LAMPIRAN

Lampiran 1

TEKS WAWANCARA

1. Apa yang dimaksud dengan seni tari?

   Jawab : Menurut pendapat Tri Wulandari, yaitu salah satu murid dari sanggar Senin Tari Sangsaka yang juga berprofesi sebagai tenaga pengajar esktrakulikuler menari di salah satu sekolah dasar yang ada di Bandar Lampung, Seni tari merupakan seni yang menggunakan gerakan tubuh secara berirama yang dilakukan ditempat dengan waktu tertentu untuk mengungkapkan perasaan, maksud dan pikiran.

2. Siapa yang dapat dikatakan sebagai penari ?

   Jawab : Tri Wulandari berpendapat seseorang dapat dikatakan sebagai penari apabila telah mengenal dan menguasai serta mempraktekkan hasta sawanda.

3. Apakah hasta sawanda itu?

   Jawab : Hasta Sawanda adalah delapan ketukan normatif yang menjadi satu kesatuan untuk dapat diterapkan bagi seorang penari agar bias membawakan tarian dengan baik.

4. Mengapa seni tari di Indonesia harus dilestarikan?

   Jawab : Tarian di Indonesia perlu dilestarikan agar keberadaan tari di Indonesia tidak punah dan tetap terjaga (ucap seorang murid Sanggar Seni Tari Sangsaka).

5. Kapan anda mulai menari?

   Jawab : Saya mulai menari sejak duduk dikelas 3 SMP.

6. Dimana pertama kali anda menari?

   Jawab : Sebelum bergabung ke dalam sanggar, pertama kali saya mengenal tentang apa itu seni tari, yaitu di sekolah saat saya bergabung kedalam ekstrakulikuler menari.

7. Kendala apa saja yang anda alami saat mempelajari gerak tari?

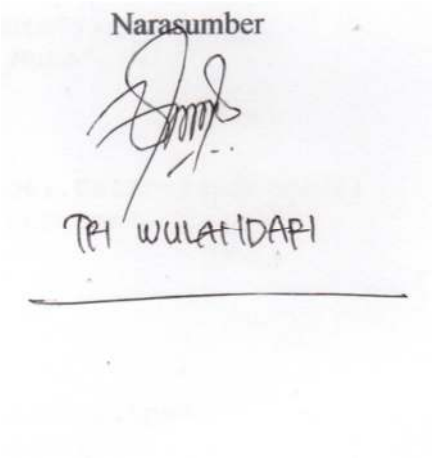
   Jawab : Kendala yang saya alami saat mempelajari gerak tari adalah keselarasan gerak tari dengan irama music,pemahaman detail gerak tari dan kelenturan gerakan tari yang diimplementasikan dengan tubuh.

8. Apakah belajar gerakan tari itu sulit?

   Jawab : Setiap proses pasti memiliki tingkat kesulitannya masing-masing,saat belajar menari dan masih dalam tingkat seorang pemula kesulitan itu pasti akan kita alami.

9. Bagaimana cara menghafal gerak seni tari agar mudah dipahami?

   Jawab : Tri Wulandari berpendapat bahwa 99% kerja keras baginya sebagai seorang penari adalah Latihan dan 1%nya adalah bakat. Gerakan yang sangat banyak akan susah juga untuk dihapalkan meskipun si penari adalah orang yang sangat berbakat, bila ia jarang latihan. Penari yang jarang berlatih meskipun memiliki talenta menari, pada saat diberikan kepercayaan untuk tampil di panggung , semuanya melakukan kesalahan dan lupa banyak gerakan. Jadi, kerja keras dan latihan itu sangat penting.



Lampiran 2

CODING MENU

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

public class Menu : MonoBehaviour {

     [Tooltip("_sceneToLoadOnPlay is the name of the scene
that will be loaded when users click play")]
     public string _sceneToLoadOnPlay = "skrp";
     [Tooltip("_webpageURL defines the URL that will be
opened when users click on your branding icon")]
     public string _webpageURL =
"http://www.instagram.com/yudhisbowo";
     [Tooltip("_soundButtons define the SoundOn[0] and
SoundOff[1] Button objects.")]
     public Button[] _soundButtons;
     [Tooltip("_audioClip defines the audio to be played on
button click.")]
     public AudioClip _audioClip;
     [Tooltip("_audioSource defines the Audio Source
component in this scene.")]
     public AudioSource _audioSource;

     UnityEngine.SceneManagement.Scene scene;

     void Awake () {
          if(!PlayerPrefs.HasKey("_Mute")){
               PlayerPrefs.SetInt("_Mute", 0);
          }

          scene =
UnityEngine.SceneManagement.SceneManager.GetActiveScene();
          PlayerPrefs.SetString("_LastScene",
scene.name.ToString());

