LAMPIRAN

Move Camera

```
// Credit to damien_oconnell from http://forum.Unity3d.com/threads/39513-Click-drag-camera-movement
// for using the mouse displacement for calculating the amount of camera movement and panning code.

using UnityEngine;
using System.Collections;

public Class MoveCamera : MonoBehaviour
{
	//
	// VARIABLES
	//

	public float turnSpeed = 4.0f;
	// Speed of camera turning when mouse moves in along an axis
	public float panSpeed = 4.0f;
	// Speed of the camera when being panned
	public float zoomSpeed = 4.0f;		// Speed of the camera going back and forth

	private Vector3 mouseOrigin;
	// Position of cursor when mouse dragging starts
	private bool isPanning;
	// Is the camera being panned?
	private bool isRotating;
	// Is the camera being rotated?
	private bool isZooming;
	// Is the camera zooming?

	//
	// UPDATE
	//

	void Update ()
	{
		// Get the left mouse button

		if(Input.GetMouseButtonDown(0))
		{
			// Get mouse origin
			mouseOrigin = Input.mousePosition;
			isRotating = true;
		}

		// Get the right mouse button

		if(Input.GetMouseButtonDown(1))
		{
			// Get mouse origin
			mouseOrigin = Input.mousePosition;
			isPanning = true;
		}

		// Get the middle mouse button

		if(Input.GetMouseButtonDown(2))
		{
			// Get mouse origin
			mouseOrigin = Input.mousePosition;
			isZooming = true;
		}
```

```
		// Disable movements on button release
		if (!Input.GetMouseButton(0)) isRotating=false;
		if (!Input.GetMouseButton(1)) isPanning=false;
		if (!Input.GetMouseButton(2)) isZooming=false;

		// Rotate camera along X and Y axis
		if (isRotating)
		{
			Vector3 pos = Camera.main.ScreenToViewportPoint(Input.mousePosition - mouseOrigin);


		transform.RotateAround(transform.position, transform.right, -pos.y * turnSpeed);

		transform.RotateAround(transform.position, Vector3.up, pos.x * turnSpeed);
		}

		// Move the camera on it's XY plane
		if (isPanning)
		{
			Vector3 pos = Camera.main.ScreenToViewportPoint(Input.mousePosition - mouseOrigin);

			Vector3 move = new Vector3(pos.x * panSpeed, pos.y * panSpeed, 0);

		transform.Translate(move, Space.Self);
		}

		// Move the camera linearly along Z axis
		if (isZooming)
		{
			Vector3 pos = Camera.main.ScreenToViewportPoint(Input.mousePosition - mouseOrigin);

			Vector3 move = pos.y * zoomSpeed * transform.forward;

		transform.Translate(move, Space.World);
		}
	}
}
```

LeanTweenDocumentationEditor

```
using UnityEngine;
using System.Collections;
#if UNITY_EDITOR
using UnityEditor;

public Class LeanTweenDocumentationEditor : Editor {

	[MenuItem ("Help/LeanTween Documentation")]
	static void openDocumentation()
	{
```

```
		#if !UNITY_3_5 && !UNITY_4_0 && !UNITY_4_0_1 && !UNITY_4_1 && !UNITY_4_2 && !UNITY_4_3
		// Loops through all items in case the user has moved the default installation directory
		string[] guids = AssetDatabase.FindAssets ("LeanTween", null);
		string documentationPath = "";
		foreach (string guid in guids){
			string path = AssetDatabase.GUIDToAssetPath(guid);

	if(path.IndexOf("classes/LeanTween.html")>=0){

	documentationPath = path;
				break;
			}
		}
		documentationPath = documentationPath.Substring(documentationPath.IndexOf("/"));
		string browserPath = "file://" + Application.dataPath + documentationPath + "#index";

	Application.OpenURL(browserPath);

		#else
		// assumes the default installation directory
		string documentationPath = "file://"+Application.dataPath + "/LeanTween/Documentation/classes/LeanTween.html#index";

	Application.OpenURL(documentationPath);
		#endif
	}

	[MenuItem ("Help/LeanTween Forum (ask questions)")]
	static void openForum()
	{

	Application.OpenURL("http://forum.Unity3d.com/threads/leantween-a-tweening-engine-that-is-up-to-5x-faster-than-competing-engines.161113/");
	}

	[MenuItem ("Help/LeanTween GitHub (contribute code)")]
	static void openGit()
	{

	Application.OpenURL("https://github.com/dentedpixel/LeanTween");
	}

	[MenuItem ("Help/LeanTween Support (donate)")]
	static void openLTDonate()
	{

	Application.OpenURL("https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=YJPUT3RAK5VL8");
	}

	[MenuItem ("Help/Dented Pixel News")]
	static void openDPNews()
	{
```

```
        }
    }

    Application.OpenURL("http://dentedpixel.com/category/developer-diary/");
}

#endif
```

