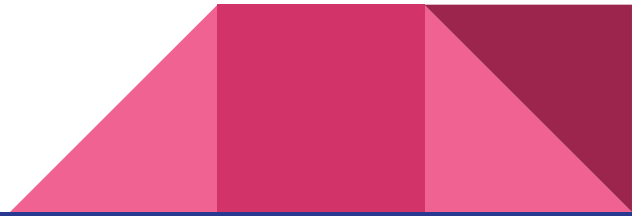
Scripts

Criando seus próprios componentes

Topicos

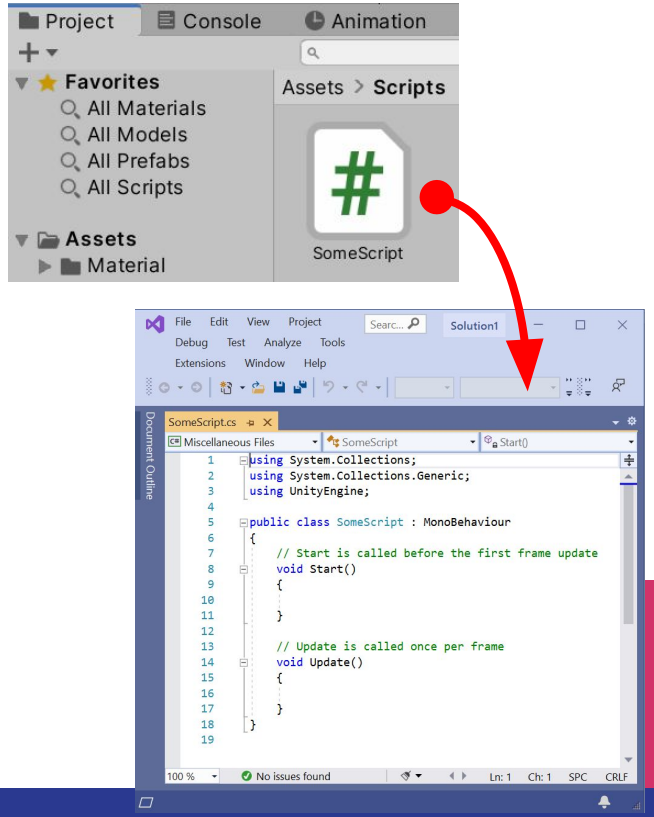
- A anatomia de um script na unity
  - Using, Class MonoBehaviour, Loops (Start e Update)
- Variáveis e propriedades
- Debug
- Componente detector de colisao
- Scripts Create e destroy



Criando e abrindo um novo Script

Os Scripts são como qualquer outro asset no projeto, e podem ser criados da mesma forma, através da opção Assets> Create > C# Script

Ao clicar duas vezes no script o mesmo será aberto no Visual Studio por padrão.



Anatomia

Faz referência a outras classes e permite usar seus métodos, propriedades, etc...

O nome da classe deve ser igual ao nome do asset

O MonoBehaviour é uma classe que permite que seu script se torne um componente e possa ser adicionado aos GameObjects.

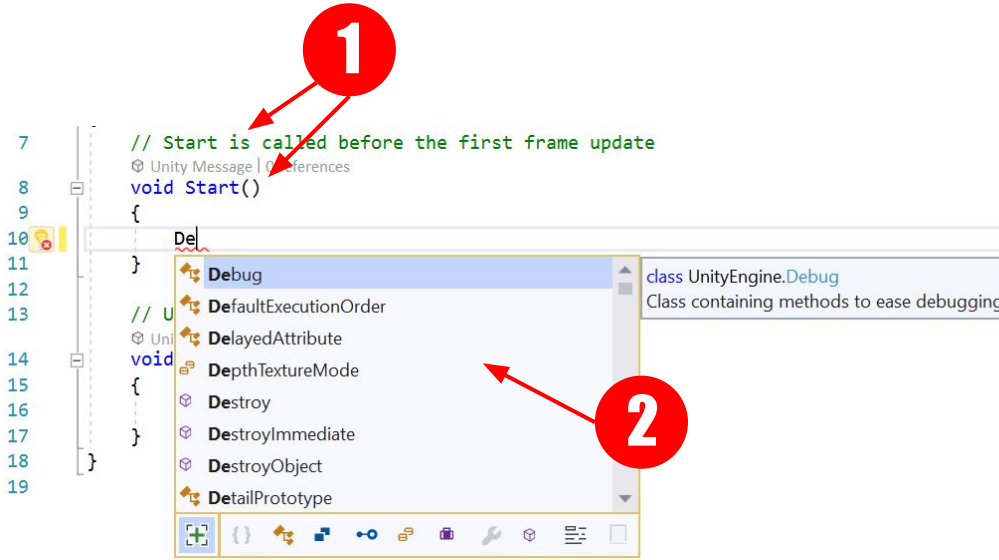
Funções padrão do MonoBehaviour.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SomeScript : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```

Sintaxe e Autocompletar

Editores de script como o Visual Studio possuem alguns recursos interessantes, como **destaque de sintaxe** (1), que diferencia os elementos por cores e o recurso de **autocompletar** (2), que sugere classes e métodos disponíveis e inclui descrições de uso.

