorEmulator has
been
    // updated will get the
wrong value applied by
GvrEditorEmulator
intsead.
#if UNITY_EDITOR
    private const string
AXIS_MOUSE_X =
"Mouse X";
    private const string
AXIS_MOUSE_Y =
"Mouse Y";

    // Simulated neck
model.  Vector from the
neck pivot point to the
point between the eyes.
    private static readonly
Vector3 NECK_OFFSET
= new Vector3(0, 0.075f,
0.08f);

    private static
GvrEditorEmulator
instance;
    private static bool
instanceSearchedFor =
false;

    // Allocate an initial
capacity; this will be
resized if needed.
    private static Camera[]
allCameras = new
Camera[32];

    // Use mouse to emulate
head in the editor.
    // These variables must
be static so that head pose
is maintained between
scene changes,
    // as it is on device.
    private float mouseX =
0;
    private float mouseY =
0;
    private float mouseZ =
0;

    /// <summary>Gets the
instance for this singleton
class.</summary>
    /// <value>The instance
for this singleton
class.</value>
    public static
GvrEditorEmulator
Instance
    {
        get
        {
            if (instance == null
&& !instanceSearchedFor)
```

```
        {
            instance =
FindObjectOfType<GvrEd
itorEmulator>();

instanceSearchedFor =
true;
        }

        return instance;
      }
    }

    /// <summary>Gets the
emulated head
position.</summary>
    /// <value>The emulated
head position.</value>
    public Vector3
HeadPosition { get; private
set; }

    /// <summary>Gets the
emulated head
rotation.</summary>
    /// <value>The emulated
head rotation.</value>
    public Quaternion
HeadRotation { get;
private set; }

    /// <summary>Recenters
the emulated
headset.</summary>
    public void Recenter()
    {
      mouseX = mouseZ =
0;  // Do not reset pitch,
which is how it works on
the phone.

UpdateHeadPositionAndR
otation();

ApplyHeadOrientationTo
VRCameras();
    }

    /// <summary>Single-
frame updates for this
module.</summary>
    /// <remarks>Should be
called in one
MonoBehavior's `Update`
method.</remarks>
    public void
UpdateEditorEmulation()
    {
      if
(InstantPreview.IsActive)
      {
        return;
      }

      if
(GvrControllerInput.Recen
tered)
      {
        Recenter();
      }

      bool rolled = false;
      if
(CanChangeYawPitch())
      {

GvrCursorHelper.HeadEm
ulationActive = true;
        mouseX +=
Input.GetAxis(AXIS_MO
USE_X) * 5;
        if (mouseX <= -
180)
        {
          mouseX += 360;
        }
        else if (mouseX >
180)
        {
          mouseX -= 360;
        }

        mouseY -=
Input.GetAxis(AXIS_MO
USE_Y) * 2.4f;
        mouseY =
Mathf.Clamp(mouseY, -
85, 85);
      }
      else if
(CanChangeRoll())
      {

GvrCursorHelper.HeadEm
ulationActive = true;
        rolled = true;
        mouseZ +=
Input.GetAxis(AXIS_MO
USE_X) * 5;
        mouseZ =
Mathf.Clamp(mouseZ, -
85, 85);
      }
      else
      {

GvrCursorHelper.HeadEm
ulationActive = false;
      }

      if (!rolled)
      {
        // People don't
usually leave their heads
tilted to one side for long.
        mouseZ =
Mathf.Lerp(mouseZ, 0,
Time.deltaTime /
(Time.deltaTime + 0.1f));
      }


UpdateHeadPositionAndR
otation();

ApplyHeadOrientationTo
VRCameras();
    }

    private void Awake()
    {
      if (Instance == null)
      {
        instance = this;
      }
      else if (Instance !=
this)
      {

Debug.LogError("More
than one active
GvrEditorEmulator
instance was found in your
" +
                "scene.
Ensure that there is only
one active
GvrEditorEmulator.");
        this.enabled =
false;
        return;
      }
    }

    private void Start()
    {
      UpdateAllCameras();
      for (int i = 0; i <
```

```
Camera.allCamerasCount; ++i)
      {
        Camera cam = allCameras[i];

        // Only check camera if it is an enabled VR Camera.
        if (cam && cam.enabled && cam.stereoTargetEye != StereoTargetEyeMask.None)
        {
          if (cam.nearClipPlane > 0.1 &&
GvrSettings.ViewerPlatform == GvrSettings.ViewerPlatformType.Daydream)
            {

Debug.LogWarningFormat(
                "Camera \"{0}\" has Near clipping plane set to {1} meters, which might " +
                "cause the rendering of the Daydream controller to clip unexpectedly.\n" +
                "Suggest using a lower value, 0.1 meters or less.",
                cam.name, cam.nearClipPlane);
            }
        }
      }
    }

    private void Update()
    {
      // GvrControllerInput automatically updates GvrEditorEmulator.
      // This guarantees that GvrEditorEmulator is updated before anything else responds to
      // controller input, which ensures that re-centering works correctly in the editor.
      // If GvrControllerInput is not available, then fallback to using Update().
      if (GvrControllerInput.ApiStatus != GvrControllerApiStatus.Error)
      {
        return;
      }

UpdateEditorEmulation();
    }

    private bool CanChangeYawPitch()
    {
      // If the MouseControllerProvider is currently active, then don't move the camera.
      if (MouseControllerProvider.IsActivateButtonPressed)
      {
        return false;
      }

      return Input.GetKey(KeyCode.LeftAlt) || Input.GetKey(KeyCode.RightAlt);
    }

    private bool CanChangeRoll()
    {
      // If the MouseControllerProvider is currently active, then don't move the camera.
      if (MouseControllerProvider.IsActivateButtonPressed)
      {
        return false;
      }

      return Input.GetKey(KeyCode.LeftControl) || Input.GetKey(KeyCode.RightControl);
    }

    private void UpdateHeadPositionAndRotation()
    {
      HeadRotation = Quaternion.Euler(mouseY, mouseX, mouseZ);
      HeadPosition = (HeadRotation * NECK_OFFSET) - (NECK_OFFSET.y * Vector3.up);
    }

    private void ApplyHeadOrientationToVRCameras()
    {
      UpdateAllCameras();

      // Update all VR cameras using Head position and rotation information.
      for (int i = 0; i < Camera.allCamerasCount; ++i)
      {
        Camera cam = allCameras[i];

        // Check if the Camera is a valid VR Camera, and if so update it to track head motion.
        if (cam && cam.enabled && cam.stereoTargetEye != StereoTargetEyeMask.None)
        {

cam.transform.localPosition = HeadPosition * cam.transform.lossyScale.y;

cam.transform.localRotation = HeadRotation;
        }
      }
    }
```

```
    // Avoids per-frame
allocations. Allocates only
when allCameras array is
resized.
    private void
UpdateAllCameras()
    {
        // Get all Cameras in
the scene using persistent
data structures.
        if
(Camera.allCamerasCount
> allCameras.Length)
        {
            int
newAllCamerasSize =
Camera.allCamerasCount;
            while
(Camera.allCamerasCount
> newAllCamerasSize)
            {

newAllCamerasSize *= 2;
            }

            allCameras = new
Camera[newAllCamerasSi
ze];
        }

        // The GetAllCameras
method doesn't allocate
memory
(Camera.allCameras does).

Camera.GetAllCameras(all
Cameras);
    }

#endif  //
UNITY_EDITOR
}
```

InstantPreview.Cs

```
//-----------------------------
-----------------------------
--------
// <copyright
file="InstantPreview.cs"
company="Google Inc.">
// Copyright 2017 Google
Inc. All rights reserved.
//
// Licensed under the
Apache License, Version
2.0 (the "License");
// you may not use this file
except in compliance with
the License.
// You may obtain a copy
of the License at
//
//
http://www.apache.org/lice
nses/LICENSE-2.0
//
// Unless required by
applicable law or agreed to
in writing, software
// distributed under the
License is distributed on
an "AS IS" BASIS,
// WITHOUT
WARRANTIES OR
CONDITIONS OF ANY
KIND, either express or
implied.
// See the License for the
specific language
governing permissions and
// limitations under the
License.
// </copyright>
//-----------------------------
-----------------------------
--------

using UnityEngine;
using
System.Runtime.InteropSe
rvices;
using System;
using System.Text;
using
System.Collections.Generi
c;
using System.IO;
using System.Threading;
using
System.Text.RegularExpre
ssions;

namespace Gvr.Internal
{
    /// <summary>A class
module for handling
Instant
Preview.</summary>
    /// <remarks><para>
    /// Handles connecting
to the Instant Preview
Unity plugin.
    /// </para><para>
    /// Serves as an interface
for retrieving many
headset-oriented fields.
    /// </para><para>
    /// Streams video data to
the Instant Preview Unity
plugin.
    /// </para></remarks>

[HelpURL("https://develop
ers.google.com/vr/unity/ref
erence/class/InstantPrevie
w")]
    public class
InstantPreview :
MonoBehaviour
    {
        /// <summary>
        /// Gets whether
Instant Preview is
currently connected to and
running on a remote
device.
        /// </summary>
        public static bool
IsActive
        {
            get
            {
#if UNITY_EDITOR
                return
Gvr.Internal.InstantPrevie
w.Instance != null
                    &&
Gvr.Internal.InstantPrevie
w.Instance.enabled
                    &&
Gvr.Internal.InstantPrevie
w.Instance.IsCurrentlyCon
nected;
#else
                return false;
#endif // UNITY_EDITOR
            }
        }

        private const string
NoDevicesFoundAdbResu
lt = "error: no
devices/emulators found";

        /// <summary>Gets or
```

```
sets this singleton's
instance.</summary>
        internal static
InstantPreview Instance {
get; set; }

        /// <summary>The
.dll filename of the Instant
Preview Unity
plugin.</summary>
        internal const string
dllName =
"instant_preview_unity_pl
ugin";

        /// <summary>Video
resolutions for streaming
to the connected
device.</summary>
        public enum
Resolutions : int
        {
            /// A high-
resolution image.
            Big,

            /// A regular-
resolution image.
            Regular,

            /// A window-sized
image.
            WindowSized,
        }

        struct ResolutionSize
        {
            public int width;
            public int height;
        }

        ///
<summary>Resolution of
video stream.</summary>
        /// <remarks>Higher
= more expensive / better
visual quality.</remarks>
        [Tooltip("Resolution
of video stream. Higher =
more expensive / better
visual quality.")]
        public Resolutions
OutputResolution =
Resolutions.Big;

        ///
<summary>Options for
anti-aliasing
sampling.</summary>
        public enum
MultisampleCounts
        {
            /// <summary>Take
one sample per
frame.</summary>
            One,

            /// <summary>Take
two samples per
frame.</summary>
            Two,

            /// <summary>Take
four samples per
frame.</summary>
            Four,

            /// <summary>Take
eight samples per
frame.</summary>
            Eight,
        }

        /// <summary>Anti-
aliasing for video
preview.</summary>
        /// <remarks>Higher
= more expensive / better
visual quality.</remarks>
        [Tooltip("Anti-
aliasing for video preview.
Higher = more expensive /
better visual quality.")]
        public
MultisampleCounts
MultisampleCount =
MultisampleCounts.One;

        /// <summary>Bit
rates for video codec
streaming.</summary>
        public enum BitRates
        {
            /// <summary>A bit
rate of 2000kb/s.  The
lowest available bit
rate.</summary>
            _2000,

            /// <summary>A bit
rate of
4000kb/s.</summary>
            _4000,

            /// <summary>A bit
rate of
8000kb/s.</summary>
            _8000,

            /// <summary>A bit
rate of
16000kb/s.</summary>
            _16000,

            /// <summary>A bit
rate of
24000kb/s.</summary>
            _24000,

            /// <summary>A bit
rate of 32000kb/s.  The
highest available bit
rate</summary>
            _32000,
        }

        /// <summary>Video
codec streaming bit
rate.</summary>
        /// <remarks>Higher
= more expensive / better
visual quality.</remarks>
        [Tooltip("Video
codec streaming bit rate.
Higher = more expensive /
better visual quality.")]
        public BitRates
BitRate =
BitRates._16000;

        /// <summary>
        /// If true, installs the
Instant Preview app if it
isn't found on the
connected device.
        /// </summary>
        [Tooltip("Installs the
Instant Preview app if it
isn't found on the
connected device.")]
        public bool
InstallApkOnRun = true;

        /// <summary>An
.apk file containing the
Instant Preview
app.</summary>
        /// <remarks>
```

```
        /// Will be installed on
connected devices which
don't already have it, or
which have an
        /// out-of-date version.
        /// </remarks>
        public
UnityEngine.Object
InstantPreviewApk;

        struct UnityRect
        {
            public float right;
            public float left;
            public float top;
            public float bottom;
        }

        struct UnityEyeViews
        {
            public Matrix4x4
leftEyePose;
            public Matrix4x4
rightEyePose;
            public UnityRect
leftEyeViewSize;
            public UnityRect
rightEyeViewSize;
        }