OldGUIExamplesCS

```
using UnityEngine;
using System.Collections;
using DentedPixel;

public class OldGUIExamplesCS : MonoBehaviour {
    public Texture2D grumpy;
    public Texture2D beauty;

    private float w;
    private float h;

    private LTRect buttonRect1;
    private LTRect buttonRect2;
    private LTRect buttonRect3;
    private LTRect buttonRect4;
    private LTRect grumpyRect;
    private LTRect beautyTileRect;


    // Use this for initialization
    void Start () {
        w = Screen.width;
        h = Screen.height;
        buttonRect1 = new LTRect(0.10f*w, 0.8f*h, 0.2f*w, 0.14f*h );
        buttonRect2 = new LTRect(1.2f*w, 0.8f*h, 0.2f*w, 0.14f*h );
        buttonRect3 = new LTRect(0.35f*w, 0.0f*h, 0.3f*w, 0.2f*h, 0f );
        buttonRect4 = new LTRect(0.0f*w, 0.4f*h, 0.3f*w, 0.2f*h, 1.0f, 15.0f );

        grumpyRect = new LTRect(0.5f*w - grumpy.width*0.5f, 0.5f*h - grumpy.height*0.5f, grumpy.width, grumpy.height );
        beautyTileRect = new LTRect(0.0f,0.0f,1.0f,1.0f );

        LeanTween.move( buttonRect2, new Vector2(0.55f*w, buttonRect2.rect.y), 0.7f ).setEase(LeanTweenType.easeOutQuad);
    }

    public void catMoved(){
        Debug.Log("cat moved...");
    }

    // Update is called once per frame
    void OnGUI () {
        GUI.DrawTexture( grumpyRect.rect, grumpy);

        Rect staticRect = new Rect(0.0f*w, 0.0f*h, 0.2f*w, 0.14f*h);
        if(GUI.Button( staticRect, "Move Cat")){

        if(LeanTween.isTweening(gr
```

```
umpyRect)==false){ // Check to see if the cat is already tweening, so it doesn't freak out

        Vector2 orig = new Vector2( grumpyRect.rect.x, grumpyRect.rect.y );

        LeanTween.move( grumpyRect, new Vector2( 1.0f*Screen.width - grumpy.width, 0.0f*Screen.height ), 1.0f).setEase(LeanTweenType.easeOutBounce).setOnComplete(catMoved);

        LeanTween.move( grumpyRect, orig, 1.0f ).setDelay(1.0f).setEase( LeanTweenType.easeOutBounce);
                }
        }


        if(GUI.Button(buttonRect1.rect, "Scale Centered")){

        LeanTween.scale( buttonRect1, new Vector2(buttonRect1.rect.width, buttonRect1.rect.height) * 1.2f, 0.25f ).setEase( LeanTweenType.easeOutQuad );

        LeanTween.move( buttonRect1, new Vector2(buttonRect1.rect.x-buttonRect1.rect.width*0.1f, buttonRect1.rect.y-buttonRect1.rect.height*0.1f), 0.25f ).setEase(LeanTweenType.easeOutQuad);
        }


        if(GUI.Button(buttonRect2.rect, "Scale")){

        LeanTween.scale( buttonRect2, new Vector2(buttonRect2.rect.width, buttonRect2.rect.height) * 1.2f, 0.25f ).setEase(LeanTweenType.easeOutBounce);
        }

        staticRect = new Rect(0.76f*w, 0.53f*h, 0.2f*w, 0.14f*h);
        if(GUI.Button( staticRect, "Flip Tile")){

        LeanTween.move( beautyTileRect, new Vector2( 0f, beautyTileRect.rect.y + 1.0f ), 1.0f ).setEase(LeanTweenType.easeOutBounce);
        }


        GUI.DrawTextureWithTexCoords( new Rect(0.8f*w, 0.5f*h - beauty.height*0.5f, beauty.width*0.5f, beauty.height*0.5f), beauty, beautyTileRect.rect);



        if(GUI.Button(buttonRect3.rect, "Alpha")){

        LeanTween.alpha( buttonRect3, 0.0f, 1.0f).setEase(LeanTweenType.easeOutQuad);

        LeanTween.alpha( buttonRect3, 1.0f,
```

```
1.0f).setDelay(1.0f).setEase(
LeanTweenType.easeInQuad);


        LeanTween.alpha(
grumpyRect, 0.0f,
1.0f).setEase(LeanTweenType.easeO
utQuad);

        LeanTween.alpha(
grumpyRect, 1.0f,
1.0f).setDelay(1.0f).setEase(LeanTw
eenType.easeInQuad);
                }
                GUI.color = new
Color(1.0f,1.0f,1.0f,1.0f); // Reset to
normal alpha, otherwise other gui
elements will be effected

        if(GUI.Button(buttonRect4.re
ct, "Rotate")){

        LeanTween.rotate(
buttonRect4, 150.0f, 1.0f
).setEase(LeanTweenType.easeOutE
lastic);

        LeanTween.rotate(
buttonRect4, 0.0f, 1.0f
).setDelay(1.0f).setEase(LeanTween
Type.easeOutElastic);
                }
                GUI.matrix =
Matrix4x4.identity;
        }
}
```