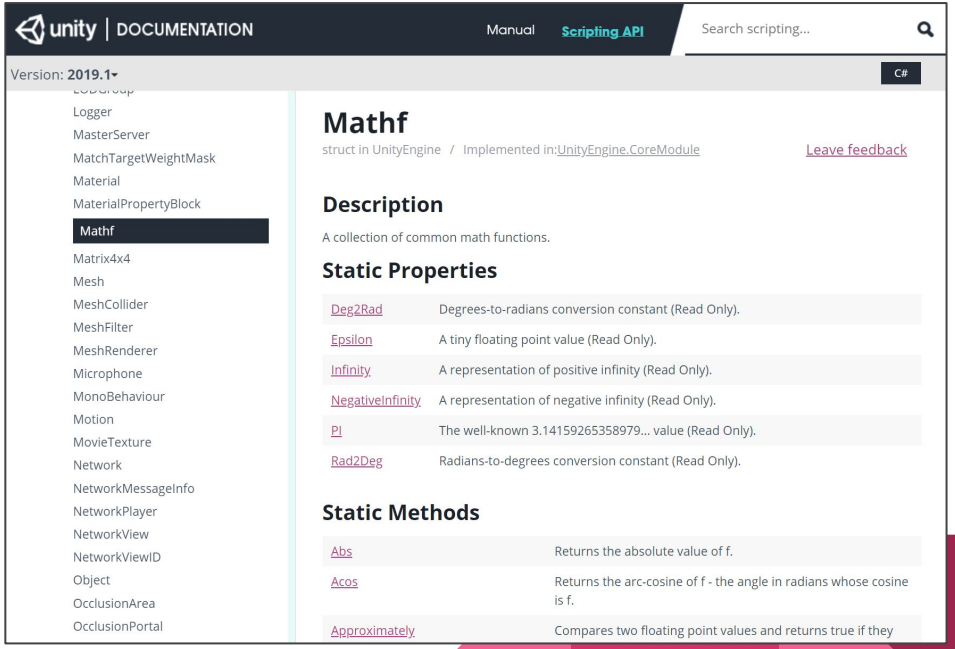
(Em caso de problemas, veja se o Visual Studio está configurado na Unity, em **Edit > Preferences > External Tools**)



Documentação

Sempre consulte a documentação para desenvolver ou em caso de dúvidas.

docs.unity3d.com/ScriptReference



Teste simples e Console

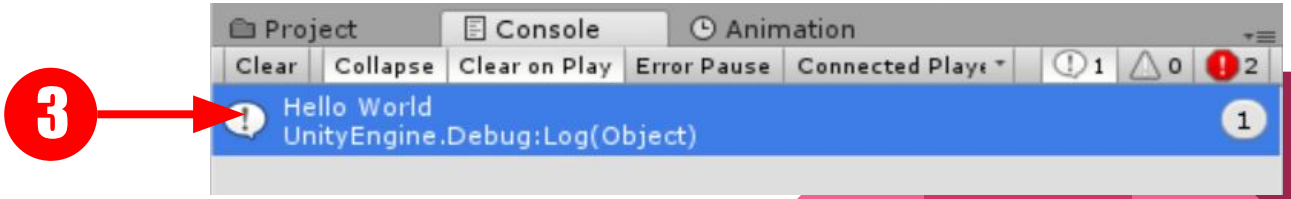
Para testar o funcionamento do script, incluir as linhas como no exemplo ao lado (1 e 2) e clicar em play

Conferir o resultado no Console (3)

```
5   public class SomeScript : MonoBehaviour
6   {
7       public string MyText = "Hello World";
            Unity Message | 0 references
8       void Start()
9       {
10          Debug.Log(MyText);
11      }
12  }
```