     }

     public void OpenWebpage () {
          _audioSource.PlayOneShot(_audioClip);
          Application.OpenURL(_webpageURL);
     }

     public void PlayGame () {
          _audioSource.PlayOneShot(_audioClip);
          PlayerPrefs.SetString("_LastScene", scene.name);

     UnityEngine.SceneManagement.SceneManager.LoadScene(_scen
eToLoadOnPlay);
```

```
	}

	public void Mute () {
		_audioSource.PlayOneShot(_audioClip);
		_soundButtons[0].interactable = true;
		_soundButtons[1].interactable = false;
		PlayerPrefs.SetInt("_Mute", 1);
	}

	public void Unmute () {
		_audioSource.PlayOneShot(_audioClip);
		_soundButtons[0].interactable = false;
		_soundButtons[1].interactable = true;
		PlayerPrefs.SetInt("_Mute", 0);
	}

	public void QuitGame () {
		_audioSource.PlayOneShot(_audioClip);
		#if !UNITY_EDITOR
		Application.Quit();
	}
}
```

Lampiran 3

CODING MUSIK

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

[RequireComponent(typeof(AudioSource))]
public class Music : MonoBehaviour {
      [Tooltip("_audioSource defines the Audio Source
component in this scene.")]
      AudioSource _audioSource;
      [Tooltip("_audioTracks defines the audio clips to be
played continuously through out the scene.")]
      public AudioClip[] _audioTracks;
      [Space(20)]
      [HeaderAttribute("Music Player Options")]
      [Tooltip("_playTracks acts as the Play/Stop function of
the Music Player")]
      public bool _playTracks;
      [Tooltip("Skips to the next available _audioTracks
clip.")]
      public bool _nextTrack;
      [Tooltip("Skips to the previous _audioTracks clip")]
      public bool _prevTrack;
      [Tooltip("Loops the current _audioTracks clip.")]
      public bool _loopTrack;
      [Space(20)]
      [HeaderAttribute("Debugging/ReadOnly")]
      [Tooltip("_playTracks is a ReadOnly variable that
displays the current _audioTracks clip that is playing")]
      public int _playingTrack;
      [Tooltip("_isMute returns the status of muting function
in the Sound_Controller.")]
      public bool _isMute = false;

      void Awake () {
            _audioSource = GetComponent<AudioSource>();
            _audioSource.clip = _audioTracks[0];
            _playingTrack = 0;
            if(PlayerPrefs.HasKey("_Mute")){
                  int _value = PlayerPrefs.GetInt("_Mute");
                  if(_value == 0){_isMute = false;}
                  if(_value == 1){_isMute = true;}
            } else {
                  _isMute = false;
                  PlayerPrefs.SetInt("_Mute", 0);
            }
            if( _isMute ){ _audioSource.mute = true; } else {
_audioSource.mute = false; _audioSource.Play();}
      }
```

```
      void Update () {
            if(!_playTracks) _audioSource.Stop();
            if(_playTracks && !_audioSource.isPlaying)
StartPlayer();
            if(_loopTrack){ _audioSource.loop = true; } else {
_audioSource.loop = false; }
            if(_nextTrack){ NextTrack(); }
            if(_prevTrack){ PreviousTrack(); }

            int _value = PlayerPrefs.GetInt("_Mute");
            if(_value == 0){_isMute = false;}
            if(_value == 1){_isMute = true;}
            if( _isMute ){ _audioSource.mute = true; } else {
_audioSource.mute = false; }
      }

      public void StartPlayer(){
            if(!_loopTrack) { NextTrack();
            } else { _audioSource.Play();
            }
      }

      public void NextTrack(){
            _nextTrack = false;
            _audioSource.Stop();
            int _newCount = _playingTrack+1;
            if(_newCount > _audioTracks.Length-1) {
_audioSource.clip = _audioTracks[0]; _playingTrack = 0;
            } else {
                  _audioSource.clip =
_audioTracks[_newCount];_playingTrack = _newCount;
            }
            _audioSource.Play();
            Debug.Log("Called NextTrack: _next="+_newCount+" :
_playing="+_playingTrack +" : _name=
"+_audioTracks[_playingTrack].name);

      }

      public void PreviousTrack(){
            _prevTrack = false;
            _audioSource.Stop();
            int _newCount = _playingTrack-1;
            if(_newCount < 0) { _audioSource.clip =
_audioTracks[_audioTracks.Length-1];
                  _playingTrack = _audioTracks.Length-1;
            } else {
                  _audioSource.clip = _audioTracks[_newCount];
                  _playingTrack = _newCount;
            }
            _audioSource.Play();
```

```
            Debug.Log("Called PreviousTrack:
_next="+_newCount+" : _playing="+_playingTrack+" : _name=
"+_audioTracks[_playingTrack].name);
      }

}
```

