.TRAIL

The new way to write sequential code.

User Guide

Release 1.3.3

Contents

Overview

This User Guide was designed to provide Dot Trail users with a basic overview of the features and functionality of the library.

Installation

Once you have downloaded Dot Trail from Unity's Asset Store, go to: "Assets->Import Package->Custom Package...". In the Import Asset window, find and select the Dot Trail. unitypackage file. After the "Importing package" window appears in Unity, verify that all items to import are selected and then click the Import button in the bottom right of the window.

API

How To Use

If you want to use Dot Trail, you must first create a class that inherits the "TrailObject<T>" class.

```
using Ra.Trail;
public class DotTrailTest : TrailObject<DotTrailTest>
{

}
```

After you create the class, you can start using Dot Trail. Create your class object. Then call the methods you want.

```
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      DotTrailTest.Trail
         .Wait(5f)
         .Print("After 5 seconds.");
   }
}
```

Note: If you don't need to add your own trail methods, just use "Dot.Trail".

```
using UnityEngine;
using Ra.Trail;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Wait(5f)
         .Print("After 5 seconds.");
   }
}
```

T Print(object text = null, Func<object> action = null)

Prints text to the console.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Print(".Trail");
   }
}
```

T Print(Func<object> action = null)

Prints text to the console.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Print(() => transform.position.x);
   }
}
```

T Wait(float duration)

It waits for a certain time before proceeding to the next operation.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Wait(1f)
         .Print(() => transform.position.x)
         .Wait(2f)
         .Print(() => transform.position.y)
         .Wait(3f)
         .Print(() => transform.position.z);
   }
}
```

T Wait(Action<bool> action)

It waits for the state to be true before proceeding to the next action.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Wait(() => transform.position.z > 5)
         .Print(".Trail");
   }
}
```

T After(Action afterAction)

Runs function during operation.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Wait(1f)
         .After(() => GameObject.CreatePrimitive(PrimitiveType.Cube);});
   }
}
```

T AfterDirect(Action afterAction)

Runs a function without being bound to a condition or sequential method.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .MethodStart("Test")
                  .AfterDirect(() =>
                  {
                      GameObject.CreatePrimitive(PrimitiveType.Cube);
                  })
                  .MethodEnd();
   }
}
```

T If(Func<bool> condition)

T EndIf()

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .If(() => Application.platform == RuntimePlatform.Android)
         .Print("Android")
         .EndIf();
   }
}
```

T Else()

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .If(() => Application.platform == RuntimePlatform.Android)
         .Print("Android")
         .Else()
         .Print("Editor")
         .EndIf();
   }
}
```

T ElseIf(Func<bool> condition)

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .If(() => Application.platform == RuntimePlatform.Android)
         .Print("Android")
         .ElseIf(() => Application.platform == RuntimePlatform.IPhonePlayer)
         .Print("IPhone")
         .Else()
         .Print("Editor")
         .EndIf();
   }
}
```

T Label(string labelName)

T Goto(string labelName)

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var turn = true;
      Dot.Trail
         .Label("T")
         .Print("Hello!")
         .If(() => turn)
            .After(() => turn = false)
            .Goto("T")
         .EndIf();
   }
}
```

T When(Func<bool> statement, Action action)

Runs when the variable changes to true.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .When(() => Input.GetButton("Fire1"), () =>
         {
            Debug.Log("Clicked.");
         });
   }
}
```

T When(Func<bool> statement)

T EndWhen()

Runs when the variable changes to true.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .When(() => Input.GetButton("Fire1"))
            .Print("Clicked.")
         .EndWhen();
   }
}
```

T OnChange<Z>(Func<Z> statement, Action<Z> action)

Runs when the variable changes.

```
using Ra.Trail;
using UnityEngine;
public class DotTrailUsingTest : MonoBehaviour
{
   public Transform reference;
   private void Start()
   {
      Dot.Trail
         .OnChange(() => reference, last =>
         {
            Destroy(last.gameObject);
         });
   }
}
```

T FixedLoop(Action inUpdate)

T Loop(Action inUpdate)