TestingPunch

```
using UnityEngine;
using System.Collections;
using DentedPixel;

public Class TestingPunch :
MonoBehaviour {

   public AnimationCurve
exportCurve;
   public float overShootValue = 1f;

   private LTDescr descr;

        void Start () {

//LeanTween.rotateAround(gameObj
ect,
gameObject.transform.rotation.euler
Angles, 360f,
5f).setDelay(1f).setEase(LeanTween
Type.easeOutBounce);

      Debug.Log( "exported curve:" +
curveToString(exportCurve) );
        }

        void Update ()
   {
      LeanTween.dtManual =
Time.deltaTime;
      if
(Input.GetKeyDown(KeyCode.Q))
      {

//LeanTween.scale(this.gameObject,
Vector3.one*3f,
1.0f).setEase(LeanTweenType.easeS
pring).setUseManualTime(true);
        //print("scale punch time
independent!");


LeanTween.moveLocalX(gameObje
ct, 5, 1).setOnComplete( () => {
```

```
        Debug.Log("on complete
move local X");

}).setOnCompleteOnStart(true);

        GameObject light =
GameObject.Find("DirectionalLight"
);
        Light lt =
light.GetComponent<Light>();


LeanTween.value(lt.gameObject,
lt.intensity, 0.0f, 1.5f)

.setEase(LeanTweenType.linear)
           .setLoopPingPong()
           .setRepeat(-1)
           .setOnUpdate((float val)=>{

        lt.intensity = val;
        });
      }
      if
(Input.GetKeyDown(KeyCode.S))
      {
        print("scale punch!");

        tweenStatically(
this.gameObject );


LeanTween.scale(this.gameObject,
new Vector3(1.15f, 1.15f, 1.15f),
0.6f);


LeanTween.rotateAround(this.game
Object, Vector3.forward, -360f,
0.3f).setOnComplete(() =>
        {

LeanTween.rotateAround(this.game
Object, Vector3.forward, -360f,
0.4f).setOnComplete(() =>
            {

LeanTween.scale(this.gameObject,
new Vector3(1f, 1f, 1f), 0.1f);


LeanTween.value(this.gameObject,
(v) =>
                {

                }, 0, 1,
0.3f).setDelay(1);

            });

          });


      }

      if
(Input.GetKeyDown(KeyCode.T))
      {
        Vector3[] pts = new
Vector3[] {new Vector3(-1f,0f,0f),
new Vector3(0f,0f,0f), new
Vector3(4f,0f,0f), new
Vector3(20f,0f,0f)};
        descr =
LeanTween.move(gameObject, pts,
15f).setOrientToPath(true).setDirecti
on(1f).setOnComplete( ()=>{
          Debug.Log("move path
finished");
        });
      }

      if
(Input.GetKeyDown(KeyCode.Y)) //
Reverse the move path
      {
        descr.setDirection(-
descr.direction);
      }

      if
(Input.GetKeyDown(KeyCode.R))
```

```
    {
        //
LeanTween.rotate(this.gameObject,
Vector3.one,
1.0f).setEase(LeanTweenType.punch
);

LeanTween.rotateAroundLocal(this.
gameObject, this.transform.forward,
-80f, 5.0f).setPoint(new
Vector3(1.25f, 0f, 0f));
        print("rotate punch!");
    }

    if
(Input.GetKeyDown(KeyCode.M))
    {
        //
LeanTween.move(this.gameObject,
new Vector3(0f,0f,1f),
1.0f).setEase(LeanTweenType.punch
);
        print("move punch!");
        Time.timeScale = 0.25f;
        float start =
Time.realtimeSinceStartup;
        LeanTween.moveX(
this.gameObject, 1f,
1f).setOnComplete( destroyOnComp
).setOnCompleteParam(
this.gameObject ).setOnComplete(
()=>{
            float end =
Time.realtimeSinceStartup;
            float diff = end - start;
            Debug.Log("start:"+start+"
end:"+end+" diff:"+diff+"
x:"+this.gameObject.transform.positi
on.x);

}).setEase(LeanTweenType.easeInB
ack).setOvershoot( overShootValue
).setPeriod(0.3f);
    }

    if
(Input.GetKeyDown(KeyCode.C))
    {
        LeanTween.color(
this.gameObject, new Color(1f, 0f,
0f, 0.5f), 1f);

        Color to = new
Color(Random.Range(0f,1f),0f,Rand
om.Range(0f,1f),0.0f);
        GameObject l =
GameObject.Find("LCharacter");
        LeanTween.color( l, to, 4.0f
).setLoopPingPong(1).setEase(Lean
TweenType.easeOutBounce);
    }

    if
(Input.GetKeyDown(KeyCode.E))
    {

LeanTween.delayedCall(gameObject
,0.3f,
delayedMethod).setRepeat(4).setOn
CompleteOnRepeat(true).setOnCom
pleteParam( "hi" );
    }

    if
(Input.GetKeyDown(KeyCode.V))
    {
        LeanTween.value(
gameObject, updateColor, new
Color(1.0f,0.0f,0.0f,1.0f), Color.blue,
4.0f
);//.setRepeat(2).setLoopPingPong().
setEase(LeanTweenType.easeOutBo
unce);
    }

    if
(Input.GetKeyDown(KeyCode.P))
    {

LeanTween.delayedCall(0.05f,
```

```
enterMiniGameStart).setOnComplete
Param( new object[]{""+5} );
        }


if(Input.GetKeyDown(KeyCode.U))
{
          #if !UNITY_FLASH

LeanTween.value(gameObject,
(Vector2 val)=>{
              // Debug.Log("tweening
vec2 val:"+val);
              transform.position = new
Vector3(val.x, transform.position.y,
transform.position.z);
          }, new Vector2(0f,0f), new
Vector2(5f,100f), 1f
).setEase(LeanTweenType.easeOutB
ounce);

          GameObject l =
GameObject.Find("LCharacter");

Debug.Log("x:"+l.transform.position
.x+" y:"+l.transform.position.y);
          LeanTween.value(l, new
Vector2( l.transform.position.x,
l.transform.position.y), new Vector2(
l.transform.position.x,
l.transform.position.y+5), 1f
).setOnUpdate(
          (Vector2 val)=>{
              Debug.Log("tweening
vec2 val:"+val);
               l.transform.position = new
Vector3(val.x, val.y,
transform.position.z);
          }

          );
          #endif
        }
          }
```

```
    static void tweenStatically(
GameObject gameObject ){
        Debug.Log("Starting to
tween...");
        LeanTween.value(gameObject,
(val)=>{
            Debug.Log("tweening
val:"+val);
        }, 0f, 1f, 1f);
    }

    void enterMiniGameStart( object
val ){
        object[] arr = (object [])val;
        int lvl =
int.Parse((string)arr[0]);
        Debug.Log("level:"+lvl);
    }

    void updateColor( Color c ){
        GameObject l =
GameObject.Find("LCharacter");
        // Debug.Log("new col:"+c);

l.GetComponent<Renderer>().materi
al.color = c;
    }

    void delayedMethod( object
myVal ){
        string castBack = myVal as
string;
        Debug.Log("delayed
call:"+Time.time +"
myVal:"+castBack);
    }

    void destroyOnComp( object p ){
      GameObject g = (GameObject)p;
      Destroy( g );
    }

    string curveToString(
AnimationCurve curve) {
        string str = "";
```

```
        for(int i = 0; i < curve.length;
i++){
            str += "new
Keyframe("+curve[i].time+"f,
"+curve[i].value+"f)";
            if(i<curve.length-1)
                str += ", ";
        }
        return "new AnimationCurve(
"+str+" )";
    }
}
```