Lampiran 4

CODING PAUSE

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

public class pauseObj : MonoBehaviour {

      public Slider sliderScrubber;
      public Slider speedSlider;
      public Animator animator;

      private float rememberTheSpeedBecauseWeMightNeedIt;

      public void Update()
      {
            float animationTime =
animator.GetCurrentAnimatorStateInfo(0).normalizedTime;
            Debug.Log("animationTime (normalized) is " +
animationTime);
            sliderScrubber.value = animationTime;
      }

      public void ScrubSliderChanged(float
ScrubSliderchangedValue)
      {
            animator.Play("Take 001", -1,
sliderScrubber.normalizedValue);
      }

      public void SpeedSliderChanged(float
SpeedSliderchangedValue)
      {
            animator.speed = speedSlider.normalizedValue;
      }

      public void UserClickedPauseButton()
      {
            if (animator.speed > 0f)
            {
                  rememberTheSpeedBecauseWeMightNeedIt =
animator.speed;
                  animator.speed = 0f;
            }
            else
            {
                  animator.speed =
rememberTheSpeedBecauseWeMightNeedIt;
            }
      }
}
```

Lampiran 5

CODING LEAN TOUCH

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

#if UNITY_EDITOR
using UnityEditor;

namespace Lean.Touch
{
	[CustomEditor(typeof(LeanTouch))]
	public class LeanTouch_Editor : Editor
	{
		private static List<LeanFinger> allFingers = new
List<LeanFinger>();

		private static GUIStyle fadingLabel;

		[MenuItem("GameObject/Lean/Touch", false, 1)]
		public static void CreateTouch()
		{
			var gameObject = new
GameObject(typeof(LeanTouch).Name);

			Undo.RegisterCreatedObjectUndo(gameObject,
"Create Touch");

			gameObject.AddComponent<LeanTouch>();

			Selection.activeGameObject = gameObject;
		}

		public override void OnInspectorGUI()
		{
			if (LeanTouch.Instances.Count > 1)
			{
				EditorGUILayout.HelpBox("There is more
than one active and enabled LeanTouch...",
MessageType.Warning);

				EditorGUILayout.Separator();
			}

			var touch = (LeanTouch)target;

			EditorGUILayout.Separator();

			DrawSettings(touch);

			EditorGUILayout.Separator();
```

```
                DrawFingers(touch);

                EditorGUILayout.Separator();

                Repaint();
            }

            private void DrawSettings(LeanTouch touch)
            {
                DrawDefault("TapThreshold");
                DrawDefault("SwipeThreshold");
                DrawDefault("ReferenceDpi");
                DrawDefault("GuiLayers");

                EditorGUILayout.Separator();

                DrawDefault("RecordFingers");

                if (touch.RecordFingers == true)
                {
                    EditorGUI.indentLevel++;
                        DrawDefault("RecordThreshold");
                        DrawDefault("RecordLimit");
                    EditorGUI.indentLevel--;
                }

                EditorGUILayout.Separator();

                DrawDefault("SimulateMultiFingers");

                if (touch.SimulateMultiFingers == true)
                {
                    EditorGUI.indentLevel++;
                        DrawDefault("PinchTwistKey");
                        DrawDefault("MovePivotKey");
                        DrawDefault("MultiDragKey");
                        DrawDefault("FingerTexture");
                    EditorGUI.indentLevel--;
                }
            }

            private void DrawFingers(LeanTouch touch)
            {
                EditorGUILayout.LabelField("Fingers",
EditorStyles.boldLabel);

                allFingers.Clear();
                allFingers.AddRange(LeanTouch.Fingers);

     allFingers.AddRange(LeanTouch.InactiveFingers);
                allFingers.Sort((a, b) =>
a.Index.CompareTo(b.Index));
```

```
                for (var i = 0; i < allFingers.Count; i++)
                {
                      var finger   = allFingers[i];
                      var progress = touch.TapThreshold > 0.0f
? finger.Age / touch.TapThreshold : 0.0f;
                      var style    =
GetFadingLabel(finger.Set, progress);

                      if (style.normal.textColor.a > 0.0f)
                      {
                            var screenPosition =
finger.ScreenPosition;

                            EditorGUILayout.LabelField("#" +
finger.Index + " x " + finger.TapCount + " (" +
Mathf.FloorToInt(screenPosition.x) + ", " +
Mathf.FloorToInt(screenPosition.y) + ") - " +
finger.Age.ToString("0.0"), style);
                      }
                }
          }

          private void DrawDefault(string name)
          {
                EditorGUI.BeginChangeCheck();


     EditorGUILayout.PropertyField(serializedObject.FindPrope
rty(name));

                if (EditorGUI.EndChangeCheck() == true)
                {

     serializedObject.ApplyModifiedProperties();
                }
          }

          private static GUIStyle GetFadingLabel(bool active,
float progress)
          {
                if (fadingLabel == null)
                {
                      fadingLabel = new
GUIStyle(EditorStyles.label);
                }

                var a = EditorStyles.label.normal.textColor;
                var b = a; b.a = active == true ? 0.5f :
0.0f;

                fadingLabel.normal.textColor = Color.Lerp(a,
b, progress);
```

```
                return fadingLabel;
            }
        }
}
#endif

namespace Lean.Touch
{
    /// <summary>If you add this component to your scene, then it will convert all mouse and touch data into easy to use data.
    /// You can access this data via Lean.Touch.LeanTouch.Instance.Fingers, or hook into the Lean.Touch.LeanTouch.On___ events.
    /// NOTE: If you experience a one frame input delay you should edit your ScriptExecutionOrder to force this script to update before your other scripts.</summary>
    [ExecuteInEditMode]
    [DisallowMultipleComponent]
    [HelpURL(HelpUrlPrefix + "LeanTouch")]
    public partial class LeanTouch : MonoBehaviour
    {
        public const string HelpUrlPrefix = "http://carloswilkes.github.io/Documentation/LeanTouch#";

        public const string PlusHelpUrlPrefix = "http://carloswilkes.github.io/Documentation/LeanTouchPlus#";

        /// <summary>This contains all the active and enabled LeanTouch instances</summary>
        public static List<LeanTouch> Instances = new List<LeanTouch>();

        /// <summary>This list contains all currently active fingers (including simulated ones)</summary>
        public static List<LeanFinger> Fingers = new List<LeanFinger>(10);

        /// <summary>This list contains all currently inactive fingers (this allows for pooling and tapping)</summary>
        public static List<LeanFinger> InactiveFingers = new List<LeanFinger>(10);

