LAMPIRAN

Menu

```
using UnityEngine;

namespace Vuforia
{
    /// <summary>
    /// A custom handler th
at implements the ITrackabl
eEventHandler interface.
    /// </summary>
    public class DefaultTra
ckableEventHandler : MonoBe
haviour,

                      ITrack
ableEventHandler
    {
        #region PRIVATE_MEM
BER_VARIABLES

        private TrackableBe
haviour mTrackableBehaviour
;

        #endregion // PRIVA
TE_MEMBER_VARIABLES



        #region UNTIY_MONOB
EHAVIOUR_METHODS

        void Start()
        {
            mTrackableBehav
iour = GetComponent<Trackab
leBehaviour>();
            if (mTrackableB
ehaviour)
            {
                mTrackableB
ehaviour.RegisterTrackableE
ventHandler(this);
            }
        }

        #endregion // UNTIY
_MONOBEHAVIOUR_METHODS



        #region PUBLIC_METH
ODS

        /// <summary>
        /// Implementation
of the ITrackableEventHandl
er function called when the
        /// tracking state
changes.
        /// </summary>
        public void OnTrack
ableStateChanged(

            TrackableBehav
iour.Status previousStatus,

            TrackableBehav
iour.Status newStatus)
        {
            if (newStatus =
= TrackableBehaviour.Status
.DETECTED ||
                newStatus =
= TrackableBehaviour.Status
.TRACKED ||
                newStatus =
= TrackableBehaviour.Status
.EXTENDED_TRACKED)
            {
                OnTrackingF
ound();
            }
            else
            {
                OnTrackingL
ost();
            }
```

```
        }

        #endregion // PUBLIC_METHODS



        #region PRIVATE_METHODS


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

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

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

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


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

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

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

            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " lost");
        }

        #endregion // PRIVATE_METHODS
```

```
    }
}
```

Button Manager

```
using UnityEngine;
using System.Collections;
using UnityEngine.SceneMana
gement;
public class ButtonManager
: MonoBehaviour {

    public void startbtn(st
ring newGameLevel)
    {
        SceneManager.LoadSc
ene(newGameLevel);
    }
}
```

Camera Focus Controller

```
using UnityEngine;
using System.Collections;
using Vuforia;

public class CameraFocusCon
troller : MonoBehaviour {

    private bool mVuforiaSt
arted = false;

    void Start ()
    {
        VuforiaARController
 vuforia = VuforiaARControl
ler.Instance;

        if (vuforia != null
)
            vuforia.Registe
rVuforiaStartedCallback(Sta
rtAfterVuforia);
    }

    private void StartAfter
Vuforia()
    {
        mVuforiaStarted = t
rue;
        SetAutofocus();
    }

    void OnApplicationPause
(bool pause)
    {
        if (!pause)
        {

            if (mVuforiaSta
rted)
            {

                SetAutofocu
s();
            }
        }
    }

    private void SetAutofoc
us()
    {
        if (CameraDevice.In
stance.SetFocusMode(CameraD
evice.FocusMode.FOCUS_MODE_
CONTINUOUSAUTO))
        {
            Debug.Log("Auto
focus set");
        }
        else
        {
            // never actual
ly seen a device that doesn
't support this, but just i
n case
            Debug.Log("this
```

```
 device doesn't support aut
o focus");
            }
        }
}
```

Splash Screen

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneMana
gement;

public class LoadingScript
: MonoBehaviour {

    public float loadingTim
e;
    public Image loadingBar
;
    public Text percent;

    void Start () {
        loadingBar.fillAmou
nt = 0;
    }

    void Update () {

        if (loadingBar.fill
Amount <= 1) {
            loadingBar.fill
Amount += 1.0f / loadingTim
e * Time.deltaTime;

    }
        if (loadingBar.fill
Amount == 1.0f) {
            Application.Loa
dLevel (1);
        }

        percent.text = (loa
dingBar.fillAmount * 100).T
oString ("f0");