        /// <summary>A
Unity C#-compliant
wrapper for a
boolean.</summary>
        /// <remarks><para>
        /// This is also defined
on the Instant Preview
plugin in native C++.
        /// </para><para>
        /// If `isValid` is
`false`, `value` should be
ignored.
        /// </para></remarks>
        public struct
UnityBoolAtom
        {

[MarshalAs(UnmanagedT
ype.I1)]
            public bool value;

[MarshalAs(UnmanagedT
ype.I1)]
            public bool isValid;
        }

        /// <summary>A
Unity C#-compliant
wrapper for a
float.</summary>
        /// <remarks><para>
        /// This is also defined
on the Instant Preview
plugin in native C++.
        /// </para><para>
        /// If `isValid` is
`false`, `value` should be
ignored.
        /// </para></remarks>
        public struct
UnityFloatAtom
        {
            public float value;

[MarshalAs(UnmanagedT
ype.I1)]
            public bool isValid;
        }

        /// <summary>A
Unity C#-compliant
wrapper for an
integer.</summary>
        /// <remarks><para>
        /// This is also defined
on the Instant Preview
plugin in native C++.
        /// </para><para>
        /// If `isValid` is
`false`, `value` should be
ignored.
        /// </para></remarks>
        public struct
UnityIntAtom
        {
            public int value;

[MarshalAs(UnmanagedT
ype.I1)]
            public bool isValid;
        }

        /// <summary>A
Unity C#-compliant
wrapper for a 4x4
matrix.</summary>
        /// <remarks><para>
        /// This is also defined
on the Instant Preview
plugin in native C++.
        /// </para><para>
        /// If `isValid` is
`false`, `value` should be
ignored.
        /// </para></remarks>
        public struct
UnityGvrMat4fAtom
        {
            public Matrix4x4
value;

[MarshalAs(UnmanagedT
ype.I1)]
            public bool isValid;
        }

        struct
UnityGlobalGvrProperties
        {
            internal
UnityBoolAtom
supportsPositionalHeadTra
cking;
            internal
UnityBoolAtom
supportsSeeThrough;
            internal
UnityFloatAtom
floorHeight;
            internal
UnityGvrMat4fAtom
recenterTransform;
            internal
UnityIntAtom
safetyRegionType;
            internal
UnityFloatAtom
safetyCylinderEnterRadius
;
            internal
UnityFloatAtom
safetyCylinderExitRadius;
        }

        /// <summary>GVR
Event Types. Associated
with ephemeral (one-
frame-long)
events.</summary>
        public enum
GvrEventType
        {
            /// <summary>A
default value. If this is
seen, something has gone
wrong.</summary>

GVR_EVENT_NONE,
```

```
        ///
<summary>Indicates a
recenter
event.</summary>
        /// <remarks>
        /// This should
always be accompanied by
a`GvrRecenterEventType`
providing additional
        /// details.
        /// </remarks>

GVR_EVENT_RECENTE
R,

        ///
<summary>Indicates that
the safety region has been
exited.</summary>

GVR_EVENT_SAFETY_
REGION_EXIT,

        ///
<summary>Indicates that
the safety region has been
entered.</summary>

GVR_EVENT_SAFETY_
REGION_ENTER,

        ///
<summary>Indicates that
head tracking has
resumed.</summary>

GVR_EVENT_HEAD_TR
ACKING_RESUMED,

        ///
<summary>Indicates that
head tracking has
paused.</summary>

GVR_EVENT_HEAD_TR
ACKING_PAUSED,
    }

    /// <summary>
    /// GVR Recenter
Event Types. Provides
details for
`GvrEventType.GVR_EV
ENT_RECENTER`.
    /// </summary>
    public enum
GvrRecenterEventType
    {
      /// <summary>A
default value. If this is
seen, something has gone
wrong.</summary>

GVR_RECENTER_EVEN
T_NONE,

      ///
<summary>Indicates that
the recenter event occurred
because of a
restart.</summary>

GVR_RECENTER_EVEN
T_RESTART,

      ///
<summary>Indicates that
the recenter event occurred
because of a
realign.</summary>

GVR_RECENTER_EVEN
T_ALIGNED,

      /// <summary>
      /// Indicates that the
recenter event occurred
because the headset was
donned.
      /// </summary>

GVR_RECENTER_EVEN
T_DON,
    }

    /// <summary>
    /// Accompanies a
`GvrEventType.GVR_EV
ENT_RECENTER`.
Provides additional details
about
    /// the recenter event.
    /// </summary>
    internal struct
UnityGvrRecenterEventDa
ta
    {
      /// <summary>The
type of recenter
event.</summary>
      internal
GvrRecenterEventType
recenter_type;

      ///
<summary>Recenter event
flags.</summary>
      internal uint
recenter_event_flags;

      /// <summary>The
offset from the initial start-
space.</summary>
      /// <remarks>
      /// Allows the app
to continue to stream from
the same point of
reference, while keeping
      /// the post-recenter
orientation consistent.
      /// </summary>
      internal Matrix4x4
start_space_from_tracking
_space_transform;
    }

    /// <summary>GVR
event details, received any
time an event
occurs.</summary>
    /// <remarks>This is
also defined on the Instant
Preview plugin in native
C++.</remarks>
    internal struct
UnityGvrEvent
    {
      ///
<summary>Timestamp in
nanoseconds.</summary>
      internal long
timestamp;

      /// <summary>The
event type.</summary>
      internal
GvrEventType type;

      /// <summary>The
event's flags.</summary>
      internal uint flags;

      ///
<summary>Additional
details on recenter
events.</summary>
      /// <remarks>Not
```

```
null if and only if event
type is
`GVR_EVENT_RECENT
ER`.</remarks>
        internal
UnityGvrRecenterEventDa
ta
gvr_recenter_event_data;
    }

    /// <summary>GVR
User
Preferences.</summary>
    /// <remarks>
    /// Associated with
options set by the user in
the Daydream
app.</remarks>
    [System.Serializable]
    public struct
UnityGvrUserPreferences
    {
      /// <summary>The
user's handedness
preference.</summary>
        public
GvrSettings.UserPrefsHan
dedness handedness;
    }

    /// <summary>If
`true`, overrides user
preferences received from
remote device with those
    /// set in the
InstantPreview editor
inspector.</summary>
    [Tooltip("Override
user preferences from
remote device with Editor
preferences.")]
    public bool
overrideDeviceUserPrefs =
false;

    [HideInInspector]
    /// <summary>The
User Preferences to use if
`overrideDeviceUserPrefs`
is `true`.</summary>
    public
UnityGvrUserPreferences
editorUserPrefs;

    /// <summary>The
User Preferences to use if
`overrideDeviceUserPrefs`
is `false`.</summary>
    public
UnityGvrUserPreferences
deviceUserPrefs { get;
private set; }

#if UNITY_EDITOR
    private readonly
string[]
RequiredAndroidFeatures
= {

"feature:android.software.v
r.mode",

"feature:android.hardware.
vr.high_performance",
    };

    static
ResolutionSize[]
resolutionSizes = new
ResolutionSize[]
    {
      new
ResolutionSize()
      {
        //
ResolutionSize.Big
        width = 2560,
height = 1440,
      },
      new
ResolutionSize()
      {
        //
ResolutionSize.Regular
        width = 1920,
height = 1080,
      },

      //
ResolutionSize.WindowSi
zed
      new
ResolutionSize(),
    };

    private static readonly
int[] multisampleCounts =
new int[]
    {
      1,  //
MultisampleCounts.One
      2,  //
MultisampleCounts.Two
      4,  //
MultisampleCounts.Four
      8,  //
MultisampleCounts.Eight
    };

    private static readonly
int[] bitRates = new int[]
    {
      2000,   //
BitRates._2000
      4000,   //
BitRates._4000
      8000,   //
BitRates._8000
      16000,  //
BitRates._16000
      24000,  //
BitRates._24000
      32000,  //
BitRates._32000
    };

    [DllImport(dllName)]
    private static extern
bool IsConnected();

    [DllImport(dllName)]
    private static extern
bool GetHeadPose(out
Matrix4x4 pose, out
double timestamp);

    [DllImport(dllName)]
    private static extern
bool GetEyeViews(out
UnityEyeViews
outputEyeViews);

    [DllImport(dllName)]
    private static extern
bool
GetGlobalGvrProperties(re
f
UnityGlobalGvrProperties
outputProperties);

    [DllImport(dllName)]
    private static extern
bool GetGvrEvent(ref
UnityGvrEvent
outputEvent);

    [DllImport(dllName)]
    private static extern
```

```
bool
GetGvrUserPreferences(re
f UnityGvrUserPreferences
outputUserPrefs);

      [DllImport(dllName)]
      private static extern
IntPtr
GetRenderEventFunc();

      [DllImport(dllName)]
      private static extern
void SendFrame(IntPtr
renderTexture, ref
Matrix4x4 pose, double
timestamp, int bitRate);

      [DllImport(dllName)]
      private static extern
void
GetVersionString(StringB
uilder dest, uint n);

      /// <summary>
      /// Gets whether this
module is currently
connected to a running
Instant Preview app.
      /// </summary>
      /// <value>
      /// Value `true` if this
module is currently
connected to a running
Instant Preview app,
      /// `false` otherwise.
      /// </value>
      public bool
IsCurrentlyConnected
      {
         get { return
connected; }
      }

      private IntPtr
renderEventFunc;
      private
RenderTexture
renderTexture;
      private Matrix4x4
headPose =
Matrix4x4.identity;
      private double
timestamp;

      private class
EyeCamera
      {
         public Camera
leftEyeCamera = null;
         public Camera
rightEyeCamera = null;
      }

      Dictionary<Camera,
EyeCamera> eyeCameras
= new Dictionary<Camera,
EyeCamera>();
      List<Camera>
camerasLastFrame = new
List<Camera>();
      private bool
connected;

      /// <summary>Gets
whether the connected
device supports positional
tracking.</summary>
      public
UnityBoolAtom
supportsPositionalHeadTra
cking { get; private set; }

      /// <summary>Gets
whether the connected
device supports see-
through mode.</summary>
      public
UnityBoolAtom
supportsSeeThrough { get;
private set; }

      /// <summary>Gets
the current height the
headset is off its perceived
floor.</summary>
      /// <remarks>The
`value` is valid only if
`floorHeight.isValid ==
true`.</remarks>
      public
UnityFloatAtom
floorHeight { get; private
set; }

      /// <summary>Gets
the last recenter's offset
transform.</summary>
      /// <remarks>The
`value` is valid only if
`recenterTransform.isValid
== true`.</remarks>
      public
UnityGvrMat4fAtom
recenterTransform { get;
private set; }

      /// <summary>Gets
the type of the safety
region.</summary>
      /// <remarks>The
`value` is valid only if
`safetyRegionType.isValid
== true`.</remarks>
      public UnityIntAtom
safetyRegionType { get;
private set; }

      /// <summary>Gets
the reentry radius of a
cylindrical safety
region.</summary>
      /// <remarks><para>
      /// Entering the safety
cylinder means stepping
close enough to its center
to suppress an
      /// active warning.
      /// </para><para>
      /// The `value` is valid
only if
`safetyCylinderEnterRadiu
s.isValid == true`.
      /// </para></remarks>
      public
UnityFloatAtom
safetyCylinderEnterRadius
{ get; private set; }

      /// <summary>Gets
the exit radius of a
cylindrical safety
region.</summary>
      /// <remarks><para>
      /// Exiting the safety
cylinder means stepping
far enough from its center
to prompt a
      /// warning.
      /// </para><para>
      /// The `value` is valid
only if
`safetyCylinderExitRadius.
isValid == true`.
      /// </para></remarks>
      public
UnityFloatAtom
safetyCylinderExitRadius
{ get; private set; }
```

```
        /// <summary>Gets
the user's handedness
preference.</summary>
        public
GvrSettings.UserPrefsHan
dedness handedness
        {
            get
            {
                if
(overrideDeviceUserPrefs)
                {
                    return
editorUserPrefs.handednes
s;
                }
                else
                {
                    return
deviceUserPrefs.handedne
ss;
                }
            }
        }

        /// <summary>A
queue for active GVR
events.</summary>
        /// <remarks>
        /// This is used
because events only trigger
for one frame, and in some
high-CPU edge cases
        /// the Instant Preview
stream may run at a
different speed than the
Unity player.
        /// Because they are
stored on a queue, the
Unity player can guarantee
it will eventually
        /// trigger them
all.</remarks>
        internal
Queue<UnityGvrEvent>
events = new
Queue<UnityGvrEvent>();

        void Awake()
        {
            renderEventFunc =
GetRenderEventFunc();

            if (Instance != null)
            {
Destroy(gameObject);

gameObject.SetActive(fals
e);
                return;
            }

            Instance = this;

DontDestroyOnLoad(game
Object);
        }

        void Start()
        {
            // Gets local
version name and prints it
out.
            var sb = new
StringBuilder(256);

GetVersionString(sb,
(uint)sb.Capacity);
            var
pluginIPVersionName =
sb.ToString();

            // Tries to install
Instant Preview apk if set
to do so.
            if
(InstallApkOnRun)
            {
                // Early outs if
set to install but the apk
can't be found.
                if
(InstantPreviewApk ==
null)
                {

Debug.LogError("Trying
to install Instant Preview
apk but reference to
InstantPreview.apk is
broken.");
                    return;
                }

                // Gets the apk
path and installs it on a
separate thread.
                var apkPath =
Path.GetFullPath(UnityEdi
tor.AssetDatabase.GetAsse
tPath(InstantPreviewApk))
;
                if
(File.Exists(apkPath))
                {
                    new Thread(
                    () =>
                    {
                        string
output;
                        string
errors;
                        string
deviceIPVersionName =
null;
                        string
unityAPKVersionName =
null;

                        // Gets
version of apk installed on
device (to remove, if
dated).