        /// <summary>This gets fired when a finger begins touching the screen (LeanFinger = The current finger)</summary>
        public static System.Action<LeanFinger> OnFingerDown;

        /// <summary>This gets fired every frame a finger is touching the screen (LeanFinger = The current finger)</summary>
```

```
		public static System.Action<LeanFinger>
OnFingerSet;

		/// <summary>This gets fired when a finger stops
touching the screen (LeanFinger = The current
finger)</summary>
		public static System.Action<LeanFinger> OnFingerUp;

		/// <summary>This gets fired when a finger taps the
screen (this is when a finger begins and stops touching the
screen within the 'TapThreshold' time) (LeanFinger = The
current finger)</summary>
		public static System.Action<LeanFinger>
OnFingerTap;

		/// <summary>This gets fired when a finger swipes
the screen (this is when a finger begins and stops touching
the screen within the 'TapThreshold' time, and also moves more
than the 'SwipeThreshold' distance) (LeanFinger = The current
finger)</summary>
		public static System.Action<LeanFinger>
OnFingerSwipe;

		/// <summary>This gets fired every frame at least
one finger is touching the screen (List = Fingers).</summary>
		public static System.Action<List<LeanFinger>>
OnGesture;

		/// <summary>This gets fired after a finger has
stopped touching the screen for more than TapThreshold
seconds, and is removed from both the active and inactive
finger lists.</summary>
		public static System.Action<LeanFinger>
OnFingerExpired;

		/// <summary>This allows you to set how many
seconds are required between a finger down/up for a tap to be
registered.</summary>
		[Tooltip("This allows you to set how many seconds
are required between a finger down/up for a tap to be
registered.")]
		public float TapThreshold = DefaultTapThreshold;

		public const float DefaultTapThreshold = 0.2f;

		public static float CurrentTapThreshold
		{
{
			get
			{
				return Instances.Count > 0 ?
Instances[0].TapThreshold : DefaultTapThreshold;
			}
```

```
		}

		/// <summary>This allows you to set how many pixels of movement (relative to the ReferenceDpi) are required within the TapThreshold for a swipe to be triggered.</summary>
		[Tooltip("This allows you to set how many pixels of movement (relative to the ReferenceDpi) are required within the TapThreshold for a swipe to be triggered.")]
		public float SwipeThreshold = DefaultSwipeThreshold;

		public const float DefaultSwipeThreshold = 100.0f;

		public static float CurrentSwipeThreshold
		{
			get
			{
				return Instances.Count > 0 ? Instances[0].SwipeThreshold : DefaultSwipeThreshold;
			}
		}

		/// <summary>This allows you to set the default DPI you want the input scaling to be based on.</summary>
		[Tooltip("This allows you to set the default DPI you want the input scaling to be based on.")]
		public int ReferenceDpi = DefaultReferenceDpi;

		public const int DefaultReferenceDpi = 200;

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

		/// <summary>This allows you to set which layers your GUI is on, so it can be ignored by each finger.</summary>
		[Tooltip("This allows you to set which layers your GUI is on, so it can be ignored by each finger.")]
		public LayerMask GuiLayers = Physics.DefaultRaycastLayers;

		public static LayerMask CurrentGuiLayers
		{
			get
			{
```

```
                    return Instances.Count > 0 ?
Instances[0].GuiLayers :
(LayerMask)Physics.DefaultRaycastLayers;
                }
            }

            /// <summary>Should each fnger record snapshots of
their screen positions?</summary>
            [Tooltip("Should each fnger record snapshots of
their screen positions?")]
            public bool RecordFingers = true;

            /// <summary>This allows you to set the amount of
pixels a finger must move for another snapshot to be
stored.</summary>
            [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;

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

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

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

            /// <summary>This allows you to set which key is
required to change the pivot point of the pinch twist
gesture.</summary>
            [Tooltip("This allows you to set which key is
required to change the pivot point of the pinch twist
gesture.")]
            public KeyCode MovePivotKey = KeyCode.LeftAlt;

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

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

		private static int highestMouseButton = 7;

		private static Vector2 pivot = new Vector2(0.5f,
0.5f);

		private static List<RaycastResult>
tempRaycastResults = new List<RaycastResult>(10);

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

		private static PointerEventData
tempPointerEventData;

		private static EventSystem tempEventSystem;

		/// <summary>The first active and enabled LeanTouch
instance.</summary>
		public static LeanTouch Instance
		{
			get
			{
				return Instances.Count > 0 ?
Instances[0] : null;
			}
		}

		/// <summary>If you multiply this value with any
other pixel delta (e.g. ScreenDelta), then it will become
device resolution independant.</summary>
		public static float ScalingFactor
		{
			get
			{
				var dpi = Screen.dpi;

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

				return CurrentReferenceDpi / dpi;
			}
		}

		{
```

```
                get
                {
                        for (var i = 0; i < highestMouseButton;
i++)
                        {
                                if (Input.GetMouseButton(i) ==
true)
                                {
                                        return true;
                                }
                        }

                        return false;
                }
        }

        /// <summary>This will return true if the mouse or
any finger is currently using the GUI.</summary>
        public static bool GuiInUse
        {
                get
                {
                        if (GUIUtility.hotControl > 0)
                        {
                                return true;
                        }

                        for (var i = Fingers.Count - 1; i >= 0;
i--)
                        {
                                if (Fingers[i].StartedOverGui ==
true)
                                {
                                        return true;
                                }
                        }

                        return false;
                }
        }

        public static Camera GetCamera(Camera
currentCamera, GameObject gameObject = null)
        {
                if (currentCamera == null)
                {
                        if (gameObject != null)