It runs the function on every frame.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var i = 0;
      Dot.Trail
         .Loop(() =>
         {
            i++;
            Debug.Log(i);
         });
   }
}
```

T FixedLoop(Action<object[]> inUpdate, params Func<object>[] arguments)

T Loop(Action<object[]> inUpdate, params Func<object>[] arguments)

Runs the Function every frame with arguments compiled in sequential time.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .Wait(5f)
         .Loop ((args) =>
         {
            var firstPosition = (Vector3)args[0];
            Debug.Log(Vector3.Distance(transform.position, firstPosition));
         }, () => transform.position);
   }
}
```

T UpdateStart()

Starts sequential update.

T UpdateEnd()

Ends sequential update.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      Dot.Trail
         .UpdateStart()
                  .Print("In Update.")
                  .UpdateEnd();
   }
}
```

T Method(string methodName, params string[] args)

Initiates a sequential method.

T EndMethod()

Ends the sequential method.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Method("SayHi", "name")
                   .Print("Hi, " + tr.currentMethod["name"])
                   .EndMethod();
   }
}
```

T Call(string methodName, params VariableInfo[] arguments)

Invokes a sequential method.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Method("SayHi", "name")
                   .Print("Hi, " + tr.currentMethod["name"])
                   .EndMethod()
         .Define("name", "Eric")
         .Call("SayHi", tr["name"]);
   }
}
```

T Define(string varName, object value)

Defines a variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr  = Dot.Trail;
      tr
         .Define("f", 1234)
         .Print(tr["f"]);
   }
}
```

T SetVar(string varName, object value = null, Func<object> action = null)

Changes the value of the variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr =  Dot.Trail;
      tr
         .Define("f", 1234)
         .SetVar("f", 213)
         .Print(tr["f"]);
   }
}
```

T SetVar(string varName, Func<object> action)

Changes the value of the variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Define("f", 1234)
         .SetVar("f", () => transform.rotation.y)
         .Print(tr["f"]);
   }
}
```

T IncreaseVar(string varName, object value = null, Func<object> action = null)

Increments the value of the variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Define("f", 0)
         .IncreaseVar("f", 10)
         .Print(tr["f"]);
   }
}
```

T IncreaseVar(string varName, Func<object> action)

Increments the value of the variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Wait(2f)
         .Define("f", 0)
         .IncreaseVar("f", () => transform.childCount)
         .Print(tr["f"]);
   }
}
```

object Resolve(object property)

Parses and returns the object.

```
public TrailTest Log(object text)
{
   After(() =>
   {
      Debug.Log(Resolve(text));
   });
   return this;
}
```

VariableInfo this[string name]

Finds and returns the defined variable.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      var tr = Dot.Trail;
      tr
         .Define("val", 15)
         .Print(tr["val"]);
   }
}
```

Create your own trail methods

All you have to do is create a method and return the class. If you want it to stick to the sequence use "After" in the method.

```
using Ra.Trail;
using UnityEngine;

public class DotTrailTest : TrailObject<DotTrailTest>
{
   public DotTrailTest CreateCube(Vector3 position)
   {
      After(() =>
      {
         var go = GameObject.CreatePrimitive(PrimitiveType.Cube);
         go.transform.position = position;
      });
      return this;
   }
}
```

```
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   private void Start()
   {
      new DotTrailTest()
         .Wait(2f)
         .CreateCube(Vector3.zero);
   }
}
```

Extras

Vectors

Vector3 x(float value)

Returns Vector3 for only x axis.

Vector3 y(float value)

Returns Vector3 for only y axis.

Vector3 z(float value)

Returns Vector3 for only z axis.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
    private void Start()
    {
        Debug.Log(Vectors.x(5));
    }
}
```

Vector3 mouseInputDelta

Returns the mouse delta position vertically. Compatible with Android, IOS and computers.

Vector3 mouseInputDeltaHorizontal

Returns the mouse delta position horizontally. Compatible with Android, IOS and computers.

Vector3 standardInputDelta

Returns the delta position of the arrow keys vertically. Compatible with computers.

Vector3 standardInputDeltaRaw

Returns the raw delta position of the arrow keys vertically. Compatible with computers.