RunCommand(InstantPrev
iewHelper.adbPath,
                            "shell
dumpsys package
com.google.instantpreview
| grep versionName",
                            out
output, out errors);

                        // Early outs
if no device is connected.
                        if
(string.Compare(errors,
NoDevicesFoundAdbResu
lt) == 0)
                        {
                            return;
                        }

                        if
(!string.IsNullOrEmpty(ou
tput) &&
string.IsNullOrEmpty(erro
rs))
                        {

deviceIPVersionName =
output.Substring(output.In
dexOf('=') + 1);
                        }

                        // Ensures
```

```
connected device is
Daydream-compatible
before continuing.

RunCommand(InstantPrev
iewHelper.adbPath,
                    "shell
pm list features", out
output, out errors);
                foreach
(string feature in
RequiredAndroidFeatures)
                {
                    if
(output.IndexOf(feature)
== -1)
                    {

Debug.Log(

"Instant Preview disabled;
device is not Daydream-
compatible.");
                        return;
                    }
                }

                // Prints
errors and exits on failure.
                if
(!string.IsNullOrEmpty(err
ors))
                {

Debug.LogError(errors);
                    return;
                }


Debug.Log("Instant
Preview Version: " +
pluginIPVersionName);

                // Gets
version of Unity's local
.apk version (to install, if
needed).

RunCommand(InstantPrev
iewHelper.aaptPath,

string.Format("dump
badging {0}", apkPath),
                    out
output, out errors);
                if
(!string.IsNullOrEmpty(ou
tput) &&
string.IsNullOrEmpty(erro
rs))
                {
                    string
unityAPKVersionInfoDum
p = output;

                    // Finds
(versionName='), captures
any alphaNumerics
separated by periods, and
selects them until (').

System.Text.RegularExpre
ssions.Match
unityAPKVersionNameRe
gex = Regex.Match(

unityAPKVersionInfoDum
p,
"versionName=\'([^']*)\'");
                    if
(unityAPKVersionNameR
egex.Groups.Count > 1)
                    {

unityAPKVersionName =
unityAPKVersionNameRe
gex.Groups[1].Value;
                    }
                    else
                    {

Debug.Log(string.Format(
"Failed to extract version
from: {0}",
unityAPKVersionInfoDum
p));
                    }
                }
                else
                {

Debug.Log(string.Format(
"Failed to run: {0} dump
badging {1}",
InstantPreviewHelper.aapt
Path, apkPath));
                }

                //
Determines if Unity plugin
and Unity's local .apk IP
file are the same version,
and exits if not.
                if
(pluginIPVersionName !=
unityAPKVersionName)
                {

Debug.LogWarning(string.
Format(
                    "Unity
Instant Preview plugin
version ({0}) does not
match Unity Instant
Preview .apk version
({1})."
                    + "
This may cause
unpredictable behavior.",

pluginIPVersionName,
unityAPKVersionName));
                }

                //
Determines if app is
installed, and installs it if
not.
                if
(deviceIPVersionName !=
unityAPKVersionName)
                {
                    if
(deviceIPVersionName ==
null)
                    {

Debug.Log(string.Format(

"Instant Preview: app not
found on device,
attempting to install it
from {0}.",

apkPath));
                    }
                    else
                    {

Debug.Log(string.Format(

"Instant Preview: installed
version \"{0}\" does not
match local version
\"{1}\", attempting
upgrade.",

deviceIPVersionName,
```

```
unityAPKVersionName));
                    }


RunCommand(InstantPreviewHelper.adbPath,

string.Format("uninstall com.google.instantpreview", apkPath),

out output, out errors);


RunCommand(InstantPreviewHelper.adbPath,

string.Format("install \"{0}\"", apkPath),

out output, out errors);

                    // Prints any output from trying to install.
                    if (!string.IsNullOrEmpty(output))
                    {

Debug.Log(output);
                    }

                    if (!string.IsNullOrEmpty(errors))
                    {
                        if (string.Equals(errors.Trim(), "Success"))
                        {

Debug.Log("Successfully installed Instant Preview app.");
                        }
                        else
                        {

Debug.LogError(errors);
                        }
                    }
                }


StartInstantPreviewActivity(InstantPreviewHelper.adbPath);
                    }).Start();
                }
            }
            else
            {

Debug.Log("Instant Preview Version: " + pluginIPVersionName);
                new Thread(() =>
                {

StartInstantPreviewActivity(InstantPreviewHelper.adbPath);
                }).Start();
            }
        }

        void UpdateCamera(Camera camera)
        {
            EyeCamera eyeCamera;

            if (!eyeCameras.TryGetValue(camera, out eyeCamera))
            {
                return;
            }

            if (connected)
            {
                if (GetHeadPose(out headPose, out timestamp))
                {

SetEditorEmulatorsEnabled(false);

camera.transform.localRotation =

Quaternion.LookRotation(headPose.GetColumn(2), headPose.GetColumn(1)) *

EditorCameraOriginDict.Get(camera).rotation;

camera.transform.localPosition =

camera.transform.localRotation * headPose.GetRow(3) * -1 +

EditorCameraOriginDict.Get(camera).position;
                }
                else
                {

SetEditorEmulatorsEnabled(true);
                }

                var eyeViews = new UnityEyeViews();
                if (GetEyeViews(out eyeViews))
                {

SetTransformFromMatrix(eyeCamera.leftEyeCamera.gameObject.transform, eyeViews.leftEyePose);

SetTransformFromMatrix(eyeCamera.rightEyeCamera.gameObject.transform, eyeViews.rightEyePose);

                    var near = Camera.main.nearClipPlane;
                    var far = Camera.main.farClipPlane;

eyeCamera.leftEyeCamera.projectionMatrix =

PerspectiveMatrixFromUnityRect(eyeViews.leftEyeViewSize, near, far);

eyeCamera.rightEyeCamera.projectionMatrix =

PerspectiveMatrixFromUnityRect(eyeViews.rightEyeViewSize, near, far);

                    bool
```

```
multisampleChanged =
multisampleCounts[(int)M
ultisampleCount] !=
renderTexture.antiAliasing
;

                // Adjusts
render texture size.
                if
(OutputResolution !=
Resolutions.WindowSized)
                {
                    var
selectedResolutionSize =
resolutionSizes[(int)Output
Resolution];
                    if
(selectedResolutionSize.wi
dth != renderTexture.width
||

selectedResolutionSize.hei
ght !=
renderTexture.height ||

multisampleChanged)
                    {

ResizeRenderTexture(sele
ctedResolutionSize.width,
selectedResolutionSize.hei
ght);
                    }
                }
                else
                {
                    //
OutputResolution ==
Resolutions.WindowSized
                    var
screenAspectRatio =
(float)Screen.width /
Screen.height;

                    var
eyeViewsWidth =
                        -
eyeViews.leftEyeViewSize
.left +

eyeViews.leftEyeViewSize
.right +
                        -
eyeViews.rightEyeViewSi
ze.left +
eyeViews.rightEyeViewSi
ze.right;
                    var
eyeViewsHeight =

eyeViews.leftEyeViewSize
.top +
                        -
eyeViews.leftEyeViewSize
.bottom;
                    if
(eyeViewsHeight > 0f)
                    {
                        int
renderTextureHeight;
                        int
renderTextureWidth;
                        var
eyeViewsAspectRatio =
eyeViewsWidth /
eyeViewsHeight;
                        if
(screenAspectRatio >
eyeViewsAspectRatio)
                        {

renderTextureHeight =
Screen.height;

renderTextureWidth =
(int)(Screen.height *
eyeViewsAspectRatio);
                        }
                        else
                        {

renderTextureWidth =
Screen.width;

renderTextureHeight =
(int)(Screen.width /
eyeViewsAspectRatio);
                        }


renderTextureWidth =
renderTextureWidth &
~0x3;

renderTextureHeight =
renderTextureHeight &
~0x3;

                        if
(multisampleChanged ||
renderTexture.width !=
renderTextureWidth ||

renderTexture.height !=
renderTextureHeight)
                        {

ResizeRenderTexture(rend
erTextureWidth,
renderTextureHeight);
                        }
                    }
                }
            }
        }
        else
        {
            // !connected

SetEditorEmulatorsEnable
d(true);

            if
(renderTexture.width !=
Screen.width ||
renderTexture.height !=
Screen.height)
            {

ResizeRenderTexture(Scre
en.width, Screen.height);
            }
        }
    }

    void
UpdateProperties()
    {

UnityGlobalGvrProperties
unityGlobalGvrProperties
= new
UnityGlobalGvrProperties(
);
        if
(GetGlobalGvrProperties(r
ef
unityGlobalGvrProperties)
)
        {

supportsPositionalHeadTra
cking
            =
unityGlobalGvrProperties.
supportsPositionalHeadTra
```

```
cking;


supportsSeeThrough =
unityGlobalGvrProperties.
supportsSeeThrough;
            floorHeight =
unityGlobalGvrProperties.f
loorHeight;

recenterTransform =
unityGlobalGvrProperties.r
ecenterTransform;

safetyRegionType =
unityGlobalGvrProperties.
safetyRegionType;

safetyCylinderEnterRadius
=
unityGlobalGvrProperties.
safetyCylinderEnterRadius
;

safetyCylinderExitRadius
=
unityGlobalGvrProperties.
safetyCylinderExitRadius;
        }
    }

    void UpdateEvents()
    {
        UnityGvrEvent
unityGvrEvent = new
UnityGvrEvent();
        while
(GetGvrEvent(ref
unityGvrEvent))
        {

events.Enqueue(unityGvrE
vent);
        }
    }

    void
UpdateUserPreferences()
    {

UnityGvrUserPreferences
unityGvrUserPreferences =
new
UnityGvrUserPreferences(
);
        if
(GetGvrUserPreferences(r
ef
unityGvrUserPreferences))
        {
            deviceUserPrefs
=
unityGvrUserPreferences;
        }
    }

    void Update()
    {
        if
(!EnsureCameras())
        {
            return;
        }

        var
newConnectionState =
IsConnected();
        if (connected &&
!newConnectionState)
        {

Debug.Log("Disconnected
from Instant Preview.");
        }
        else if (!connected
&& newConnectionState)
        {

Debug.Log("Connected to
Instant Preview.");
        }

        connected =
newConnectionState;

        foreach
(KeyValuePair<Camera,
EyeCamera> eyeCamera
in eyeCameras)
        {

UpdateCamera(eyeCamera
.Key);
        }

        UpdateProperties();
        UpdateEvents();

UpdateUserPreferences();
    }

    void OnPostRender()
    {
        if (connected &&
renderTexture != null)
        {
            var
nativeTexturePtr =
renderTexture.GetNativeT
exturePtr();

SendFrame(nativeTexture
Ptr, ref headPose,
timestamp,
bitRates[(int)BitRate]);

GL.IssuePluginEvent(rend
erEventFunc, 69);
        }
    }

    void
EnsureCamera(Camera
camera)
    {
        // renderTexture
might still be null so this
creates and assigns it.
        if (renderTexture
== null)
        {
            if
(OutputResolution !=
Resolutions.WindowSized)
            {
                var
selectedResolutionSize =
resolutionSizes[(int)Output
Resolution];

ResizeRenderTexture(sele
ctedResolutionSize.width,
selectedResolutionSize.hei
ght);
            }
            else
            {