TestingRigidbodyCS

```
using UnityEngine;
using System.Collections;
using DentedPixel;

public Class TestingRigidbodyCS :
MonoBehaviour {

    private GameObject ball1;
    // Use this for initialization
    void Start () {
        ball1 =
GameObject.Find("Sphere1");


    LeanTween.rotateAround(
ball1, Vector3.forward, -90f, 1.0f);
        LeanTween.move(
ball1, new Vector3(2f,0f,7f),
1.0f).setDelay(1.0f).setRepeat(-1);
    }

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

    }
}
```

Following

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

public Class Following :
MonoBehaviour {

    public TransForm planet;

    public TransForm followArrow;

    public TransForm dude1;
    public TransForm dude2;
    public TransForm dude3;
    public TransForm dude4;
    public TransForm dude5;

    public TransForm dude1Title;
    public TransForm dude2Title;
    public TransForm dude3Title;
    public TransForm dude4Title;
    public TransForm dude5Title;

    private Color dude1ColorVelocity;

    private Vector3 velocityPos;
```

```
    private void Start()
    {

followArrow.gameObject.LeanDelay
edCall(3f,
moveArrow).setOnStart(moveArrow
).setRepeat(-1);

        // Follow Local Y Position of
Arrow

LeanTween.followDamp(dude1,
followArrow, LeanProp.localY,
1.1f);

LeanTween.followSpring(dude2,
followArrow, LeanProp.localY,
1.1f);

LeanTween.followBounceOut(dude3
, followArrow, LeanProp.localY,
1.1f);

LeanTween.followSpring(dude4,
followArrow, LeanProp.localY, 1.1f,
-1f, 1.5f, 0.8f);

LeanTween.followLinear(dude5,
followArrow, LeanProp.localY, 50f);

        // Follow Arrow color

LeanTween.followDamp(dude1,
followArrow, LeanProp.color, 1.1f);

LeanTween.followSpring(dude2,
followArrow, LeanProp.color, 1.1f);

LeanTween.followBounceOut(dude3
, followArrow, LeanProp.color,
1.1f);

LeanTween.followSpring(dude4,
followArrow, LeanProp.color, 1.1f, -
1f, 1.5f, 0.8f);

LeanTween.followLinear(dude5,
followArrow, LeanProp.color, 0.5f);

        // Follow Arrow scale

LeanTween.followDamp(dude1,
followArrow, LeanProp.scale, 1.1f);

LeanTween.followSpring(dude2,
followArrow, LeanProp.scale, 1.1f);

LeanTween.followBounceOut(dude3
, followArrow, LeanProp.scale,
1.1f);

LeanTween.followSpring(dude4,
followArrow, LeanProp.scale, 1.1f, -
1f, 1.5f, 0.8f);

LeanTween.followLinear(dude5,
followArrow, LeanProp.scale, 5f);

        // Titles
        var titleOffset = new
Vector3(0.0f, -20f, -18f);

LeanTween.followDamp(dude1Title,
dude1, LeanProp.localPosition,
0.6f).setOffset(titleOffset);

LeanTween.followSpring(dude2Title
, dude2, LeanProp.localPosition,
0.6f).setOffset(titleOffset);

LeanTween.followBounceOut(dude3
Title, dude3, LeanProp.localPosition,
0.6f).setOffset(titleOffset);

LeanTween.followSpring(dude4Title
, dude4, LeanProp.localPosition,
0.6f, -1f, 1.5f,
0.8f).setOffset(titleOffset);

LeanTween.followLinear(dude5Title
```

```
, dude5, LeanProp.localPosition,
30f).setOffset(titleOffset);

        // Rotate Planet
        var localPos =
Camera.main.transform.InverseTrans
formPoint(planet.transform.position)
;

LeanTween.rotateAround(Camera.m
ain.gameObject, Vector3.left, 360f,
300f).setPoint(localPos).setRepeat(-
1);
    }

    private float fromY;
    private float velocityY;
    private Vector3 fromVec3;
    private Vector3 velocityVec3;
    private Color fromColor;
    private Color velocityColor;

    private void Update()
    {
        // Use the smooth methods to
follow variables in which ever
manner you wish!
        fromY =
LeanSmooth.spring(fromY,
followArrow.localPosition.y, ref
velocityY, 1.1f);
        fromVec3 =
LeanSmooth.spring(fromVec3,
dude5Title.localPosition, ref
velocityVec3, 1.1f);
        fromColor =
LeanSmooth.spring(fromColor,
dude1.GetComponent<Renderer>().
material.color, ref velocityColor,
1.1f);
        Debug.Log("Smoothed y:" +
fromY + " vec3:" + fromVec3 + "
color:" + fromColor);
    }

        private void moveArrow()
    {

LeanTween.moveLocalY(followArro
w.gameObject, Random.Range(-
100f, 100f), 0f);

        var randomCol = new
Color(Random.value, Random.value,
Random.value);