1

2



3

Erros comuns no C#

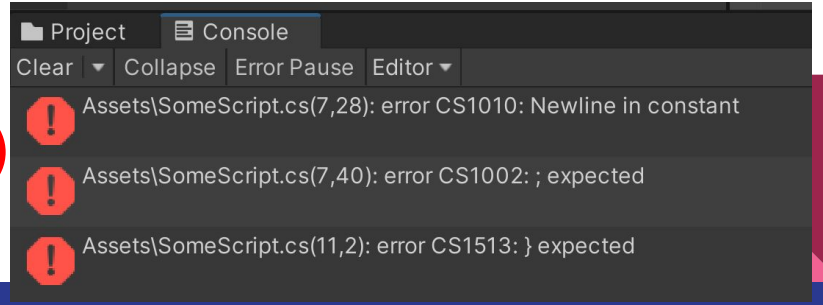
Aspas, colchetes e parenteses (1 e 3) devem estar sempre completos. Ponto e vírgula (2) devem constar ao fim de cada sentença.

Atente para erros sublinhados (em vermelho).

Muitas vezes também são sinalizados no Console (4)

```
public class SomeScript : MonoBehaviour
{
    public string MyText = "Hello World
    Unity Message | 0 references
    void Start()
    {
        Debug.Log(MyText);
    }
```

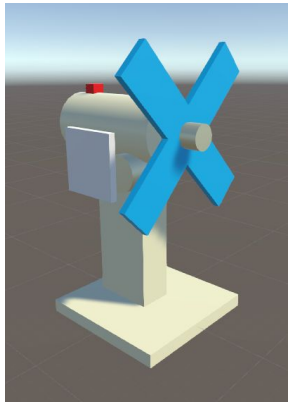
1 2 3

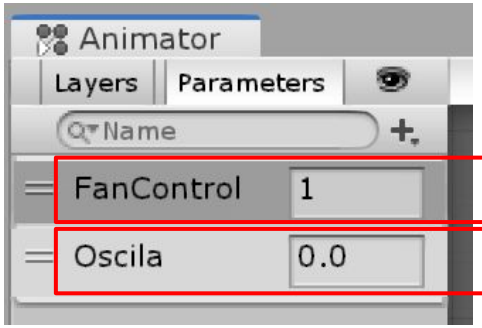


4

Script para Controlar o Ventilador

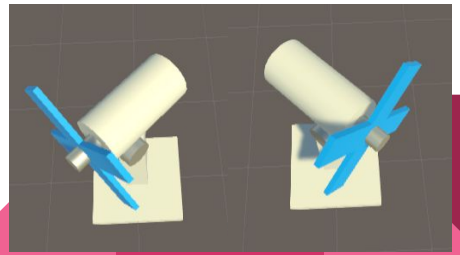
Vamos criar um script para controlar os parâmetros do Animator Controller do prefab de ventilador criado na aula anterior.





Velocidade da Hélice (0 a 3)

Oscilação (0.0 ou 1.0)