Vector3 standardInputDeltaHorizontal

Returns the delta position of the arrow keys horizontally. Compatible with computers.

Vector3 standardInputDeltaRawHorizontal

Returns the raw delta position of the arrow keys horizontally. Compatible with computers.

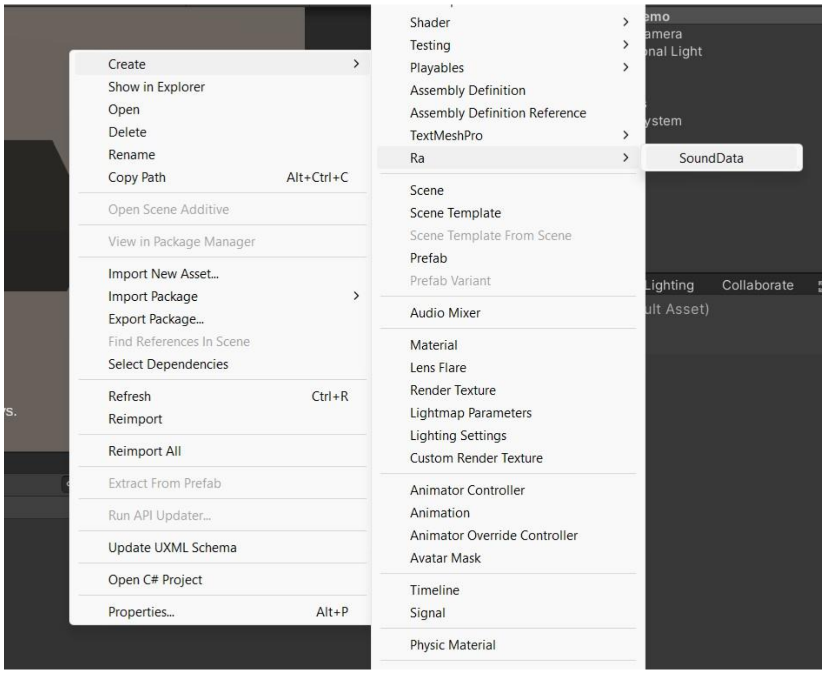
```
using Ra;
using Ra.Trail;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
  [Range(0, 1)] public float speed;
  private void Start()
  {
    Dot.Trail.FixedLoop(() =>
    {
      transform.position += Vectors.standardInputDeltaRaw * speed;
    });
  }
}
```

SoundMaker

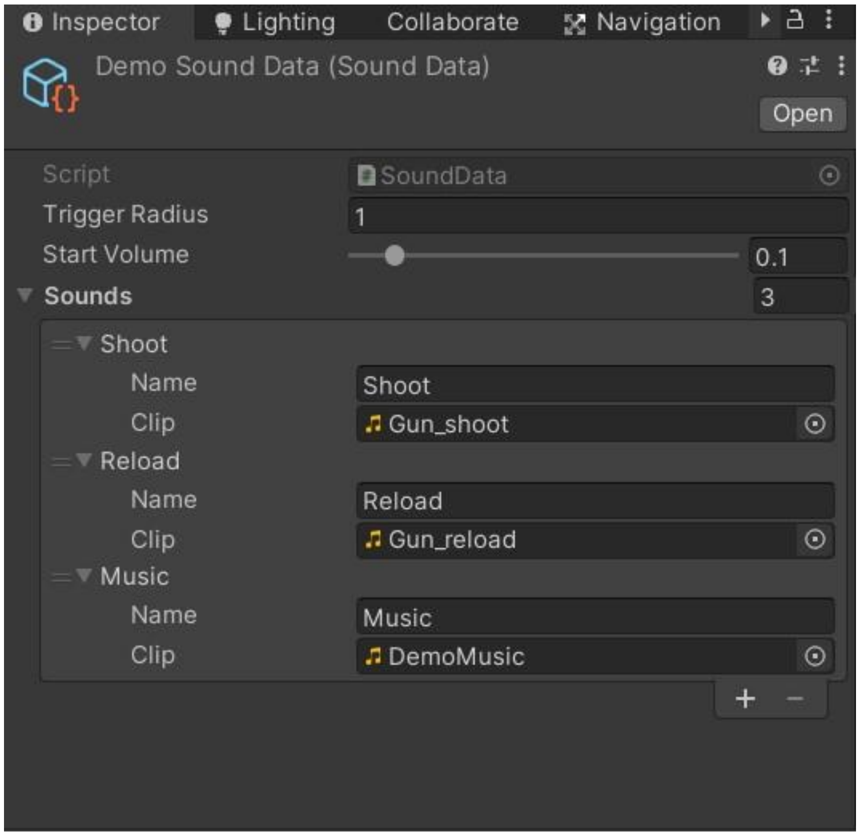
SoundMaker is a DotTrail plugin with which you can change the volume depending on time or location.