LeanTween.color(followArrow.game
Object, randomCol, 0f);

        var randomVal =
Random.Range(5f, 10f);
        followArrow.localScale =
Vector3.one * randomVal;
    }
}
```

LeanAudioStream

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

public Class LeanAudioStream {

        public int position = 0;

        public AudioClip audioClip;

        public float[] audioArr;

        public LeanAudioStream(
float[] audioArr ){
                this.audioArr =
audioArr;
        }
```

```
        public void
OnAudioRead(float[] data) {
      int count = 0;
      while (count < data.Length) {
         data[count] =
audioArr[this.position];
         position++;
         count++;
      }
   }

   public void
OnAudioSetPosition(int
newPosition) {
         this.position = newPosition;
   }
}

/**
* Create Audio dynamically and
easily playback
*
* @Class LeanAudio
* @constructor
*/
public Class LeanAudio : object {

         public static float
MIN_FREQEUNCY_PERIOD =
0.000115f;
         public static int
PROCESSING_ITERATIONS_MA
X = 50000;
         public static float[]
generatedWaveDistances;
         public static int
generatedWaveDistancesCount = 0;

         private static float[] longList;

         public static
LeanAudioOptions options(){

         if(generatedWaveDistances=
=null){
         generatedWaveDistances =
new float[
PROCESSING_ITERATIONS_MA
X ];
                  longList =
new float[
PROCESSING_ITERATIONS_MA
X ];
               }
               return new
LeanAudioOptions();
         }

         public static
LeanAudioStream
createAudioStream( AnimationCurve
volume, AnimationCurve frequency,
LeanAudioOptions options = null ){
               if(options==null)
                  options = new
LeanAudioOptions();

               options.useSetData =
false;

               int
generatedWavePtsLength =
createAudioWave( volume,
frequency, options);

         createAudioFromWave(
generatedWavePtsLength, options );

               return options.stream;
         }

         /**
         * Create dynamic audio from
a set of Animation Curves and other
options.
         *
         * @method createAudio
         * @param
{AnimationCurve}
volumeCurve:AnimationCurve
```

```
describing the shape of the audios volume (from 0-1). The length of the audio is dicated by the end value here.
	* @param {AnimationCurve} frequencyCurve:AnimationCurve describing the width of the oscillations between the sound waves in seconds. Large numbers mean a lower note, while higher numbers mean a tighter frequency and therefor a higher note. Values are usually between 0.01 and 0.000001 (or smaller)
	* @param {LeanAudioOptions} options:LeanAudioOptions You can pass any other values in here like vibrato or the frequency you would like the sound to be encoded at. See <a href="LeanAudioOptions.html">LeanAudioOptions</a> for more details.
	* @return {AudioClip} AudioClip of the procedurally generated audio
	* @example
	* AnimationCurve volumeCurve = new AnimationCurve( new Keyframe(0f, 1f, 0f, -1f), new Keyframe(1f, 0f, -1f, 0f));<br>
	* AnimationCurve frequencyCurve = new AnimationCurve( new Keyframe(0f, 0.003f, 0f, 0f), new Keyframe(1f, 0.003f, 0f, 0f));<br>
	* AudioClip audioClip = LeanAudio.createAudio(volumeCurve, frequencyCurve, LeanAudio.options().setVibrato( new Vector3[]{ new Vector3(0.32f,0f,0f)} ));<br>
	*/
	public static AudioClip createAudio( AnimationCurve volume, AnimationCurve frequency, LeanAudioOptions options = null ){
		if(options==null)
			options = new LeanAudioOptions();

		int generatedWavePtsLength = createAudioWave( volume, frequency, options);
		// Debug.Log("generatedWavePtsLength:"+generatedWavePtsLength);
		return createAudioFromWave( generatedWavePtsLength, options );
	}

	private static int createAudioWave( AnimationCurve volume, AnimationCurve frequency, LeanAudioOptions options ){
		float time = volume[ volume.length - 1 ].time;
		int listLength = 0;
		// List<float> list = new List<float>();

		// generatedWaveDistances = new List<float>();
		// float[] vibratoValues = new float[ vibrato.Length ];
		float passed = 0f;
		for(int i = 0; i < PROCESSING_ITERATIONS_MAX; i++){
			float f = frequency.Evaluate(passed);

	if(f<MIN_FREQEUNCY_PERIOD)
```

```
                f = MIN_FREQEUNCY_PERIOD;
                float height = volume.Evaluate(passed + 0.5f*f);

        if(options.vibrato!=null){
                for(int j=0; j<options.vibrato.Length; j++){

        float peakMulti = Mathf.Abs( Mathf.Sin( 1.5708f + passed * (1f/options.vibrato[j][0]) * Mathf.PI ) );

        float diff = (1f-options.vibrato[j][1]);

        peakMulti = options.vibrato[j][1] + diff*peakMulti;

        height *= peakMulti;
                }
            }


            // Debug.Log("i:"+i+" f:"+f+" passed:"+passed+" height:"+height+" time:"+time);
            if(passed + 0.5f*f>=time)
                break;
            if(listLength >= PROCESSING_ITERATIONS_MAX-1){

        Debug.LogError("LeanAudio has reached it's processing cap. To avoid this error increase the number of iterations ex: LeanAudio.PROCESSING_ITERATIONS_MAX = "+(PROCESSING_ITERATIONS_MAX*2));
                break;
            }else{
                int distPoint = listLength / 2;


        //generatedWaveDistances.Add( f );
                passed += f;


        generatedWaveDistances[ distPoint ] = passed;

        //Debug.Log("distPoint:"+distPoint+" passed:"+passed);


        //list.Add( passed );

        //list.Add( i%2==0 ? -height : height );


        longList[ listLength ] = passed;

        longList[ listLength + 1 ] = i%2==0 ? -height : height;
                }


                listLength += 2;

            }

            listLength += -2;

        generatedWaveDistancesCount = listLength / 2;

            /*float[] wave = new float[ listLength ];