                        {
                                currentCamera =
gameObject.GetComponent<Camera>();
                        }

                        if (currentCamera == null)
```

```
                    {
                        currentCamera = Camera.main;
                    }
                }

                return currentCamera;
            }

            public static float GetDampenFactor(float
dampening, float deltaTime)
            {
                if (dampening < 0.0f)
                {
                    return 1.0f;
                }

                if (Application.isPlaying == false)
                {
                    return 1.0f;
                }

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

            /// <summary>This will return true if the specified
screen point is over any GUI elements.</summary>
            public static bool PointOverGui(Vector2
screenPosition)
            {
                return RaycastGui(screenPosition).Count > 0;
            }

            /// <summary>This will return all the
RaycastResults under the specified screen point using the
current layerMask.
            /// NOTE: The first result (0) will be the top UI
element that was first hit.</summary>
            public static List<RaycastResult>
RaycastGui(Vector2 screenPosition)
            {
                return RaycastGui(screenPosition,
CurrentGuiLayers);
            }

            /// <summary>This will return all the
RaycastResults under the specified screen point using the
specified layerMask.
            /// NOTE: The first result (0) will be the top UI
element that was first hit.</summary>
            public static List<RaycastResult>
RaycastGui(Vector2 screenPosition, LayerMask layerMask)
            {
```

```
                tempRaycastResults.Clear();

                var currentEventSystem = EventSystem.current;

                if (currentEventSystem != null)
                {
                      if (currentEventSystem !=
tempEventSystem)
                      {
                            tempEventSystem =
currentEventSystem;

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

                      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);
                                  }
                            }
                      }
                }
                else
                {
```

```
                    Debug.LogError("Failed to RaycastGui
because your scene doesn't have an event system! To add one,
go to: GameObject/UI/EventSystem");
                }

                return tempRaycastResults;
          }

          /// <summary>This allows you to filter all the
fingers based on the specified requirements.
          /// NOTE: If ignoreGuiFingers is set, Fingers will
be filtered to remove any with StartedOverGui.
          /// NOTE: If requiredFingerCount is greather than
0, this method will return null if the finger count doesn't
match.
          /// NOTE: If requiredSelectable is set, and its
SelectingFinger isn't null, it will return just that
finger.</summary>
          public static List<LeanFinger> GetFingers(bool
ignoreIfStartedOverGui, bool ignoreIfOverGui, int
requiredFingerCount = 0)
          {
                filteredFingers.Clear();

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

                      if (ignoreIfStartedOverGui == true &&
finger.StartedOverGui == true)
                      {
                            continue;
                      }

                      if (ignoreIfOverGui == true &&
finger.IsOverGui == true)
                      {
                            continue;
                      }

                      filteredFingers.Add(finger);
                }

                if (requiredFingerCount > 0)
                {
                      if (filteredFingers.Count !=
requiredFingerCount)
                      {
                            filteredFingers.Clear();

                            return filteredFingers;
                      }
                }
```

```
                return filteredFingers;
          }

          protected virtual void Awake()
          {
#if UNITY_EDITOR
                if (FingerTexture == null)
                {
                      var guids =
AssetDatabase.FindAssets("FingerVisualization t:texture2d");

                      if (guids.Length > 0)
                      {
                            var path =
AssetDatabase.GUIDToAssetPath(guids[0]);

                            FingerTexture =
AssetDatabase.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)
                {
                      BeginFingers();

                      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--)
                  {
                        var inactiveFinger = InactiveFingers[i];

                        inactiveFinger.Age +=
Time.unscaledDeltaTime;

                        if (inactiveFinger.Expired == false &&
inactiveFinger.Age > TapThreshold)
                        {
                              inactiveFinger.Expired = true;

                              if (OnFingerExpired != null)
OnFingerExpired(inactiveFinger);
                        }
                  }

                  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.LastPressure       =
finger.Pressure;
                        finger.LastScreenPosition =
finger.ScreenPosition;

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

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

                    if (finger.Up == true)
                    {
                        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);

                              if (touch.phase ==
TouchPhase.Began || touch.phase == TouchPhase.Stationary ||
touch.phase == TouchPhase.Moved)
                              {
                                    var pressure = 1.0f;
#if UNITY_5_4_OR_NEWER
                                    pressure = touch.pressure;
#endif
                                    AddFinger(touch.fingerId,
touch.position, pressure);
                              }
                        }
                  }
                  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(-1, mousePosition,
1.0f);

                              if (SimulateMultiFingers == true)
                              {
                                    //var finger0 =
FindFinger(0);

                                    if
(Input.GetKey(MovePivotKey) == true)
                                    {
                                          pivot.x =
mousePosition.x / Screen.width;
                                          pivot.y =
mousePosition.y / Screen.height;
                                    }

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