ResizeRenderTexture(Scre
en.width, Screen.height);
            }
        }

        EyeCamera
eyeCamera;

        if
(!eyeCameras.TryGetValu
```

```
e(camera, out eyeCamera))
        {
            eyeCamera =
new EyeCamera();

eyeCameras.Add(camera,
eyeCamera);
        }


EnsureEyeCamera(camera,
":Instant Preview Left",
new Rect(0.0f, 0.0f, 0.5f,
1.0f), ref
eyeCamera.leftEyeCamera
);

EnsureEyeCamera(camera,
":Instant Preview Right",
new Rect(0.5f, 0.0f, 0.5f,
1.0f), ref
eyeCamera.rightEyeCamer
a);
    }

    private void
CheckRemoveCameras(Li
st<Camera> cameras)
    {
        // Any cameras that
were here last frame and
not here this frame need
removing from
eyeCameras.
        foreach (Camera
oldCamera in
camerasLastFrame)
        {
            if
(!cameras.Contains(oldCa
mera))
            {
                // Destroys the
eye cameras.
                EyeCamera
curEyeCamera;
                if
(eyeCameras.TryGetValue
(oldCamera, out
curEyeCamera))
                {
                    if
(curEyeCamera.leftEyeCa
mera != null)
                    {
Destroy(curEyeCamera.lef
tEyeCamera.gameObject);
                    }

                    if
(curEyeCamera.rightEyeC
amera != null)
                    {

Destroy(curEyeCamera.rig
htEyeCamera.gameObject)
;
                    }
                }

                // Removes
eye camera entry from
dictionary.

eyeCameras.Remove(oldC
amera);
            }
        }

        camerasLastFrame
= cameras;
    }

    bool EnsureCameras()
    {
        var mainCamera =
Camera.main;
        if (!mainCamera)
        {
            // If the main
camera doesn't exist,
destroys a remaining
render texture and exits.
            if (renderTexture
!= null)
            {

Destroy(renderTexture);
                renderTexture
= null;
            }

            return false;
        }

        // Find all the
cameras and make sure
any non-Instant Preview
cameras have left/right
eyes attached.
        var cameras = new
List<Camera>(ValidCame
ras());

CheckRemoveCameras(ca
meras);

        // Now go and
make sure that all cameras
that are to be driven by
Instant Preview have the
correct setup.
        foreach (Camera
camera in cameras)
        {
            // Skips the
Instant Preview camera,
which is used for a
            // convenience
preview.
            if
(camera.gameObject ==
gameObject)
            {
                continue;
            }


EnsureCamera(camera);
        }

        return true;
    }

    void
EnsureEyeCamera(Camera
mainCamera, String
eyeCameraName, Rect
rect, ref Camera
eyeCamera)
    {
        // Creates eye
camera object if it doesn't
exist.
        if (eyeCamera ==
null)
        {
            var
eyeCameraObject = new
GameObject(mainCamera.
gameObject.name +
eyeCameraName);
            eyeCamera =
eyeCameraObject.AddCo
mponent<Camera>();

eyeCameraObject.transfor
```

```
m.SetParent(mainCamera.
gameObject.transform,
false);
        }


eyeCamera.CopyFrom(mai
nCamera);
        eyeCamera.rect =
rect;

eyeCamera.targetTexture =
renderTexture;

        // Match child
camera's skyboxes to main
camera.
        Skybox
monoCameraSkybox =
mainCamera.gameObject.
GetComponent<Skybox>()
;
        Skybox
customSkybox =
eyeCamera.GetComponent
<Skybox>();
        if
(monoCameraSkybox !=
null)
        {
            if
(customSkybox == null)
            {
                customSkybox
=
eyeCamera.gameObject.A
ddComponent<Skybox>();
            }


customSkybox.material =
monoCameraSkybox.mater
ial;
        }
        else if
(customSkybox != null)
        {

Destroy(customSkybox);
        }
    }

    void
ResizeRenderTexture(int
width, int height)
    {
        var
newRenderTexture = new
RenderTexture(width,
height, 16);

newRenderTexture.antiAli
asing =
multisampleCounts[(int)M
ultisampleCount];
        if (renderTexture
!= null)
        {
            foreach
(KeyValuePair<Camera,
EyeCamera> camera in
eyeCameras)
            {
                if
(camera.Value.leftEyeCam
era != null)
                {

camera.Value.leftEyeCam
era.targetTexture = null;
                }

                if
(camera.Value.rightEyeCa
mera != null)
                {

camera.Value.rightEyeCa
mera.targetTexture = null;
                }
            }


Destroy(renderTexture);
        }

        renderTexture =
newRenderTexture;
    }

    private static void
SetEditorEmulatorsEnable
d(bool enabled)
    {
        foreach (var
editorEmulator in
FindObjectsOfType<GvrE
ditorEmulator>())
        {

editorEmulator.enabled =
enabled;
        }
    }

    private static
Matrix4x4
PerspectiveMatrixFromUn
ityRect(UnityRect rect,
float near, float far)
    {
        if (rect.left ==
rect.right || rect.bottom ==
rect.top || near == far ||
            near <= 0f || far
<= 0f)
        {
            return
Matrix4x4.identity;
        }

        rect.left *= near;
        rect.right *= near;
        rect.top *= near;
        rect.bottom *=
near;
        var X = (2 * near) /
(rect.right - rect.left);
        var Y = (2 * near) /
(rect.top - rect.bottom);
        var A = (rect.right
+ rect.left) / (rect.right -
rect.left);
        var B = (rect.top +
rect.bottom) / (rect.top -
rect.bottom);
        var C = (near + far)
/ (near - far);
        var D = (2 * near *
far) / (near - far);

        var
perspectiveMatrix = new
Matrix4x4();

perspectiveMatrix[0, 0] =
X;

perspectiveMatrix[0, 2] =
A;

perspectiveMatrix[1, 1] =
Y;

perspectiveMatrix[1, 2] =
B;

perspectiveMatrix[2, 2] =
```

```
C;

perspectiveMatrix[2, 3] =
D;

perspectiveMatrix[3, 2] = -
1f;
        return
perspectiveMatrix;
    }

    private static void
SetTransformFromMatrix(
Transform transform,
Matrix4x4 matrix)
    {
        var position =
matrix.GetRow(3);
        position.x *= -1;

transform.localPosition =
position;

transform.localRotation =
Quaternion.LookRotation(
matrix.GetColumn(2),
matrix.GetColumn(1));
    }

    private static void
StartInstantPreviewActivit
y(string adbPath)
    {
        string output;
        string errors;

RunCommand(adbPath,
                "shell am
start -n
com.google.instantpreview
/.InstantPreviewActivity",
                out output,
                out errors);

        // Early outs if no
device is connected.
        if
(string.Compare(errors,
NoDevicesFoundAdbResu
lt) == 0)
        {
            return;
        }
    }

    private static void
RunCommand(string
fileName, string
arguments, out string
output, out string errors)
    {
        using (var process
= new
System.Diagnostics.Proces
s())
        {

System.Diagnostics.Proces
sStartInfo startInfo = new
System.Diagnostics.Proces
sStartInfo(fileName,
arguments);

startInfo.UseShellExecute
= false;

startInfo.RedirectStandard
Error = true;

startInfo.RedirectStandard
Output = true;


startInfo.CreateNoWindow
= true;
            process.StartInfo
= startInfo;

            var
outputBuilder = new
StringBuilder();
            var errorBuilder
= new StringBuilder();

process.OutputDataReceiv
ed += (o, ef) =>
outputBuilder.AppendLine
(ef.Data);

process.ErrorDataReceived
+= (o, ef) =>
errorBuilder.AppendLine(e
f.Data);

            process.Start();

process.BeginOutputRead
Line();

process.BeginErrorReadLi
ne();

process.WaitForExit();
            process.Close();

            // Trims the
output strings to make
comparison easier.
            output =
outputBuilder.ToString().T
rim();
            errors =
errorBuilder.ToString().Tri
m();
        }
    }

    // Gets active, stereo,
non-eye cameras in the
scene.
    private
IEnumerable<Camera>
ValidCameras()
    {
        foreach (var
camera in
Camera.allCameras)
        {
            if
(!camera.enabled ||
camera.stereoTargetEye
==
StereoTargetEyeMask.Non
e)
            {
                continue;
            }

            // Skips camera
if it is determined to be an
eye camera.
            var parent =
camera.transform.parent;
            if (parent != null)
            {
                var
parentCamera =
parent.GetComponent<Ca
mera>();
                if
(parentCamera != null)
                {
                    EyeCamera
parentEyeCamera;
                    if
(eyeCameras.TryGetValue
(parentCamera, out
parentEyeCamera))
```

```
                {
                    if
(camera ==
parentEyeCamera.leftEyeC
amera || camera ==
parentEyeCamera.rightEye
Camera)
                    {

continue;
                    }
                }
            }
        }

        yield return
camera;
    }
}
#else
    /// <summary>
    /// Gets whether this
module is currently
connected to a running
Instant Preview app.
    /// </summary>
    /// <value>
    /// Value `true` if this
module is currently
connected to a running
Instant Preview app,
    /// `false` otherwise.
    /// </value>
    public bool
IsCurrentlyConnected
    {
        get { return false; }
    }
#endif
  }
}
```

GvrControllerInput.Cs

```
//-----------------------------
-------------------------------
--------
// <copyright
file="GvrControllerInput.c
s" company="Google
Inc.">
// Copyright 2017 Google
Inc. All rights reserved.
//
// Licensed under the
Apache License, Version
2.0 (the "License");
// you may not use this file
except in compliance with
the License.
// You may obtain a copy
of the License at
//
//
http://www.apache.org/lice
nses/LICENSE-2.0
//
// Unless required by
applicable law or agreed to
in writing, software
// distributed under the
License is distributed on
an "AS IS" BASIS,
// WITHOUT
WARRANTIES OR
CONDITIONS OF ANY
KIND, either express or
implied.
// See the License for the
specific language
governing permissions and
// limitations under the
License.
// </copyright>
//-----------------------------
-------------------------------
--------

using System;
using System.Collections;
using Gvr.Internal;
using UnityEngine;

/// <summary>Represents
a controller's current
connection
state.</summary>
/// <remarks>
/// All values and
semantics below (except
for Error) are from
gvr_types.h in the GVR C
API.
/// </remarks>
public enum
GvrConnectionState
{
  /// <summary>Indicates
that an error has
occurred.</summary>
  Error = -1,

  /// <summary>Indicates
a controller is
disconnected.</summary>
  Disconnected = 0,

  /// <summary>Indicates
that the device is scanning
for
controllers.</summary>
  Scanning = 1,

  /// <summary>Indicates
that the device is
connecting to a
controller.</summary>
  Connecting = 2,

  /// <summary>Indicates
that the device is
connected to a
controller.</summary>
  Connected = 3,
}

/// <summary>Represents
the status of the controller
API.</summary>
/// <remarks>Values and
semantics are from
`gvr_types.h` in the GVR
C API.</remarks>
public enum
GvrControllerApiStatus
{
  /// <summary>A Unity-
localized error
occurred.</summary>
  /// <remarks>This is the
only value that isn't in
`gvr_types.h`.</remarks>
  Error = -1,

  /// <summary>API is
happy and
healthy.</summary>
  /// <remarks>
  /// This doesn't mean
any controllers are
connected, it just means
that the underlying service
  /// is working properly.
  /// </remarks>
  Ok = 0,

  // Any other status
```

```
represents a permanent failure that requires
    // external action to fix:

    /// <summary>
    /// API failed because this device does not support controllers (API is too low, or other
    /// required feature not present).
    /// </summary>
    Unsupported = 1,

    /// <summary>
    /// This app was not authorized to use the service (e.g., missing permissions, the app is
    /// blacklisted by the underlying service, etc).
    /// </summary>
    NotAuthorized = 2,

    /// <summary>The underlying VR service is not present.</summary>
    Unavailable = 3,

    /// <summary>The underlying VR service is too old, needs upgrade.</summary>
    ApiServiceObsolete = 4,

    /// <summary>
    /// The underlying VR service is too new, is incompatible with current client.
    /// </summary>
    ApiClientObsolete = 5,

    /// <summary>The underlying VR service is malfunctioning. Try again later.</summary>
    ApiMalfunction = 6,
}

/// <summary>Represents a controller's current battery level.</summary>
/// <remarks>Values and semantics from gvr_types.h in the GVR C API.</remarks>
public enum GvrControllerBatteryLevel
{
    /// <summary>A Unity-localized error occurred.</summary>
    /// <remarks>This is the only value that isn't in `gvr_types.h`.</remarks>
    Error = -1,

    /// <summary>The battery state is currently unreported.</summary>
    Unknown = 0,

    /// <summary>Equivalent to 1 out of 5 bars on the battery indicator.</summary>
    CriticalLow = 1,

    /// <summary>Equivalent to 2 out of 5 bars on the battery indicator.</summary>
    Low = 2,