```

```
                for(int i = 0; i < wave.Length; i++){
                        wave[i] = longList[i];
                }*/
                return listLength;
        }

        private static AudioClip createAudioFromWave( int waveLength, LeanAudioOptions options ){
                float time = longList[ waveLength - 2 ];
                float[] audioArr = new float[ (int)(options.frequencyRate*time) ];

                int waveIter = 0;
                float subWaveDiff = longList[waveIter];
                float subWaveTimeLast = 0f;
                float subWaveTime = longList[waveIter];
                float waveHeight = longList[waveIter+1];
                for(int i = 0; i < audioArr.Length; i++){
                        float passedTime = (float)i / (float)options.frequencyRate;
                        if(passedTime > longList[waveIter] ){

        subWaveTimeLast = longList[waveIter];

        waveIter += 2;

        subWaveDiff = longList[waveIter] - longList[waveIter-2];

        waveHeight = longList[waveIter+1];
                                //
Debug.Log("passed wave i:"+i);
                        }
                        subWaveTime = passedTime - subWaveTimeLast;
                        float ratioElapsed = subWaveTime / subWaveDiff;

                        float value = Mathf.Sin( ratioElapsed * Mathf.PI );


        if(options.waveStyle==LeanAudioOptions.LeanAudioWaveStyle.Square){

        if(value>0f)

        value = 1f;

        if(value<0f)

        value = -1f;
                        }else if(options.waveStyle==LeanAudioOptions.LeanAudioWaveStyle.Sawtooth){
                                float sign = value > 0f ? 1f : -1f;

        if(ratioElapsed<0.5f){

        value = (ratioElapsed*2f)*sign;
                                }else{ // 0.5f - 1f

        value = (1f - ratioElapsed)*2f*sign;
                                }
                        }else if(options.waveStyle==LeanAudioOptions.LeanAudioWaveStyle.Noise){
```

```
                                float peakMulti = (1f-options.waveNoiseInfluence) + Mathf.PerlinNoise(0f, passedTime * options.waveNoiseScale ) * options.waveNoiseInfluence;


        /*if(i<25){

        Debug.Log("passedTime:"+passedTime+" peakMulti:"+peakMulti+" infl:"+options.waveNoiseInfluence);
                                }*/

                                value *= peakMulti;
                        }

                        //if(i<25)
                        //
        Debug.Log("passedTime:"+passedTime+" value:"+value+" ratioElapsed:"+ratioElapsed+" subWaveTime:"+subWaveTime+" subWaveDiff:"+subWaveDiff);

                        value *= waveHeight;



        if(options.modulation!=null){
                                for(int k=0; k<options.modulation.Length; k++){

        float peakMulti = Mathf.Abs( Mathf.Sin( 1.5708f + passedTime * (1f/options.modulation[k][0]) * Mathf.PI ) );

        float diff = (1f-options.modulation[k][1]);

        peakMulti = options.modulation[k][1] + diff*peakMulti;

        // if(k<10){

        // Debug.Log("k:"+k+" peakMulti:"+peakMulti+" value:"+value+" after:"+(value*peakMulti));

        // }

        value *= peakMulti;
                                }
                        }

                        audioArr[i] = value;
                        // Debug.Log("pt:"+pt+" i:"+i+" val:"+audioArr[i]+" len:"+audioArr.Length);
                }


                int lengthSamples = audioArr.Length;

                #if UNITY_3_5 || UNITY_4_0 || UNITY_4_0_1 || UNITY_4_1 || UNITY_4_2 || UNITY_4_3 || UNITY_4_5 || UNITY_4_6 || UNITY_4_7
                bool is3dSound = false;
                AudioClip audioClip = AudioClip.Create("Generated Audio", lengthSamples, 1, options.frequencyRate, is3dSound, false);
                #else
                AudioClip audioClip = null;
```

```
        if(options.useSetData){
                audioClip =
AudioClip.Create("Generated
Audio", lengthSamples, 1,
options.frequencyRate, false, null,
OnAudioSetPosition);

        audioClip.SetData(audioArr,
0);
            }else{
                options.stream
= new LeanAudioStream(audioArr);
                //
Debug.Log("len:"+audioArr.Length+
" lengthSamples:"+lengthSamples+"
freqRate:"+options.frequencyRate);
                audioClip =
AudioClip.Create("Generated
Audio", lengthSamples, 1,
options.frequencyRate, false,
options.stream.OnAudioRead,
options.stream.OnAudioSetPosition);

        options.stream.audioClip =
audioClip;
            }

            #endif

            return audioClip;
        }

        private static void
OnAudioSetPosition(int
newPosition) {

    }

        public static AudioClip
generateAudioFromCurve(
AnimationCurve curve, int
frequencyRate = 44100 ){
            float curveTime =
curve[ curve.length - 1 ].time;
            float time =
curveTime;
            float[] audioArr =
new float[ (int)(frequencyRate*time)
];

            //
Debug.Log("curveTime:"+curveTim
e+"
AudioSettings.outputSampleRate:"+
AudioSettings.outputSampleRate);
            for(int i = 0; i <
audioArr.Length; i++){
                float pt =
(float)i / (float)frequencyRate;
                audioArr[i] =
curve.Evaluate( pt );
                //
Debug.Log("pt:"+pt+" i:"+i+"
val:"+audioArr[i]+"
len:"+audioArr.Length);
            }

            int lengthSamples =
audioArr.Length;//(int)(
(float)frequencyRate * curveTime );
            #if UNITY_3_5 ||
UNITY_4_0 || UNITY_4_0_1 ||
UNITY_4_1 || UNITY_4_2 ||
UNITY_4_3 || UNITY_4_5 ||
UNITY_4_6 || UNITY_4_7
            bool is3dSound =
false;
            AudioClip audioClip
= AudioClip.Create("Generated
Audio", lengthSamples, 1,
frequencyRate, is3dSound, false);
            #else
            AudioClip audioClip
= AudioClip.Create("Generated
Audio", lengthSamples, 1,