How to use?

First of all, you have to create a sound data.



Add your sounds to the data you created.



Demo Sound Data (Sound Data)

Script	SoundData
Trigger Radius	1
Start Volume	0.1
Sounds	3
Shoot	
Name	Shoot
Clip	Gun_shoot
Reload	
Name	Reload
Clip	Gun_reload
Music	
Name	Music
Clip	DemoMusic

After that you need to create a script and add your sound data as reference.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
   public SoundData soundData;
   private void Start()
   {

   }
}
```

SoundMaker Configure(AudioSource source, SoundData data)

Applies a sound data to SoundMaker that it can use.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
```

```
    public SoundData soundData;
    private void Start()
    {
        SoundMaker.Trail.Configure(audioSource, soundData);
    }
}
```

SoundMaker Play(string clipName)

Finds and plays the named sound in the sound data.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
    public AudioSource audioSource;
    public SoundData soundData;
    private void Start()
    {
        SoundMaker.Trail.Configure(audioSource, soundData).Play("Shoot");
    }
}
```

SoundMaker SetVolume(float volume)

Changes the volume.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
    public AudioSource audioSource;
    public SoundData soundData;
    private void Start()
    {
        SoundMaker.Trail.Configure(audioSource, soundData).Play("Shoot").SetVolume(0.5f);
    }
}
```

SoundMaker SetVolumeSmoothly(float volume, float time)

Smoothly changes the volume in the specified time.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
   public SoundData soundData;
   private void Start()
   {
      SoundMaker.Trail.Configure(audioSource, soundData)
         .Play("Music")
         .SetVolumeSmoothly(0.5f, 1);
   }
}
```

SoundMaker SetVolumeTrigger(string tag, float enterValue, float exitValue, float time)

Changes the volume depending on whether the object is in the trigger or not.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
   public SoundData soundData;
   private void Start()
   {
      SoundMaker.Trail.Configure(audioSource, soundData)
         .Play("Music")
         .SetVolumeTrigger("Area", 1f, 0.1f, 2f);
   }
}
```

SoundMaker SetVolumeTriggerCurve(string tag, AnimationCurve enterCurve, AnimationCurve exitCurve)

Changes the sound according to the curve, depending on whether the object is on trigger or not.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
   public SoundData soundData;
   public AnimationCurve enterCurve, exitCurve;
   private void Start()
   {
      SoundMaker.Trail.Configure(audioSource, soundData)
         .Play("Music")
         .SetVolumeTriggerCurve("Area", enterCurve, exitCurve);
   }
}
```

SoundMaker WaitForTriggerEnter(string tag)

It waits until the object enters the trigger.

```
using Ra;
using UnityEngine;

public class DotTrailUsingTest : MonoBehaviour
{
   public AudioSource audioSource;
   public SoundData soundData;
   private void Start()
   {
      SoundMaker.Trail.Configure(audioSource, soundData)
         .Play("Music")
         .WaitForTriggerEnter("Area")
         .Play("Shoot")
         .Wait(0.5f)
         .Play("Reload");
   }
}
```

SoundMaker SetTransform(Transform newTransform)

Changes the central transform for the trigger control. The default center transform is the transform where the AudioSource component is located.

DotAI

DotAI is a DotTrail plugin with which you can easily design AI agents. This feature will be available in future versions.