    /// <summary>Equivalent to 3 out of 5 bars on the battery indicator.</summary>
    Medium = 3,

    /// <summary>Equivalent to 4 out of 5 bars on the battery indicator.</summary>
    AlmostFull = 4,

    /// <summary>Equivalent to 5 out of 5 bars on the battery indicator.</summary>
    Full = 5,
}

/// <summary>Represents controller buttons.</summary>
/// <remarks>
/// Values 0-9 are from `gvr_types.h` in the GVR C API.  Value 31 is not represented in the C API.
/// </remarks>
public enum GvrControllerButton
{
    /// <summary>The Button under the touch pad.</summary>
    /// <remarks>Formerly known as Click.</remarks>
    TouchPadButton = 1 << 1,

    /// <summary>Touch pad touching indicator.</summary>
    TouchPadTouch = 1 << 31,

    /// <summary>General application button.</summary>
    App = 1 << 3,

    /// <summary>System button.</summary>
    /// <remarks>Formerly known as Home.</remarks>
    System = 1 << 2,

    /// <summary>Primary button on the underside of the controller.</summary>
    Trigger = 1 << 6,

    /// <summary>Secondary button on the underside of the controller.</summary>
    Grip = 1 << 7,

    /// <summary>Buttons reserved for future use. Subject to name change.</summary>
    Reserved2 = 1 << 8,
}

/// <summary>Represents controller handedness.</summary>
public enum GvrControllerHand
{
    /// <summary>Right
```

```
hand.</summary>
    Right,

    /// <summary>Left
hand.</summary>
    Left,

    /// <summary>Alias for
dominant hand as specified
by
`GvrSettings.Handedness`.
</summary>
    Dominant,

    /// <summary>Alias for
non-dominant
hand.</summary>
    NonDominant,
}

/// <summary>Main entry
point for the Daydream
controller
API.</summary>
/// <remarks>
/// To use this API, add this
script to a game object in
your scene, or use the
/// **GvrControllerMain**
prefab.  This is a singleton
object. There can only be
one object with
/// this script in your scene.
/// <para>
/// To access a controller's
state, get a device from
`GvrControllerInput.GetDe
vice` then query it
/// for state. For example,
to the dominant
controller's current
orientation, use
///
`GvrControllerInput.GetDe
vice(GvrControllerHand.D
ominant).Orientation`.
/// </para></remarks>
[HelpURL("https://develop
ers.google.com/vr/referenc
e/unity/class/GvrController
Input")]
public class
GvrControllerInput :
MonoBehaviour
{
    /// <summary>Indicates
how to connect to the
controller
emulator.</summary>
#if UNITY_EDITOR
    [GvrInfo("Hold Shift to
use the Mouse as the
dominant controller.\n\n" +
            "Controls:  Shift
+\n" +
            "   • Move Mouse
= Change Orientation\n" +
            "   • Left Mouse
Button = ClickButton\n" +
            "   • Right Mouse
Button = AppButton\n" +
            "   • Middle Mouse
Button =
HomeButton/Recenter\n"
+
            "   • Ctrl =
IsTouching\n" +
            "   • Ctrl + Move
Mouse = Change
TouchPos", 8,
UnityEditor.MessageType.
None)]
    [Tooltip("How to
connect to the emulator:
USB cable (recommended)
or WIFI.")]
    [GvrInfo("Controller
Emulator is now
Deprecated", 2,
UnityEditor.MessageType.
Warning)]
#endif  //
UNITY_EDITOR
    public
EmulatorConnectionMode
emulatorConnectionMode
=
EmulatorConnectionMode.
USB;

    private static
GvrControllerInputDevice[
] instances = new
GvrControllerInputDevice[
0];
    private static
IControllerProvider
controllerProvider;
    private static
GvrSettings.UserPrefsHan
dedness handedness;
    private static Action
onDevicesChangedInternal
;

    /// <summary>Event
handler for when the
connection state of a
controller
changes.</summary>
    /// <param
name="state">The new
state.</param>
    /// <param
name="oldState">The
previous state.</param>
    public delegate void
OnStateChangedEvent(Gv
rConnectionState state,
GvrConnectionState
oldState);

    /// <summary>
    /// Event handler for
receiving button, touchpad,
and IMU updates from the
controllers.
    /// </summary>
    /// <remarks>Use this
handler to update app state
based on controller
input.</remarks>
    public static event
Action
OnControllerInputUpdated
;

    /// <summary>
    /// Event handler for
receiving a second
notification callback, after
all
    ///
`OnControllerInputUpdate
d` events have fired.
    /// </summary>
    public static event
Action
OnPostControllerInputUpd
ated;

    /// <summary>Event
handler for when controller
devices have
changed.</summary>
    /// <remarks>
    /// Any code that stores a
`GvrControllerInputDevice
```

```
` should get a new device instance from
    /// `GetDevice`. Existing `GvrControllerInputDevice`s will be marked invalid and will log errors
    /// when used. Event handlers are called immediately when added.
    /// </remarks>
    public static event Action OnDevicesChanged
    {

[SuppressMemoryAllocationError(
            IsWarning = false, Reason = "Only called on input device change.")]
        add
        {

onDevicesChangedInternal += value;
            value();
        }


[SuppressMemoryAllocationError(
            IsWarning = false, Reason = "Only called on input device change.")]
        remove
        {

onDevicesChangedInternal -= value;
        }
    }

    /// <summary>
    /// Event handler for when the connection state of the dominant controller changes.
    /// </summary>

[System.Obsolete("Replaced by GvrControllerInputDevice.OnStateChangedEvent.")]
    public static event OnStateChangedEvent OnStateChanged
    {
        add
        {
            if (instances.Length > 0)
            {

instances[0].OnStateChanged += value;
            }
            else
            {
                Debug.LogError(

"GvrControllerInput: Adding OnStateChanged event before instance created.");
            }
        }

        remove
        {
            if (instances.Length > 0)
            {

instances[0].OnStateChanged -= value;
            }
            else
            {
                Debug.LogError(

"GvrControllerInput: Removing OnStateChanged event before instance created.");
            }
        }
    }

    /// <summary>Controller Emulatuor connection modes.</summary>
    public enum EmulatorConnectionMode
    {
        /// <summary>Emulator disconnected.</summary>
        OFF,

        /// <summary>Emulator connects over USB.</summary>
        USB,

        /// <summary>Emulator connects over WIFI.</summary>
        WIFI,
    }

    /// @deprecated Replaced by `GvrControllerInputDevice.State`.
    /// <summary>Gets the dominant controller's current connection state.</summary>
    /// <remarks>
    /// Returns `GvrConnectionState.Error` if `GvrControllerInput` is uninitialized.
    /// </remarks>
    /// <value>The state.</value>

[System.Obsolete("Replaced by GvrControllerInputDevice.State.")]
    public static GvrConnectionState State
    {
        get
        {
            if (instances.Length == 0)
            {
                return GvrConnectionState.Error;
            }

            return instances[0].State;
        }
    }

    /// <summary>Gets the status of the controller API.</summary>
    /// <remarks>
    /// Returns `GvrControllerApiStatus.Error` if `GvrControllerInput` is
```

```
uninitialized.
    /// </remarks>
    /// <value>The api
status.</value>
    public static
GvrControllerApiStatus
ApiStatus
    {
        get
        {
            if (instances.Length
== 0)
            {
                return
GvrControllerApiStatus.Er
ror;
            }

            return
instances[0].ApiStatus;
        }
    }

    /// <summary>Gets a
value indicating whether
battery status is
supported.</summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// Value `true` if the
GVR Controller Input
supports BatteryStatus
calls, `false` otherwise.
    /// </value>
    public static bool
SupportsBatteryStatus
    {
        get
        {
            if
(controllerProvider ==
null)
            {
                return false;
            }

            return
controllerProvider.Support
sBatteryStatus;
        }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.Orientation`.
    /// <summary>
    /// Gets the dominant
controller's current
orientation in space, as a
quaternion.
    /// </summary>
    /// <remarks>
    /// The rotation is
provided in 'orientation
space' which means the
rotation is given relative
    /// to the last time the
user recentered their
controllers.  To make a
game object in your scene
    /// have the same
orientation as the dominant
controller, simply assign
this quaternion to the
    /// object's
`transform.rotation`.  To
match the relative rotation,
use
    ///
`transform.localRotation`
instead.
    /// </remarks>
    /// <value>
    /// The orientation.  This
is `Quaternion.identity` if
`GvrControllerInput` is
uninitialized.
    /// </value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
Orientation.")]
    public static Quaternion
Orientation
    {
        get
        {
            if (instances.Length
== 0)
            {
                return
Quaternion.identity;
            }

            return
instances[0].Orientation;
        }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.Gyro`.
    /// <summary>
    /// Gets the dominant
controller's current angular
speed in radians per
second.
    /// </summary>
    /// <remarks>
    /// Uses the right-hand
rule (positive means a
right-hand rotation about
the given axis), as
    /// measured by the
controller's gyroscope.
Returns `Vector3.zero` if
`GvrControllerInput` is
    /// uninitialized.
    /// <para>
    /// The controller's axes
are:
    /// - X points to the right.
    /// - Y points
perpendicularly up from
the controller's top surface.
    /// - Z lies along the
controller's body, pointing
towards the front.
    /// </para></remarks>
    /// <value>The gyro's
angular speed.</value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
Gyro.")]
    public static Vector3
Gyro
    {
        get
        {
            if (instances.Length
== 0)
            {
                return
Vector3.zero;
            }

            return
instances[0].Gyro;
        }
    }
```

```
    /// @deprecated Replaced by `GvrControllerInputDevice.Accel`.
    /// <summary>
    /// Gets the dominant controller's current acceleration in meters per second squared.
    /// </summary>
    /// <remarks>
    /// The controller's axes are:
    /// - X points to the right.
    /// - Y points perpendicularly up from the controller's top surface.
    /// - Z lies along the controller's body, pointing towards the front.
    /// <para>
    /// Note that gravity is indistinguishable from acceleration, so when the controller is resting
    /// on a surface, expect to measure an acceleration of 9.8 m/s^2 on the Y axis. The
    /// accelerometer reading will be zero on all three axes only if the controller is in free fall,
    /// or if the user is in a zero gravity environment like a space station.
    /// </para></remarks>
    /// <value>
    /// The acceleration. Will be `Vector3.zero` if `GvrControllerInput` is uninitialized.
    /// </value>

    [System.Obsolete("Replaced by GvrControllerInputDevice.Accel.")]
    public static Vector3 Accel
    {
      get
      {
        if (instances.Length == 0)
        {
          return Vector3.zero;
        }

        return instances[0].Accel;
      }
    }

    /// @deprecated Replaced by
    /// `GvrControllerInputDevice.GetButton(GvrControllerButton.TouchPadTouch)`.
    /// <summary>
    /// Gets a value indicating whether the user is touching the dominant controller's touchpad.
    /// </summary>
    /// <remarks>Returns `false` if `GvrControllerInput` is uninitialized.</remarks>
    /// <value>Value `true` if is touching. Otherwise, `false`.</value>
    [System.Obsolete(
        "Replaced by GvrControllerInputDevice.GetButton(GvrControllerButton.TouchPadTouch).")]
    public static bool IsTouching
    {
      get
      {
        if (instances.Length == 0)
        {
          return false;
        }

        return instances[0].GetButton(GvrControllerButton.TouchPadTouch);
      }
    }

    /// @deprecated Replaced by
    /// `GvrControllerInputDevice.GetButtonDown(GvrControllerButton.TouchPadTouch)`.
    /// <summary>
    /// Gets a value indicating whether this frame is the frame the user starts touching the
    /// dominant controller's touchpad.
    /// </summary>
    /// <remarks>
    /// Returns `false` if `GvrControllerInput` is uninitialized.  Every `TouchDown` event is
    /// guaranteed to be followed by exactly one `TouchUp` event in a later frame. Also, `TouchDown`
    /// and `TouchUp` will never both be `true` in the same frame.
    /// </remarks>
    /// <value>
    /// Value `true` if this is the frame after the user starts touching the dominant controller's
    /// touchpad, `false` otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by GvrControllerInputDevice.GetButtonDown(GvrControllerButton.TouchPadTouch).")]
    public static bool TouchDown
    {
      get
      {
        if (instances.Length == 0)
        {
          return false;
        }

        return instances[0].GetButtonDown(GvrControllerButton.TouchPadTouch);
      }
    }
```