frequencyRate, false);
            #endif

        audioClip.SetData(audioArr,
0);
```

```
		return audioClip;
	}

	public static AudioSource play( AudioClip audio, float volume ){
		AudioSource audioSource = playClipAt(audio, Vector3.zero);
		audioSource.volume = volume;
		return audioSource;
	}

	public static AudioSource play( AudioClip audio ){
		return playClipAt( audio, Vector3.zero );
	}

	public static AudioSource play( AudioClip audio, Vector3 pos ){
		return playClipAt( audio, pos );
	}

	public static AudioSource play( AudioClip audio, Vector3 pos, float volume ){
		// Debug.Log("audio length:"+audio.length);
		AudioSource audioSource = playClipAt(audio, pos);

	audioSource.minDistance = 1f;
		//audioSource.pitch = pitch;
		audioSource.volume = volume;

		return audioSource;
	}

	public static AudioSource playClipAt( AudioClip clip, Vector3 pos ) {
		GameObject tempGO = new GameObject(); // create the temp object

	tempGO.transform.position = pos; // set its position
		AudioSource aSource = tempGO.AddComponent<AudioSource>(); // add an audio source
		aSource.clip = clip; // define the clip
		aSource.Play(); // start the sound

	GameObject.Destroy(tempGO, clip.length); // destroy object after clip duration
		return aSource; // return the AudioSource reference
	}

	public static void printOutAudioClip( AudioClip audioClip, ref AnimationCurve curve, float scaleX = 1f ){
		// Debug.Log("Audio channels:"+audioClip.channels+" frequency:"+audioClip.frequency+" length:"+audioClip.length+" samples:"+audioClip.samples);
		float[] samples = new float[audioClip.samples * audioClip.channels];
	audioClip.GetData(samples, 0);
	int i = 0;

	Keyframe[] frames = new Keyframe[samples.Length];
	while (i < samples.Length) {
	  frames[i] = new Keyframe( (float)i * scaleX, samples[i] );
```

```
            ++i;
        }
        curve = new AnimationCurve( frames );
        }
}


/**
* Pass in options to LeanAudio
*
* @Class LeanAudioOptions
* @constructor
*/
public Class LeanAudioOptions : object {

        public enum LeanAudioWaveStyle{
                Sine,
                Square,
                Sawtooth,
                Noise
        }

        public LeanAudioWaveStyle waveStyle = LeanAudioWaveStyle.Sine;
        public Vector3[] vibrato;
        public Vector3[] modulation;
        public int frequencyRate = 44100;
        public float waveNoiseScale = 1000;
        public float waveNoiseInfluence = 1f;

        public bool useSetData = true;
        public LeanAudioStream stream;

        public LeanAudioOptions(){}

        /**
        * Set the frequency for the audio is encoded. 44100 is CD quality, but you can usually get away with much lower (or use a lower amount to get a more 8-bit sound).
        *
        * @method setFrequency
        * @param {int} frequencyRate:int of the frequency you wish to encode the AudioClip at
        * @return {LeanAudioOptions} LeanAudioOptions describing optional values
        * @example
        * AnimationCurve volumeCurve = new AnimationCurve( new Keyframe(0f, 1f, 0f, -1f), new Keyframe(1f, 0f, -1f, 0f));<br>
        * AnimationCurve frequencyCurve = new AnimationCurve( new Keyframe(0f, 0.003f, 0f, 0f), new Keyframe(1f, 0.003f, 0f, 0f));<br>
        * AudioClip audioClip = LeanAudio.createAudio(volumeCurve, frequencyCurve, LeanAudio.options().setVibrato( new Vector3[]{ new Vector3(0.32f,0f,0f)} ).setFrequency(12100) );<br>
        */
        public LeanAudioOptions setFrequency( int frequencyRate ){
                this.frequencyRate = frequencyRate;
                return this;
        }

        /**
        * Set details about the shape of the curve by adding vibrato modulations through it (alters the peak values giving it a wah-wah effect). You can add as many as you
```

```
want to sculpt out more detail in the sound wave.
		*
		* @method setVibrato
		* @param {Vector3[]} vibratoArray:Vector3[] The first value is the period in seconds that you wish to have the vibrato wave fluctuate at. The second value is the minimum height you wish the vibrato wave to dip down to (default is zero). The third is reserved for future effects.
		* @return {LeanAudioOptions} LeanAudioOptions describing optional values
		* @example
		* AnimationCurve volumeCurve = new AnimationCurve( new Keyframe(0f, 1f, 0f, -1f), new Keyframe(1f, 0f, -1f, 0f));<br>
		* AnimationCurve frequencyCurve = new AnimationCurve( new Keyframe(0f, 0.003f, 0f, 0f), new Keyframe(1f, 0.003f, 0f, 0f));<br>
		* AudioClip audioClip = LeanAudio.createAudio(volumeCurve, frequencyCurve, LeanAudio.options().setVibrato( new Vector3[]{ new Vector3(0.32f,0.3f,0f)} ).setFrequency(12100) );<br>
		*/
		public LeanAudioOptions setVibrato( Vector3[] vibrato ){
			this.vibrato = vibrato;
			return this;
		}

		/*
		public LeanAudioOptions setModulation( Vector3[] modulation ){
			this.modulation = modulation;
			return this;
		}*/

		public LeanAudioOptions setWaveSine(){
			this.waveStyle = LeanAudioWaveStyle.Sine;
			return this;
		}

		public LeanAudioOptions setWaveSquare(){
			this.waveStyle = LeanAudioWaveStyle.Square;
			return this;
		}

		public LeanAudioOptions setWaveSawtooth(){
			this.waveStyle = LeanAudioWaveStyle.Sawtooth;
			return this;
		}

		public LeanAudioOptions setWaveNoise(){
			this.waveStyle = LeanAudioWaveStyle.Noise;
			return this;
		}

		public LeanAudioOptions setWaveStyle( LeanAudioWaveStyle style ){
			this.waveStyle = style;
			return this;
		}


		public LeanAudioOptions setWaveNoiseScale( float waveScale ){
```

```
                this.waveNoiseScale
= waveScale;
                return this;
        }

        public LeanAudioOptions
setWaveNoiseInfluence( float
influence ){
                this.waveNoiseInfluence =
influence;
                return this;
        }
}
```