```
                                    var center = new
Vector2(Screen.width * pivot.x, Screen.height * pivot.y);

                                    AddFinger(-2, center -
(mousePosition - center), 1.0f);
                                    //AddFinger(-2,
finger0.StartScreenPosition - finger0.SwipeScreenDelta, 1.0f);
                                }
                                else if
(Input.GetKey(MultiDragKey) == true)
                                {
                                    AddFinger(-2,
mousePosition, 1.0f);
                                }
                            }
                        }
                    }
                }

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

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

                            if (finger.Tap   == true &&
OnFingerTap   != null) OnFingerTap(finger);
                            if (finger.Swipe == true &&
OnFingerSwipe != null) OnFingerSwipe(finger);
                            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 (OnGesture != null)
                        {
                            filteredFingers.Clear();
                            filteredFingers.AddRange(Fingers);

                            OnGesture(filteredFingers);
                        }
                    }
                }

                private void AddFinger(int index, Vector2
screenPosition, float pressure)
```

```
            {
                  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;
                              finger.Expired = false;
                        }
                        else
                        {
                              finger = new LeanFinger();

                              finger.Index = index;
                        }

                        finger.StartScreenPosition =
screenPosition;
                        finger.LastScreenPosition  =
screenPosition;
                        finger.LastPressure        = pressure;
                        finger.StartedOverGui      =
PointOverGui(screenPosition);