```
    /// @deprecated
Replaced by
    ///
`GvrControllerInputDevice
.GetButtonUp(GvrControll
erButton.TouchPadTouch)
`.
    /// <summary>
    /// Gets a value
indicating whether this
frame is the frame after the
user stops touching the
    /// dominant controller's
touchpad.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized.  Every
`TouchUp` event is
    /// guaranteed to be
preceded by exactly one
`TouchDown` event in an
earlier frame. Also,
    /// `TouchDown` and
`TouchUp` will never both
be `true` in the same
frame.
    /// </remarks>
    /// <value>
    /// Value `true` if this is
the frame after the user
stops touching the
dominant controller's
    /// touchpad, `false`
otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by
GvrControllerInputDevice.
GetButtonUp(GvrControll
erButton.TouchPadTouch).
")]
    public static bool
TouchUp
    {
        get
        {
            if (instances.Length
== 0)
            {
                return false;
            }

            return
instances[0].GetButtonUp(
GvrControllerButton.Touc
hPadTouch);
        }
    }

    /// @deprecated Please
migrate to the center-
relative
`GvrControllerInputDevice
.TouchPos`.
    /// <summary>
    /// Gets the position of
the dominant controller's
current touch, if touching
the touchpad.
    /// </summary>
    /// <remarks>
    /// Returns
`Vector2(0.5f, 0.5f)` if
`GvrControllerInput` is
uninitialized.  If not
touching,
    /// this is the position of
the last touch (when the
finger left the touchpad).
The X and Y
    /// range is from 0 to 1.
(0, 0) is the top left of the
touchpad and (1, 1) is the
bottom right
    /// of the touchpad.
    /// </remarks>
    /// <value>The touch
position.</value>

[System.Obsolete("Obsolet
e. Migrate to the center-
relative
GvrControllerInputDevice.
TouchPos.")]
    public static Vector2
TouchPos
    {
        get
        {
            if (instances.Length
== 0)
            {
                return new
Vector2(0.5f, 0.5f);
            }

            Vector2 touchPos =
instances[0].TouchPos;
            touchPos.x =
(touchPos.x / 2.0f) + 0.5f;
            touchPos.y = (-
touchPos.y / 2.0f) + 0.5f;
            return touchPos;
        }
    }

    /// @deprecated Please
migrate to the center-
relative
`GvrControllerInputDevice
.TouchPos`.
    /// <summary>
    /// Gets the position of
the dominant controller's
current touch, if touching
the touchpad.
    /// </summary>
    /// <remarks>
    /// Returns
`Vector2.zero` if
`GvrControllerInput` is
uninitialized.  If not
touching, this is
    /// the position of the last
touch (when the finger left
the touchpad).  The X and
Y range is
    /// from -1 to 1. (-.707,-
.707) is bottom left,
(.707,.707) is upper right.
(0, 0) is the
    /// center of the
touchpad.  The magnitude
of the touch vector is
guaranteed to be less than
or
    /// equal to 1.
    /// </remarks>
    /// <value>The touch
position centered.</value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
TouchPos.")]
    public static Vector2
TouchPosCentered
    {
        get
        {
            if (instances.Length
== 0)
            {
                return
Vector2.zero;
            }
```

```
            return
instances[0].TouchPos;
        }
    }

    /// @deprecated Use
`Recentered` to detect
when user has completed
the recenter gesture.
    /// <summary>Gets a
value indicating whether
the user is currently
recentering.</summary>
    /// <value>Value `true`
if the user is currently
recentering, `false`
otherwise.</value>
    [System.Obsolete("Use
Recentered to detect when
user has completed the
recenter gesture.")]
    public static bool
Recentering
    {
        get
        {
            return false;
        }
    }

    /// <summary>
    /// Gets a value
indicating whether the user
just completed the recenter
gesture.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized.  The headset
and the dominant
    /// controller's
orientation are now being
reported in the new
recentered coordinate
system.
    /// This is an event flag
(it is true for only one
frame after the event
happens, then reverts
    /// to false).
    /// </remarks>
    /// <value>Value `true`
if the user has just finished
recentering, `false`
otherwise.</value>
    public static bool
Recentered
    {
        get
        {
            if (instances.Length
== 0)
            {
                return false;
            }

            return
instances[0].Recentered;
        }
    }

    /// @deprecated
Replaced by
    ///
`GvrControllerInputDevice
.GetButton(GvrController
Button.TouchPadButton)`.
    /// <summary>
    /// Gets a value
indicating whether the user
currently holds down the
dominant controller's
    /// touchpad button.
    /// </summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// Value `true` if the
user currently holds down
the dominant controller's
touchpad button,
    /// `false` otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by
GvrControllerInputDevice.
GetButton(GvrControllerB
utton.TouchPadButton).")]
    public static bool
ClickButton
    {
        get
        {
            if (instances.Length
== 0)
            {
                return false;
            }

            return
instances[0].GetButton(Gv
rControllerButton.TouchPa
dButton);
        }
    }

    /// @deprecated
Replaced by
    ///
`GvrControllerInputDevice
.GetButtonDown(GvrCont
rollerButton.TouchPadButt
on)`.
    /// <summary>
    /// Gets a value
indicating whether this is
the frame the user starts
pressing down the
dominant
    /// controller's touchpad
button.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized.  Every
`ClickButtonDown` event
is
    /// guaranteed to be
followed by exactly one
`ClickButtonUp` event in a
later frame. Also,
    /// `ClickButtonDown`
and `ClickButtonUp` will
never both be `true` in the
same frame.
    /// </remarks>
    /// <value>
    /// Value `true` this is
the frame the user started
pressing down the
dominant controller's
    /// touchpad button,
`false` otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by
GvrControllerInputDevice.
GetButtonDown(GvrContr
ollerButton.TouchPadButt
on).")]
    public static bool
ClickButtonDown
    {
```

```
      get
      {
        if (instances.Length
== 0)
        {
          return false;
        }

        return
instances[0].GetButtonDo
wn(GvrControllerButton.T
ouchPadButton);
      }
    }

    /// @deprecated
Replaced by
    ///
`GvrControllerInputDevice
.GetButtonUp(GvrControll
erButton.TouchPadButton)
`.
    /// <summary>
    /// Gets a value
indicating whether this is
the frame after the user
stops pressing down the
    /// dominant controller's
touchpad button.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized.  Every
`ClickButtonUp` event is
    /// guaranteed to be
preceded by exactly one
`ClickButtonDown` event
in an earlier frame. Also,
    /// `ClickButtonDown`
and `ClickButtonUp` will
never both be `true` in the
same frame.
    /// </remarks>
    /// <value>
    /// Value `true` if this is
the frame after the user
stops pressing down the
dominant
    /// controller's touchpad
button, `false` otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by
GvrControllerInputDevice.
GetButtonUp(GvrControll
erButton.TouchPadButton)
.")]
    public static bool
ClickButtonUp
    {
      get
      {
        if (instances.Length
== 0)
        {
          return false;
        }

        return
instances[0].GetButtonUp(
GvrControllerButton.Touc
hPadButton);
      }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.GetButton(GvrController
Button.App)`.
    /// <summary>
    /// Gets a value
indicating whether the user
is currently holding down
the dominant controller's
    /// app button.
    /// </summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// Value `true` if the
user is currently holding
down the dominant
controller's app button,
    /// `false` otherwise.
    /// </value>

    [System.Obsolete("Replac
ed by
GvrControllerInputDevice.
GetButton(GvrControllerB
utton.App).")]
    public static bool
AppButton
    {
      get
      {
        if (instances.Length
== 0)
        {
          return false;
        }

        return
instances[0].GetButton(Gv
rControllerButton.App);
      }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.GetButtonDown(GvrCont
rollerButton.App)`.
    /// <summary>
    /// Gets a value
indicating whether this is
the frame the user started
pressing down the
    /// dominant controller's
app button.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized. Every
`AppButtonDown` event is
    /// guaranteed to be
followed by exactly one
`AppButtonUp` event in a
later frame.
    /// `AppButtonDown`
and `AppButtonUp` will
never both be `true` in the
same frame.
    /// </remarks>
    /// <value>
    /// Value `true` if this is
the frame the user started
pressing down the
dominant controller's
    /// app button, `false`
otherwise.
    /// </value>
    [System.Obsolete(
        "Replaced by
GvrControllerInputDevice.
GetButtonDown(GvrContr
ollerButton.App).")]
    public static bool
AppButtonDown
    {
      get
      {
        if (instances.Length
```

```
== 0)
          {
             return false;
          }

          return
instances[0].GetButtonDo
wn(GvrControllerButton.A
pp);
       }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.GetButtonUp(GvrControll
erButton.App)`.
    /// <summary>
    /// Gets a value
indicating whether this is
the frame after the user
stopped pressing down the
    /// dominant controller's
app button.
    /// </summary>
    /// <remarks>
    /// Returns `false` if
`GvrControllerInput` is
uninitialized. Every
`AppButtonUp` event is
    /// guaranteed to be
preceded by exactly one
`AppButtonDown` event
in an earlier frame. Also,
    /// `AppButtonDown`
and `AppButtonUp` will
never both be `true` in the
same frame.
    /// </remarks>
    /// <value>
    /// Value `true` if this is
the frame after the user
stopped pressing down the
dominant
    /// controller's app
button, `false` otherwise.
    /// </value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
GetButtonUp(GvrControll
erButton.App).")]
    public static bool
AppButtonUp
    {
       get
       {
          if (instances.Length
== 0)
          {
             return false;
          }

          return
instances[0].GetButtonUp(
GvrControllerButton.App);
       }
    }

    /// @deprecated
Replaced by
    ///
`GvrControllerInputDevice
.GetButtonDown(GvrCont
rollerButton.System)`.
    /// <summary>
    /// Gets a value
indicating whether this is
the frame the user started
pressing down the
    /// dominant controller's
system button.
    /// </summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// Value `true` if this is
the frame the user started
pressing down the
dominant controller's
    /// system button, `false`
otherwise.
    /// </value>
    [System.Obsolete(
       "Replaced by
GvrControllerInputDevice.
GetButtonDown(GvrContr
ollerButton.System).")]
    public static bool
HomeButtonDown
    {
       get
       {
          if (instances.Length
== 0)
          {
             return false;
          }

          return
instances[0].GetButtonDo
wn(GvrControllerButton.S
ystem);
       }
    }

    /// @deprecated
Replaced by
GvrControllerInputDevice.
GetButton(GvrControllerB
utton.System).
    /// <summary>
    /// Gets a value
indicating whether the user
is holding down the
dominant controller's
system
    /// button.
    /// </summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// Value `true` if the
user is holding down the
dominant controller's
system button, `false`
    /// otherwise.
    /// </value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
GetButton(GvrControllerB
utton.System).")]
    public static bool
HomeButtonState
    {
       get
       {
          if (instances.Length
== 0)
          {
             return false;
          }

          return
instances[0].GetButton(Gv
rControllerButton.System);
       }
    }

    /// @deprecated
Replaced by
```

```
`GvrControllerInputDevice
.ErrorDetails`.
    /// <summary>
    /// Gets details about the
reasoning behind the
Dominant Controller's
error state.
    /// </summary>
    /// <remarks>
    /// If the dominant
controller's state `==
GvrConnectionState.Error`
, this contains details
    /// about the error.  If
`GvrControllerInput` is
uninitialized this returns an
error string
    /// describing the
uninitialized state.
    /// </remarks>
    /// <value>Details about
the reasoning behind the
dominant controller's error
state.</value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
ErrorDetails.")]
    public static string
ErrorDetails
    {
        get
        {
            if (instances.Length
> 0)
            {
                return
instances[0].ErrorDetails;
            }
            else
            {
                return "No
GvrControllerInput
initialized instance found
in scene. "
                        + "It may be
missing, or it might not
have initialized yet.";
            }
        }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.StatePtr`.
    /// <summary>
    /// Gets the GVR C
library controller state
pointer
(`gvr_controller_state*`)
for the dominant
    /// controller.
    /// </summary>
    /// <remarks>Returns
`IntPtr.Zero` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>
    /// The GVR C library
controller state pointer
(`gvr_controller_state*`)
for the dominant
    /// controller.
    /// </value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
StatePtr.")]
    public static IntPtr
StatePtr
    {
        get
        {
            if (instances.Length
== 0)
            {
                return
IntPtr.Zero;
            }

            return
instances[0].StatePtr;
        }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.IsCharging`.
    /// <summary>
    /// Gets a value
indicating whether the
dominant controller is
currently being charged.
    /// </summary>
    /// <remarks>Returns
`false` if
`GvrControllerInput` is
uninitialized.</remarks>
    /// <value>Value `true`
if the dominant controller
is charging.  Otherwise,
`false`.</value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
IsCharging.")]
    public static bool
IsCharging
    {
        get
        {
            if (instances.Length
== 0)
            {
                return false;
            }

            return
instances[0].IsCharging;
        }
    }

    /// @deprecated
Replaced by
`GvrControllerInputDevice
.BatteryLevel`.
    /// <summary>Gets the
dominant controller's
current battery charge
level.</summary>
    /// <remarks>
    /// Returns
`GvrControllerBatteryLeve
l.Error` if
`GvrControllerInput` is
uninitialized.
    /// </remarks>
    /// <value>The dominant
controller's current battery
charge level.</value>