PopUp

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

public Class PopUp :
MonoBehaviour
{
   public Canvas cow;
   public Canvas goat;
   public Canvas wolf;
   public Canvas goose;
   public Canvas shark;
   public Canvas chicken;
   public Canvas panduan;
   public Canvas creator;
   public Canvas tanya;

   public bool a = false;
   public bool b = false;
   public bool c = false;
   public bool d = false;
   public bool e = false;
   public bool f = false;
   public bool g = false;
   public bool h = false;
   public bool i = false;


   public void sapi()
   {
      if (a == false)
      {
         a = true;
         cow.enabled = true;
      }
      else if (a == true)
      {
         a = false;
         cow.enabled = false;
      }
   }

   public void kambing()
   {
      if (b == false)
      {
         b = true;
         goat.enabled = true;

      }
      else if (b == true)
      {
         b = false;
         goat.enabled = false;
      }
   }

   public void serigala()
   {
      if (c == false)
      {
         c = true;
         wolf.enabled = true;

      }
      else if (c == true)
      {
```

```
        c = false;
        wolf.enabled = false;
    }
}

public void angsa()
{
    if (d == false)
    {
        d = true;
        goose.enabled = true;

    }
    else if (d == true)
    {
        d = false;
        goose.enabled = false;
    }
}

public void hiu()
{
    if (e == false)
    {
        e = true;
        shark.enabled = true;

    }
    else if (e == true)
    {
        e = false;
        shark.enabled = false;
    }
}

public void ayam()
{
    if (f == false)
    {
        f = true;
        chicken.enabled = true;

    }
    else if (f == true)
    {
        f = false;
        chicken.enabled = false;
    }
}

public void poppanduan()
{
    if (g == false)
    {
        g = true;
        panduan.enabled = true;

    }
    else if (g == true)
    {
        g = false;
        panduan.enabled = false;
    }
}

public void popcreator()
{
    if (h == false)
    {
        h = true;
        creator.enabled = true;

    }
    else if (h == true)
    {
        h = false;
        creator.enabled = false;
    }
}

public void poptanya()
{
    if (i == false)
    {
        i = true;
        tanya.enabled = true;

    }
    else if (i == true)
    {
        i = false;
        tanya.enabled = false;
```

```
        }
    }
}
```

ScaleInOut

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

public Class ScaleInOut :
MonoBehaviour
{
    public GameObject Object;

    private bool _ZoomIn;
    private bool _ZoomOut;

    //object scale speed
    public float Scale = 0.1f;

    // Update is called once per frame
    void Update()
    {
        if (_ZoomIn)
        {
            //make a bigger object
            Object.transform.localScale
+= new Vector3(Scale, Scale, Scale);
        }

        if (_ZoomOut)
        {
            //make a small object
            Object.transform.localScale -
= new Vector3(Scale, Scale, Scale);
        }
    }

    //Make object scaled big
    public void OnPressZoomIn()
    {
        _ZoomIn = true;
    }

    public void OnReleaseZoomIn()
    {
        _ZoomIn = false;
    }

    //Make object scaled small
    public void OnPressZoomOut()
    {
        _ZoomOut = true;
    }

    public void OnReleaseZoomOut()
    {
        _ZoomOut = false;
    }
}
```