}

}
```

Exit

```
using System.Collections;
using System.Collections.Ge
neric;
using UnityEngine;

public class Keluar : MonoB
ehaviour {

    public void KeluarDariG
ame(){
        Application.Quit()
;
}
}
```

Lean Touch

```
using UnityEngine;
using UnityEngine.EventSyst
ems;
using System.Collections.Ge
neric;

namespace Lean.Touch
{

    [ExecuteInEditMode]
    [DisallowMultipleCompon
ent]
    public partial class Le
anTouch : MonoBehaviour
    {

        public static List<
LeanTouch> Instances = new
```

```
List<LeanTouch>();

        public static List<
LeanFinger> Fingers = new L
ist<LeanFinger>(10);

        public static List<
LeanFinger> InactiveFingers
 = new List<LeanFinger>(10)
;

        public static Syste
m.Action<LeanFinger> OnFing
erDown;

        public static Syste
m.Action<LeanFinger> OnFing
erSet;

        public static Syste
m.Action<LeanFinger> OnFing
erUp;

        public static Syste
m.Action<LeanFinger> OnFing
erTap;

        public static Syste
m.Action<LeanFinger> OnFing
erSwipe;

        public static Syste
m.Action<List<LeanFinger>>
OnGesture;

        public float TapThr
eshold = DefaultTapThreshol
d;

        public const float
DefaultTapThreshold = 0.5f;

        public static float
 CurrentTapThreshold
        {
            get
            {
                return Inst
ances.Count > 0 ? Instances
[0].TapThreshold : DefaultT
apThreshold;
            }
        }

        public float SwipeT
hreshold = DefaultSwipeThre
shold;

        public const float
DefaultSwipeThreshold = 50.
0f;

        public static float
 CurrentSwipeThreshold
        {
            get
            {
                return Inst
ances.Count > 0 ? Instances
[0].SwipeThreshold : Defaul
tSwipeThreshold;
            }
        }

        [Tooltip("This allo
ws you to set the default D
PI you want the input scali
ng to be based on")]
        public int Referenc
eDpi = DefaultReferenceDpi;
```

```
        public const int DefaultReferenceDpi = 200;

        public static int CurrentReferenceDpi
        {
            get
            {
                return Instances.Count > 0 ? Instances[0].ReferenceDpi : DefaultReferenceDpi;
            }
        }



        public static LayerMask CurrentGuiLayers
        {
            get
            {
                return Instances.Count > 0 ? Instances[0].GuiLayers : (LayerMask)Physics.DefaultRaycastLayers;
            }
        }

        [Tooltip("This allows you to enable recording of finger movements")]
        public bool RecordFingers = true;

        [Tooltip("This allows you to set the amount of pixels a finger must move for another snapshot to be stored")]
        public float RecordThreshold = 5.0f;

        [Tooltip("This allows you to set the maximum amount of seconds that can be recorded, 0 = unlimited")]
        public float RecordLimit = 10.0f;

        [Tooltip("This allows you to simulate multi touch inputs on devices that don't support them (e.g. desktop)")]
        public bool SimulateMultiFingers = true;

        [Tooltip("This allows you to set which key is required to simulate multi key twisting")]
        public KeyCode PinchTwistKey = KeyCode.LeftControl;

        [Tooltip("This allows you to set which key is required to simulate multi key dragging")]
        public KeyCode MultiDragKey = KeyCode.LeftAlt;

        [Tooltip("This allows you to set which texture will be used to show the simulated fingers")]
        public Texture2D FingerTexture;

        // This stores the highest mouse button index
        private static int highestMouseButton = 7;

        // Used to find if the GUI is in use
        private static List
```

```
<RaycastResult> tempRaycast
Results = new List<RaycastR
esult>(10);


        private static List
<LeanFinger> filteredFinger
s = new List<LeanFinger>(10
);


        private static Poin
terEventData tempPointerEve
ntData;

        // Used by RaycastG
ui
        private static Even
tSystem tempEventSystem;