                        Fingers.Add(finger);
                  }

                  finger.Set            = true;
                  finger.ScreenPosition = screenPosition;
                  finger.Pressure       = pressure;

                  if (RecordFingers == true)
                  {
                        if (RecordLimit > 0.0f)
                        {
```

```
                              if (finger.SnapshotDuration >
RecordLimit)
                              {
                                    var removeCount =
LeanSnapshot.GetLowerIndex(finger.Snapshots, finger.Age -
RecordLimit);

      finger.ClearSnapshots(removeCount);
                              }
                        }

                        if (RecordThreshold > 0.0f)
                        {
                              if (finger.Snapshots.Count == 0 ||
finger.LastSnapshotScreenDelta.magnitude >= RecordThreshold)
                              {
                                    finger.RecordSnapshot();
                              }
                        }
                        else
                        {
                              finger.RecordSnapshot();
                        }
                  }
            }

            private LeanFinger FindFinger(int index)
            {
                  for (var i = Fingers.Count - 1; i>= 0; i--)
                  {
                        var finger = Fingers[i];

                        if (finger.Index == index)
                        {
                              return finger;
                        }
                  }
                  return null;
            }
            private int FindInactiveFingerIndex(int index)
            {
                  for (var i = InactiveFingers.Count - 1; i>=
0; i--)
                  {
                        if (InactiveFingers[i].Index == index)
                        {
                              return i;
                        }
                  }
                  return -1;
            }
      }
}
```

Lampiran 6

CODING LEAN TRANSLATE

```
using UnityEngine;

namespace Lean.Touch
{
	/// <summary>This script allows you to translate the
current GameObject relative to the camera.</summary>
	[HelpURL(LeanTouch.HelpUrlPrefix + "LeanTranslate")]
	public class LeanTranslate : MonoBehaviour
	{
		[Tooltip("Ignore fingers with StartedOverGui?")]
		public bool IgnoreStartedOverGui = true;

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

		[Tooltip("Ignore fingers if the finger count
doesn't match? (0 = any)")]
		public int RequiredFingerCount;

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

		[Tooltip("The camera the translation will be
calculated using (None = MainCamera)")]
		public Camera Camera;

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

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

		protected virtual void Update()
		{
			var fingers =
LeanSelectable.GetFingers(IgnoreStartedOverGui,
IgnoreIsOverGui, RequiredFingerCount, RequiredSelectable);
```

```
                var screenDelta =
LeanGesture.GetScreenDelta(fingers);

                if (screenDelta != Vector2.zero)
                {
                      if (transform is RectTransform)
                      {
                            TranslateUI(screenDelta);
                      }
                      else
                      {
                            Translate(screenDelta);
                      }
                }
          }

          protected virtual void TranslateUI(Vector2
screenDelta)
          {
                var screenPoint =
RectTransformUtility.WorldToScreenPoint(Camera,
transform.position);

                screenPoint += screenDelta;

                var worldPoint = default(Vector3);

                if
(RectTransformUtility.ScreenPointToWorldPointInRectangle(trans
form.parent as RectTransform, screenPoint, Camera, out
worldPoint) == true)
                {
                      transform.position = worldPoint;
                }
          }

          protected virtual void Translate(Vector2
screenDelta)
          {
                var camera = LeanTouch.GetCamera(Camera,
gameObject);

                if (camera != null)
                {
                      var screenPoint =
camera.WorldToScreenPoint(transform.position);

                      screenPoint += (Vector3)screenDelta;

                      transform.position =
camera.ScreenToWorldPoint(screenPoint);
                }
                else
```

```
                {
                        Debug.LogError("Failed to find camera.
Either tag your camera as MainCamera, or set one in this
component.", this);
                }
            }
      }
}
```

Lampiran 7

CODING LEAN ROTATE

```
using UnityEngine;

namespace Lean.Touch
{
	/// <summary>This script allows you to transform the current GameObject.</summary>
	[HelpURL(LeanTouch.HelpUrlPrefix + "LeanRotate")]
	public class LeanRotate : MonoBehaviour
	{
		[Tooltip("Ignore fingers with StartedOverGui?")]
		public bool IgnoreStartedOverGui = true;

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

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

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

		[Tooltip("The camera we will be used to calculate relative rotations (None = MainCamera)")]
		public Camera Camera;

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

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

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

		protected virtual void Update()
		{
```

```
                var fingers =
LeanSelectable.GetFingers(IgnoreStartedOverGui,
IgnoreIsOverGui, RequiredFingerCount, RequiredSelectable);

                var twistDegrees =
LeanGesture.GetTwistDegrees(fingers);

                if (twistDegrees != 0.0f)
                {
                      if (Relative == true)
                      {
                            var twistScreenCenter =
LeanGesture.GetScreenCenter(fingers);