[System.Obsolete("Replac
ed by
GvrControllerInputDevice.
BatteryLevel.")]
    public static
GvrControllerBatteryLevel
BatteryLevel
    {
        get
        {
            if (instances.Length
== 0)
```

```
        {
            return
GvrControllerBatteryLevel
.Error;
        }

        return
instances[0].BatteryLevel;
      }
   }

   /// <summary>Returns a
controller device for the
specified
hand.</summary>
   /// <returns>The
controller input
device.</returns>
   /// <param
name="hand">The hand
whose input device should
be fetched.</param>
   public static
GvrControllerInputDevice
GetDevice(GvrController
Hand hand)
   {
      if (instances.Length
== 0)
      {
         return null;
      }

      // Remap Right and
Left to Dominant or
NonDominant according to
settings handedness.
      if (hand ==
GvrControllerHand.Left ||
hand ==
GvrControllerHand.Right)
      {
         if ((int)hand !=
(int)handedness)
         {
            hand =
GvrControllerHand.NonD
ominant;
         }
         else
         {
            hand =
GvrControllerHand.Domin
ant;
         }
      }
      if (hand ==
GvrControllerHand.NonD
ominant)
      {
         return instances[1];
      }
      else
      {
         // Dominant is
always controller 0.
         return instances[0];
      }
   }

   private void Awake()
   {
      if (instances.Length >
0)
      {
         Debug.LogError(
            "More than one
active GvrControllerInput
instance was found in your
scene. "
            + "Ensure that
there is only one
GvrControllerInput.");
         this.enabled =
false;
         return;
      }

      if (controllerProvider
== null)
      {
         controllerProvider
=
ControllerProviderFactory.
CreateControllerProvider(t
his);
      }

      handedness =
GvrSettings.Handedness;
      int controllerCount =
2;
      instances = new
GvrControllerInputDevice[
controllerCount];
      for (int i = 0; i <
controllerCount; i++)
      {
         instances[i] = new
GvrControllerInputDevice(
controllerProvider, i);
      }

      if
(onDevicesChangedInterna
l != null)
      {

onDevicesChangedInternal
();
      }

      // Keep screen on
here, since
GvrControllerInput must
be in any GVR scene in
order to enable
      // controller
capabilities.
      Screen.sleepTimeout
=
SleepTimeout.NeverSleep;
   }

   private void Update()
   {
      foreach (var instance
in instances)
      {
         if (instance != null)
         {

instance.Update();
         }
      }

      if
(OnControllerInputUpdate
d != null)
      {

OnControllerInputUpdated
();
      }

      if
(OnPostControllerInputUp
dated != null)
      {

OnPostControllerInputUpd
ated();
      }
   }

   private void
OnDestroy()
```

```
    {
      foreach (var instance
in instances)
      {
        // Ensure this
device will error if used
again.

instance.Invalidate();
      }

      instances = new
GvrControllerInputDevice[
0];
      if
(onDevicesChangedInterna
l != null)
      {

onDevicesChangedInternal
();
      }
    }

    private void
OnApplicationPause(bool
paused)
    {
      if (controllerProvider
== null)
      {
        return;
      }

      if (paused)
      {

controllerProvider.OnPaus
e();
      }
      else
      {
        handedness =
GvrSettings.Handedness;

controllerProvider.OnResu
me();
      }
    }
}
```

GvrPointerInputModule.Cs

```
//-----------------------------
-----------------------------
--------
// <copyright
file="GvrPointerInputMod
ule.cs" company="Google
Inc.">
// Copyright 2016 Google
Inc. All rights reserved.
//
// Licensed under the MIT
License, you may not use
this file except in
// compliance with the
License. You may obtain a
copy of the License at
//
//
http://www.opensource.org
/licenses/mit-license.php
//
// Unless required by
applicable law or agreed to
in writing, software
// distributed under the
License is distributed on
an "AS IS" BASIS,
// WITHOUT
WARRANTIES OR
CONDITIONS OF ANY
KIND, either express or
implied.
// See the License for the
specific language
governing permissions and
// limitations under the
License.
// </copyright>
//-----------------------------
-----------------------------
--------

using
System.Collections.Generi
c;
using Gvr.Internal;
using UnityEngine;
using
UnityEngine.EventSystem
s;

/// <summary>This script
provides an implemention
of Unity's
`BaseInputModule`
class.</summary>
/// <remarks><para>
/// Exists so that Canvas-
based (`uGUI`) UI
elements and 3D scene
objects can be interacted
with in
/// a Gvr Application.
/// </para><para>
/// This script is intended
for use with either a 3D
Pointer with the Daydream
Controller
/// (Recommended for
Daydream), or a Gaze-
based-Pointer
(Recommended for
Cardboard).
/// </para><para>
/// To use, attach to the
scene's **EventSystem**
object.  Be sure to move it
above the
/// other modules, such as
`TouchInputModule` and
`StandaloneInputModule`,
in order
/// for the Pointer to take
priority in the event
system.
/// </para><para>
/// If you are using a
**Canvas**, set the
`Render Mode` to **World
Space**, and add the
///
`GvrPointerGraphicRaycas
ter` script to the object.
/// </para><para>
/// If you'd like pointers to
work with 3D scene
objects, add a
`GvrPointerPhysicsRaycas
ter` to the
/// main camera, and add a
component that
implements one of the
`Event` interfaces
(`EventTrigger`
/// will work nicely) to an
object with a collider.
/// </para><para>
///
`GvrPointerInputModule`
emits the following events:
`Enter`, `Exit`, `Down`,
`Up`, `Click`,
```

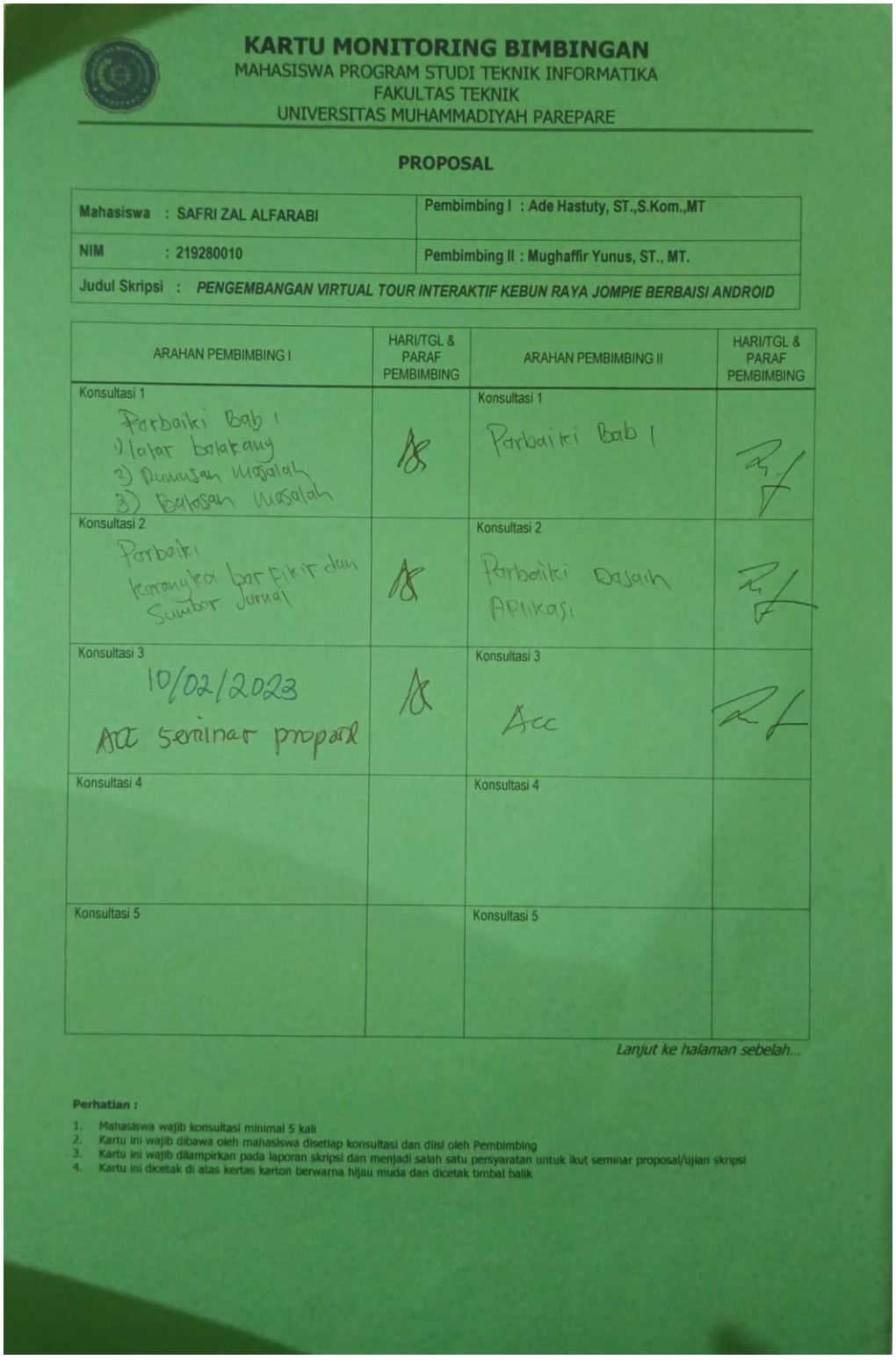
```
/// `Select`, `Deselect`, `UpdateSelected`, and `GvrPointerHover`.
Scroll, move, and submit/cancel
/// events are not emitted.
/// </para><para>
/// To use a 3D Pointer with the Daydream Controller:
///   - Add the prefab GoogleVR/Prefabs/UI/GvrControllerPointer to your scene.
///   - Set the parent of `GvrControllerPointer` to the same parent as the main camera
///     (With a local position of 0,0,0).
/// </para><para>
/// To use a Gaze-based-pointer:
///   - Add the prefab GoogleVR/Prefabs/UI/GvrReticlePointer to your scene.
///   - Set the parent of `GvrReticlePointer` to the main camera.
/// </para></remarks>
[AddComponentMenu("GoogleVR/GvrPointerInputModule")]
[HelpURL("https://developers.google.com/vr/unity/reference/class/GvrPointerInputModule")]
public class GvrPointerInputModule : BaseInputModule, IGvrInputModuleController
{
  /// <summary>
  /// If `true`, pointer input is active in VR Mode only.
  /// If `false`, pointer input is active all of the time.
  /// </summary>
  /// <remarks>
  /// Set to false if you plan to use direct screen taps or other input when not in VR Mode.
  /// </remarks>
  [Tooltip("Whether Pointer input is active in VR Mode only (true), or all the time (false).")]
  public bool vrModeOnly = false;

  /// <summary>Manages scroll events for the input module.</summary>
  [Tooltip("Manages scroll events for the input module.")]
  public GvrPointerScrollInput scrollInput = new GvrPointerScrollInput();

  /// <summary>Gets or sets the static reference to the `GvrBasePointer`.</summary>
  /// <value>The static reference to the `GvrBasePointer`.</value>
  public static GvrBasePointer Pointer
  {
    get
    {

      GvrPointerInputModule module = FindInputModule();
      if (module == null || module.Impl == null)
      {
        return null;
      }

      return module.Impl.Pointer;
    }

    set
    {

      GvrPointerInputModule module = FindInputModule();
      if (module == null || module.Impl == null)
      {
        return;
      }


      module.Impl.Pointer = value;
    }
  }

  /// <summary>Gets the current `RaycastResult`.</summary>
  /// <value>The current `RaycastResult`.</value>
  public static RaycastResult CurrentRaycastResult
  {
    get
    {

      GvrPointerInputModule inputModule = GvrPointerInputModule.FindInputModule();
      if (inputModule == null)
      {
        return new RaycastResult();
      }

      if (inputModule.Impl == null)
      {
        return new RaycastResult();
      }

      if (inputModule.Impl.CurrentEventData == null)
      {
        return new RaycastResult();
      }

      return inputModule.Impl.CurrentEventData.pointerCurrentRaycast;
    }
  }

  /// <summary>Gets the
```

```
implementation object of this module.</summary>
    /// <value>The implementation object of this module.</value>
    public GvrPointerInputModuleImpl Impl { get; private set; }

    /// <summary>Gets the executor this module uses to process events.</summary>
    /// <value>The executor this module uses to process events.</value>
    public GvrEventExecutor EventExecutor { get; private set; }

    /// <summary>Gets the event system reference.</summary>
    /// <value>The event system reference.</value>