        // Returns the main
 instance
        public static LeanT
ouch Instance
        {
            get
            {
                return Inst
ances.Count > 0 ? Instances
[0] : null;
            }
        }


        public static float
 ScalingFactor
        {
            get
            {
                // Get the
current screen DPI
                var dpi = S
creen.dpi;

                if (dpi <=
0)
                {
                    return
1.0f;
                }


                return Math
f.Sqrt(CurrentReferenceDpi)
 / Mathf.Sqrt(dpi);
            }
        }

        public static bool
AnyMouseButtonSet
        {
            get
            {
                for (var i
= 0; i < highestMouseButton
; i++)
                {
                    if (Inp
ut.GetMouseButton(i) == tru
e)
                    {
                        ret
urn true;
                    }
                }

                return fals
e;
            }
        }

        public static bool
GuiInUse
        {
            get
            {
                // Legacy G
UI in use?
                if (GUIUtil
ity.hotControl > 0)
```

```
                {
                    return
true;
                }

                // New GUI
in use?
                for (var i
= Fingers.Count -
 1; i >= 0; i--)
                {
                    if (Fin
gers[i].StartedOverGui == t
rue)
                    {
                        ret
urn true;
                    }
                }

                return fals
e;
            }
        }


        public static bool
GetCamera(ref Camera camera
, GameObject gameObject = n
ull)
        {
            if (camera == n
ull)
            {
                if (gameObj
ect != null)
                {
                    camera
= gameObject.GetComponent<C
amera>();
                }

                if (camera
== null)
                {
                    camera
= Camera.main;
                }
            }

            return camera !
= null;
        }

        // Return the frame
rate independant damping fa
ctor
        public static float
 GetDampenFactor(float damp
ening, float deltaTime)
        {
            if (Application
.isPlaying == false)
            {
                return 1.0f
;
            }

            return 1.0f -
 Mathf.Exp(-
dampening * deltaTime);
        }


        public static bool
PointOverGui(Vector2 screen
Position)
        {
            return RaycastG
ui(screenPosition).Count >
0;
        }


        public static List<
RaycastResult> RaycastGui(V
ector2 screenPosition)
        {
            return RaycastG
ui(screenPosition, CurrentG
```

```
uiLayers);
        }


        public static List<RaycastResult> RaycastGui(Vector2 screenPosition, LayerMask layerMask)
        {
            tempRaycastResults.Clear();

            var currentEventSystem = EventSystem.current;

            if (currentEventSystem != null)
            {
                // Create point event data for this event system?
                if (currentEventSystem != tempEventSystem)
                {
                    tempEventSystem = currentEventSystem;

                    if (tempPointerEventData == null)
                    {
                        tempPointerEventData = new PointerEventData(tempEventSystem);
                    }
                    else
                    {
                        tempPointerEventData.Reset();
                    }
                }

                // Raycast event system at the specified point
                tempPointerEventData.position = screenPosition;

                currentEventSystem.RaycastAll(tempPointerEventData, tempRaycastResults);


                if (tempRaycastResults.Count > 0)
                {
                    for (var i = tempRaycastResults.Count - 1; i >= 0; i--)
                    {
                        var raycastResult = tempRaycastResults[i];
                        var raycastLayer  = 1 << raycastResult.gameObject.layer;

                        if ((raycastLayer & layerMask) == 0)
                        {
                            tempRaycastResults.RemoveAt(i);
                        }
                    }
                }
            }

            return tempRaycastResults;
        }


        public static List<
```

```
LeanFinger> GetFingers(bool
 ignoreGuiFingers, int requ
iredFingerCount = 0, LeanSe
lectable requiredSelectable
 = null)
        {
            filteredFingers
.Clear();

            if (requiredSel
ectable != null && required
Selectable.SelectingFinger
!= null)
            {
                filteredFin
gers.Add(requiredSelectable
.SelectingFinger);

                return filt
eredFingers;
            }

            for (var i = 0;
 i < Fingers.Count; i++)
            {
                var finger
= Fingers[i];