                            if (transform is RectTransform)
                            {
                                  TranslateUI(twistDegrees,
twistScreenCenter);
                                  RotateUI(twistDegrees);
                            }
                            else
                            {
                                  Translate(twistDegrees,
twistScreenCenter);
                                  Rotate(twistDegrees);
                            }
                      }
                      else
                      {
                            if (transform is RectTransform)
                            {
                                  RotateUI(twistDegrees);
                            }
                            else
                            {
                                  Rotate(twistDegrees);
                            }
                      }
                }
          }

          protected virtual void TranslateUI(float
twistDegrees, Vector2 twistScreenCenter)
          {
                var screenPoint =
RectTransformUtility.WorldToScreenPoint(Camera,
transform.position);

                var twistRotation = Quaternion.Euler(0.0f,
0.0f, twistDegrees);
                var screenDelta   = twistRotation *
(screenPoint - twistScreenCenter);
```

```
                screenPoint.x = twistScreenCenter.x +
screenDelta.x;
                screenPoint.y = twistScreenCenter.y +
screenDelta.y;

                var worldPoint = default(Vector3);

                if
(RectTransformUtility.ScreenPointToWorldPointInRectangle(trans
form.parent as RectTransform, screenPoint, Camera, out
worldPoint) == true)
                {
                     transform.position = worldPoint;
                }
           }

           protected virtual void Translate(float
twistDegrees, Vector2 twistScreenCenter)
           {
                var camera = LeanTouch.GetCamera(Camera,
gameObject);

                if (camera != null)
                {
                     var screenPoint =
camera.WorldToScreenPoint(transform.position);

                     var twistRotation =
Quaternion.Euler(0.0f, 0.0f, twistDegrees);
                     var screenDelta   = twistRotation *
((Vector2)screenPoint - twistScreenCenter);

                     screenPoint.x = twistScreenCenter.x +
screenDelta.x;
                     screenPoint.y = twistScreenCenter.y +
screenDelta.y;

                     transform.position =
camera.ScreenToWorldPoint(screenPoint);
                }
                else
                {
                     Debug.LogError("Failed to find camera.
Either tag your cameras MainCamera, or set one in this
component.", this);
                }
           }

           protected virtual void RotateUI(float twistDegrees)
           {
                transform.rotation *= Quaternion.Euler(0.0f,
0.0f, twistDegrees);
           }
```

```
            protected virtual void Rotate(float twistDegrees)
            {
                  var camera = LeanTouch.GetCamera(Camera,
gameObject);

                  if (camera != null)
                  {
                        var axis =
transform.InverseTransformDirection(camera.transform.forward);

                        transform.rotation *=
Quaternion.AngleAxis(twistDegrees, axis);
                  }
                  else
                  {
                        Debug.LogError("Failed to find camera.
Either tag your cameras MainCamera, or set one in this
component.", this);
                  }
            }
      }
}
```

Lampiran 8

CODING LEAN SCALE

```
using UnityEngine;

namespace Lean.Touch
{
	/// <summary>This script allows you to scale the current GameObject.</summary>
	[HelpURL(LeanTouch.HelpUrlPrefix + "LeanScale")]
	public class LeanScale : MonoBehaviour
	{
		[Tooltip("Ignore fingers with StartedOverGui?")]
		public bool IgnoreStartedOverGui = true;

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

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

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

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

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

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

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

		[Tooltip("The minimum scale value on all axes")]
		public Vector3 ScaleMin;

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

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

```
		}
#endif

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

		protected virtual void Update()
		{
			var fingers =
LeanSelectable.GetFingers(IgnoreStartedOverGui,
IgnoreIsOverGui, RequiredFingerCount, RequiredSelectable);

			var pinchScale =
LeanGesture.GetPinchScale(fingers, WheelSensitivity);

			if (pinchScale != 1.0f)
			{
				if (Relative == true)
				{
					var pinchScreenCenter =
LeanGesture.GetScreenCenter(fingers);

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

				Scale(transform.localScale *
pinchScale);
			}
		}

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

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

                var worldPoint = default(Vector3);

                if
(RectTransformUtility.ScreenPointToWorldPointInRectangle(trans
form.parent as RectTransform, screenPoint, Camera, out
worldPoint) == true)
                {
                     transform.position = worldPoint;
                }
           }

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

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

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

                     transform.position =
camera.ScreenToWorldPoint(screenPosition);
                }
                else
                {
                     Debug.LogError("Failed to find camera.
Either tag your cameras MainCamera, or set one in this
component.", this);
                }
           }

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

```
                }

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