[System.Diagnostics.CodeAnalysis.SuppressMessage(

"UnityRules.LegacyGvrStyleRules",

"VR1001:AccessibleNonConstantPropertiesMustBeUpperCamelCase",
        Justification = "Legacy Public API.")]
    public new EventSystem eventSystem
    {
        get
        {
            return base.eventSystem;
        }
    }

    /// <summary>Gets the list of raycast results used as a cache.</summary>
    /// <value>The list of raycast results used as a cache.</value>
    public List<RaycastResult> RaycastResultCache
    {
        get
        {
            return m_RaycastResultCache;
        }
    }

    /// <summary>The `GvrBasePointer` calls this when it is created.</summary>
    /// <remarks>
    /// If a pointer hasn't already been assigned, it will assign the newly created one by default.
    /// This simplifies the common case of having only one `GvrBasePointer` so it can be
    /// automatically hooked up to the manager.  If multiple `GvrBasePointers` are in the scene,
    /// the app has to take responsibility for setting which one is active.
    /// </remarks>
    /// <param name="createdPointer">The pointer whose creation triggered this call.</param>
    public static void OnPointerCreated(GvrBasePointer createdPointer)
    {

GvrPointerInputModule module = FindInputModule();
        if (module == null || module.Impl == null)
        {
            return;
        }

        if (module.Impl.Pointer == null)
        {
            module.Impl.Pointer = createdPointer;
        }
    }

    /// <summary>
    /// Helper function to find the Event executor that is part of the input module if one exists
    /// in the scene.
    /// </summary>
    /// <returns>A found GvrEventExecutor or null.</returns>
    public static GvrEventExecutor FindEventExecutor()
    {

GvrPointerInputModule gvrInputModule = FindInputModule();
        if (gvrInputModule == null)
        {
            return null;
        }

        return gvrInputModule.EventExecutor;
    }

    /// <summary>
    /// Helper function to find the input module if one exists in the scene and it is the active
    /// module.
    /// </summary>
    /// <returns>A found `GvrPointerInputModule` or null.</returns>
    public static GvrPointerInputModule FindInputModule()
    {
        if (EventSystem.current == null)
        {
            return null;
        }

        EventSystem
```

```
eventSystem =
EventSystem.current;
      if (eventSystem ==
null)
      {
         return null;
      }


GvrPointerInputModule
gvrInputModule =

eventSystem.GetCompone
nt<GvrPointerInputModul
e>();

      return
gvrInputModule;
   }

   /// <inheritdoc/>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public override bool
ShouldActivateModule()
   {
      return
Impl.ShouldActivateModu
le();
   }

   /// <inheritdoc/>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public override void
DeactivateModule()
   {

Impl.DeactivateModule();
   }

   /// <inheritdoc/>
   public override bool
IsPointerOverGameObject
(int pointerId)
   {
      return
Impl.IsPointerOverGameO
bject(pointerId);
   }

   /// <inheritdoc/>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public override void
Process()
   {

UpdateImplProperties();
      Impl.Process();
   }

   /// <summary>Whether
the module should be
activated.</summary>
   /// <returns>Returns
`true` if this module should
be activated, `false`
otherwise.</returns>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public bool
ShouldActivate()
   {
      return
base.ShouldActivateModul
e();
   }

   ///
<summary>Deactivate this
instance.</summary>
   public void Deactivate()
   {

base.DeactivateModule();
   }

   /// <summary>Finds the
common root between two
`GameObject`s.</summary
>
   /// <returns>The
common root.</returns>
   /// <param
name="g1">The first
`GameObject`.</param>
   /// <param
name="g2">The second
`GameObject`.</param>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public new GameObject
FindCommonRoot(GameO
bject g1, GameObject g2)
   {
      return
BaseInputModule.FindCo
mmonRoot(g1, g2);
   }

   /// <summary>Gets the
base event
data.</summary>
   /// <returns>The base
event data.</returns>

[SuppressMemoryAllocati
onError(IsWarning = true,
Reason = "Pending
documentation.")]
   public new
BaseEventData
GetBaseEventData()
   {
      return
base.GetBaseEventData();
   }

   /// <summary>Finds the
first raycast.</summary>
   /// <returns>The first
raycast.</returns>
   /// <param
name="candidates">
   /// The list of
`RaycastResult`s to search
for the first Raycast.
   /// </param>
   public new
RaycastResult
FindFirstRaycast(List<Ray
castResult> candidates)
   {
      return
BaseInputModule.FindFirs
tRaycast(candidates);
   }

   /// @cond
   /// <inheritdoc/>
   protected override void
Awake()
```

```
    {
        base.Awake();
        Impl = new
GvrPointerInputModuleIm
pl();
        EventExecutor = new
GvrEventExecutor();

UpdateImplProperties();
    }

    /// @endcond

    /// <summary>Update
implementation
properties.</summary>
    private void
UpdateImplProperties()
    {
        if (Impl == null)
        {
            return;
        }

        Impl.ScrollInput =
scrollInput;
        Impl.VrModeOnly =
vrModeOnly;

Impl.ModuleController =
this;
        Impl.EventExecutor =
EventExecutor;
    }
}
```

Lampiran - 2 Kartu Monitoring Bimbingan Proposal



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : SAFRI ZAL ALFARABI	**Pembimbing I : Ade Hastuty, ST.,S.Kom.,MT**
NIM : 219280010	**Pembimbing II : Mughaffir Yunus, ST., MT.**
Judul Skripsi : *PENGEMBANGAN VIRTUAL TOUR INTERAKTIF KEBUN RAYA JOMPIE BERBAISI ANDROID*	

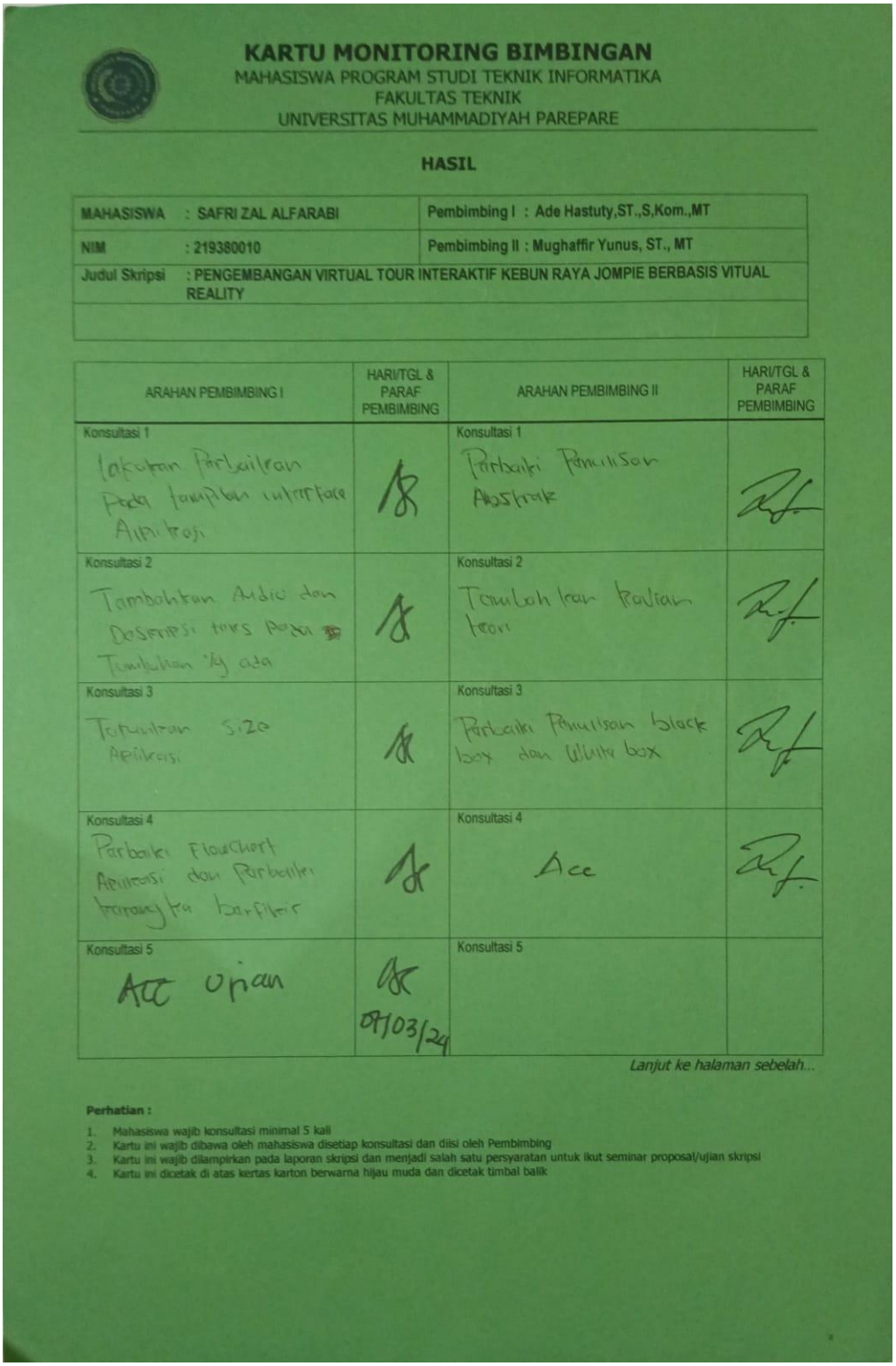
ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Perbaiki Bab I 1) latar belakang 2) Rumusan Masalah 3) Batasan Masalah		Konsultasi 1 Perbaiki Bab I	
Konsultasi 2 Perbaiki kerangka berfikir dan Sumber Jurnal		Konsultasi 2 Perbaiki Desain Aplikasi	
Konsultasi 3 10/02/2023 Acc seminar proposal		Konsultasi 3 Acc	
Konsultasi 4		Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran - 3 Kartu Monitoring Bimbingan Skripsi



KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

HASIL

MAHASISWA	: SAFRI ZAL ALFARABI	Pembimbing I	: Ade Hastuty,ST.,S,Kom.,MT
NIM	: 219380010	Pembimbing II	: Mughaffir Yunus, ST., MT
Judul Skripsi	: PENGEMBANGAN VIRTUAL TOUR INTERAKTIF KEBUN RAYA JOMPIE BERBASIS VITUAL REALITY		

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Lakukan Perbaikan Pada tampilan interface Aplikasi		Konsultasi 1 Perbaiki Penulisan Abstrak	
Konsultasi 2 Tambahkan Audio dan Deskripsi tiap Pojok Tambahkan sitasi		Konsultasi 2 Tambahkan kajian teori	
Konsultasi 3 Tambahkan Saran Aplikasi		Konsultasi 3 Perbaiki Penulisan black box dan White box	
Konsultasi 4 Perbaiki Flowchart Aplikasi dan Perbaiki kerangka berfikir		Konsultasi 4 Acc	
Konsultasi 5 Acc Ujian	09/03/24	Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lanjutan...

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 6 Sesuaikan revisi dari penguji		Konsultasi 6 Perbaiki Penulisan	
Konsultasi 7 Abstrak		Konsultasi 7 Batasan masalah dan tujuan	
Konsultasi 8 Rumusan masalah dan Kesimpulan		Konsultasi 8 Kesimpulan dan Saran	
Konsultasi 9 Acc Ujian		Konsultasi 9	
Konsultasi 10		Konsultasi 10	

Parepare, 19 Februari 2024

Mengetahui
Ketua Program Studi

Marlina, S.Kom.,M.Kom.
NBM. 1162 680

Mahasiswa

SAFRI ZAL ALFARABI
NIM. **219280010**

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran - 4 Sertifikat Laboratorium Praktikum V

SERTIFIKAT

Nomor : 433/PRK/LB-TI/II.3.AU/D/2023

Diberikan Kepada :

Safri Zal Alfarabi

NIM.219280010

Telah Menyelesaikan

PRAKTIKUM V

Laboratorium Teknik Informatika
Program Studi Teknik Informatika Fakultas Teknik
Universitas Muhammadiyah Parepare

Pada tanggal 16 Januari 2023 - 29 Maret 2023

Parepare, 15 Juni 2023

KETUA PRODI
TEKNIK INOFORMATIKA

Marlina. S.kom., M.kom
NBM. 1162 680

KEPALA
LABORATORIUM

Ir. Untung Suwardoyo, S.Kom., MT., IPP.
NBM. 1288 973

Nilai Praktikum laboratorium Teknik Informatika
Fakultas Teknik Universitas Muhammadiyah Parepare Tahun 2023

MATERI	NILAI ANGKA
PENGOLAHAN CITRA DIGITAL	92,5
AUGMENTED DAN VIRTUAL REALITY	80
MIKROKONTROLER	85
PEMROGRAMAN MOBILE	89
RATA-RATA	86,63
NILAI HURUF	A