                if (ignoreG
uiFingers == true)
                {
                    if (fin
ger.StartedOverGui == false
)
                    {
                        fil
teredFingers.Add(finger);
                    }
                }
                else
                {
                    filtere
dFingers.Add(finger);
                }
            }

            if (requiredFin
gerCount > 0)
            {
                if (filtere
dFingers.Count != requiredF
ingerCount)
                {
                    return
null;
                }
            }

            return filtered
Fingers;
        }

        protected virtual v
oid Awake()
        {
#if UNITY_EDITOR
            // Set the fing
er texture?
            if (FingerTextu
re == null)
            {
                var guids =
 UnityEditor.AssetDatabase.
FindAssets("FingerVisualiza
tion t:texture2d");

                if (guids.L
ength > 0)
                {
                    var pat
h = UnityEditor.AssetDataba
se.GUIDToAssetPath(guids[0]
);

                    FingerT
exture = UnityEditor.AssetD
atabase.LoadMainAssetAtPath
(path) as Texture2D;
                }
            }
```

```
#endif
        }

        protected virtual void OnEnable()
        {
            Instances.Add(this);
        }

        protected virtual void OnDisable()
        {
            Instances.Remove(this);
        }

        protected virtual void Update()
        {

            if (Instances[0] == this)
            {
                // Prepare old finger data for new information
                BeginFingers();

                // Poll current finger + mouse data and conver it to fingers
                PollFingers();


                EndFingers();


                UpdateEvents();
            }
        }

        protected virtual void OnGUI()
        {

            if (FingerTexture != null && Input.touchCount == 0 && Fingers.Count > 1)
            {
                for (var i = Fingers.Count - 1; i >= 0; i--)
                {
                    var finger = Fingers[i];


                    if (finger.Up == false)
                    {
                        var screenPosition = finger.ScreenPosition;
                        var screenRect     = new Rect(0, 0, FingerTexture.width, FingerTexture.height);

                        screenRect.center = new Vector2(screenPosition.x, Screen.height - screenPosition.y);

                        GUI.DrawTexture(screenRect, FingerTexture);
                    }
                }
            }
        }


        private void BeginFingers()
```

```
        {

            for (var i = InactiveFingers.Count - 1; i >= 0; i--)
            {
                InactiveFingers[i].Age += Time.unscaledDeltaTime;
            }


            for (var i = Fingers.Count - 1; i >= 0; i--)
            {
                var finger = Fingers[i];


                if (finger.Up == true)
                {
                   Fingers.RemoveAt(i); InactiveFingers.Add(finger);

                    finger.Age = 0.0f;


                    finger.ClearSnapshots();
                }
                else
                {
                    finger.LastSet            = finger.Set;
                    finger.LastScreenPosition = finger.ScreenPosition;

                    finger.Set     = false;
                    finger.Tap     = false;
                }
            }
        }


        private void EndFingers()
        {
            for (var i = Fingers.Count - 1; i >= 0; i--)
            {
                var finger = Fingers[i];


                if (finger.Up == true)
                {
                    // Tap?
                    if (finger.Age <= TapThreshold)
                    {
                        if (finger.SwipeScreenDelta.magnitude * ScalingFactor < SwipeThreshold)
                        {
                            finger.Tap       = true;
                            finger.TapCount += 1;
                        }
                        else
                        {
                            finger.TapCount = 0;
                            finger.Swipe    = true;
                        }
                    }
```

```
                    else
                    {
                        finger.TapCount = 0;
                    }
                }

                else if (finger.Down == false)
                {

                    finger.Age += Time.unscaledDeltaTime;
                }
            }
        }


        private void PollFingers()
        {

            if (Input.touchCount > 0)
            {
                for (var i = 0; i < Input.touchCount; i++)
                {
                    var touch = Input.GetTouch(i);

                    //
                    {
                        AddFinger(touch.fingerId, touch.position);
                    }
                }
            }

            else if (AnyMouseButtonSet == true)
            {
                var screen = new Rect(0, 0, Screen.width, Screen.height);
                var mousePosition = (Vector2)Input.mousePosition;


                if (screen.Contains(mousePosition) == true)
                {
                    AddFinger(0, mousePosition);


                    if (SimulateMultiFingers == true)
                    {
                        if (Input.GetKey(PinchTwistKey) == true)
                        {

 var center = new Vector2(Screen.width * 0.5f, Screen.height * 0.5f);


 AddFinger(1, center - (mousePosition - center));
                        }
                        //
Simulate multi drag?
                        else if (Input.GetKey(MultiDragKey) == true)
                        {

 AddFinger(1, mousePosition);
                        }
                    }
                }
            }
```

```
        }

        private void UpdateEvents()
        {
            var fingerCount = Fingers.Count;

            if (fingerCount > 0)
            {
                for (var i = 0; i < fingerCount; i++)
                {
                    var finger = Fingers[i];

                    if (finger.Down  == true && OnFingerDown  != null) OnFingerDown(finger);
                    if (finger.Set   == true && OnFingerSet   != null) OnFingerSet(finger);
                    if (finger.Up    == true && OnFingerUp    != null) OnFingerUp(finger);
                    if (finger.Tap   == true && OnFingerTap   != null) OnFingerTap(finger);
                    if (finger.Swipe == true && OnFingerSwipe != null) OnFingerSwipe(finger);
                }

                if (OnGesture != null)
                {
                    filteredFingers.Clear();
                    filteredFingers.AddRange(Fingers);

                    OnGesture(filteredFingers);
                }
            }
        }


        private void AddFinger(int index, Vector2 screenPosition)
        {
            var finger = FindFinger(index);

            if (finger == null)
            {
                var inactiveIndex = FindInactiveFingerIndex(index);

                if (inactiveIndex >= 0)
                {
                    finger = InactiveFingers[inactiveIndex]; InactiveFingers.RemoveAt(inactiveIndex);

                    if (finger.Age > TapThreshold)
                    {
                        finger.TapCount = 0;
                    }

                    finger.Age     = 0.0f;
                    finger.
```

```
Set     = false;
                    finger.
LastSet = false;
                    finger.
Tap     = false;
                    finger.
Swipe   = false;
                }

                else
                {
                    finger
= new LeanFinger();

                    finger.
Index = index;
                }

                finger.Star
tScreenPosition = screenPos
ition;
                finger.Last
ScreenPosition  = screenPos
ition;
                finger.Scre
enPosition      = screenPos
ition;
                finger.Star
tedOverGui      = finger.Is
OverGui;

                Fingers.Add
(finger);
            }

            finger.Set
       = true;
            finger.ScreenPo
sition = screenPosition;


            if (RecordFinge
rs == true)
            {
                if (RecordL
imit > 0.0f)
                {
                    if (fin
ger.SnapshotDuration > Reco
rdLimit)
                    {
                        var
 removeCount = LeanSnapshot
.GetLowerIndex(finger.Snaps
hots, finger.Age -
 RecordLimit);

                        fin
ger.ClearSnapshots(removeCo
unt);
                    }
                }

                if (RecordT
hreshold > 0.0f)
                {
                    if (fin
ger.Snapshots.Count == 0 ||
 finger.LastSnapshotScreenD
elta.magnitude >= RecordThr
eshold)
                    {
                        fin
ger.RecordSnapshot();
                    }
                }
                else
                {
                    finger.
RecordSnapshot();
                }
            }
        }


        private LeanFinger
FindFinger(int index)
        {
```

```
            for (var i = Fi
ngers.Count - 1; i>= 0; i--
)
            {
                var finger
= Fingers[i];

                if (finger.
Index == index)
                {
                    return
finger;
                }
            }

            return null;
        }


        private int FindIna
ctiveFingerIndex(int index)
        {
            for (var i = In
activeFingers.Count -
 1; i>= 0; i--)
            {
                if (Inactiv
eFingers[i].Index == index)
                {
                    return
i;
                }
            }

            return -1;
        }
    }
}
```