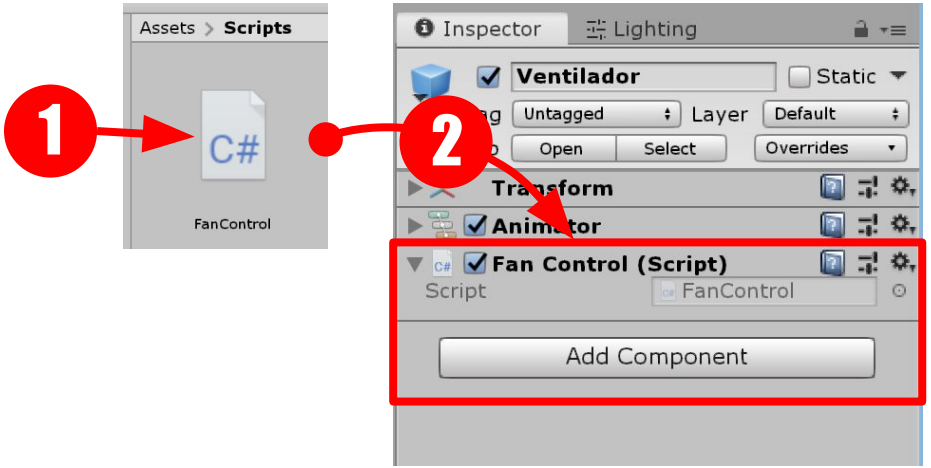


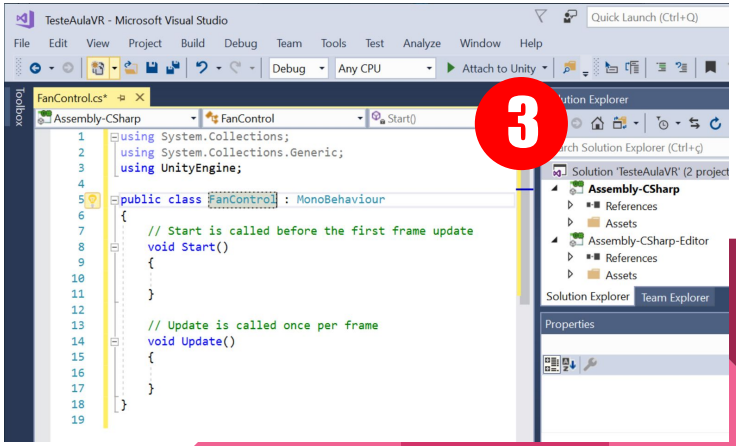
Adicionando o Script

Em Project:
Crie um novo script chamado “FanControl” (1)

Arraste o script (2) para o objeto Ventilador (ou adicione pelo botão Add Component)

Clicar 2x no script para abrir o Visual Studio (3)





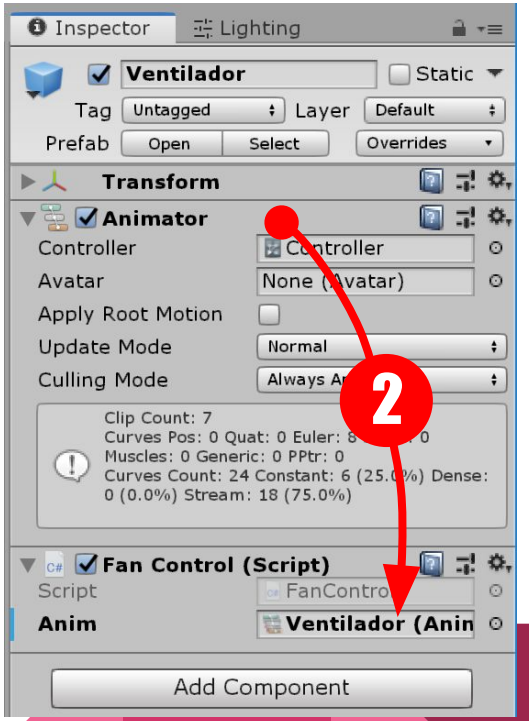
Acessando o Animator

O primeiro passo é criar uma variável para armazenar o **Animator Controller** para que possamos acessar seus parâmetros.

Crie a variável classe Animator chamada “anim” (1) e **salvar** (CTRL+S). O modificador “public” garante que ela apareça no Inspector. Assim é possível arrastar o componente Animator p/ campo “Anim” (2) do script. Sempre salve o script no Visual Studio antes de testar. A Unity atualizará prontamente.

```
5   public class FanControl : MonoBehaviour
6   {
7       public Animator anim;
8
9       // Start is called before the first
10      void Start()
11      {
```

1



Loop e Métodos

Criar uma booleana chamada “oscillator” (1)

Usar o Loop Update (executado a cada frame) para alterar o parâmetro do Animator dependendo do valor do “oscillator”: true = 1.0; false = 0.0 (2)

Criar método público “SwitchOscillator” para modificar a variável “oscillator” (3). Chamaremos esse método a partir de um botão a seguir

```
public class FanControl : MonoBehaviour
{
    public Animator anim;
    private bool oscillator;

    void Update() {
        if(oscillator)
            anim.SetFloat("Oscila", 1.0f);
        else
            anim.SetFloat("Oscila", 0.0f);
    }
    public void SwitchOscillator() {
        if (oscillator) {
            oscillator = false;
        }else{
            oscillator = true;
        }
    }
}
```

1

2

3

```
public class FanControl : MonoBehaviour
{
    public Animator anim;
    private bool oscillator;

    void Update() {
        if(oscillator)
            anim.SetFloat("Oscila", 1.0f);
        else
            anim.SetFloat("Oscila", 0.0f);
    }
    public void SwitchOscillator() {
        if (oscillator) {
            oscillator = false;
        }else{
            oscillator = true;
        }
    }
}
```

Chamando o Método

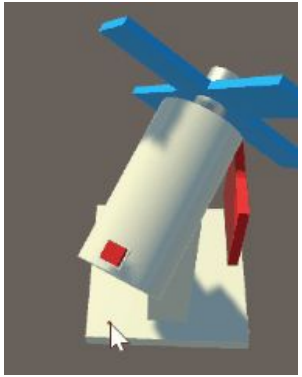
Utilize o botão “Hover Button” criado na aula 5

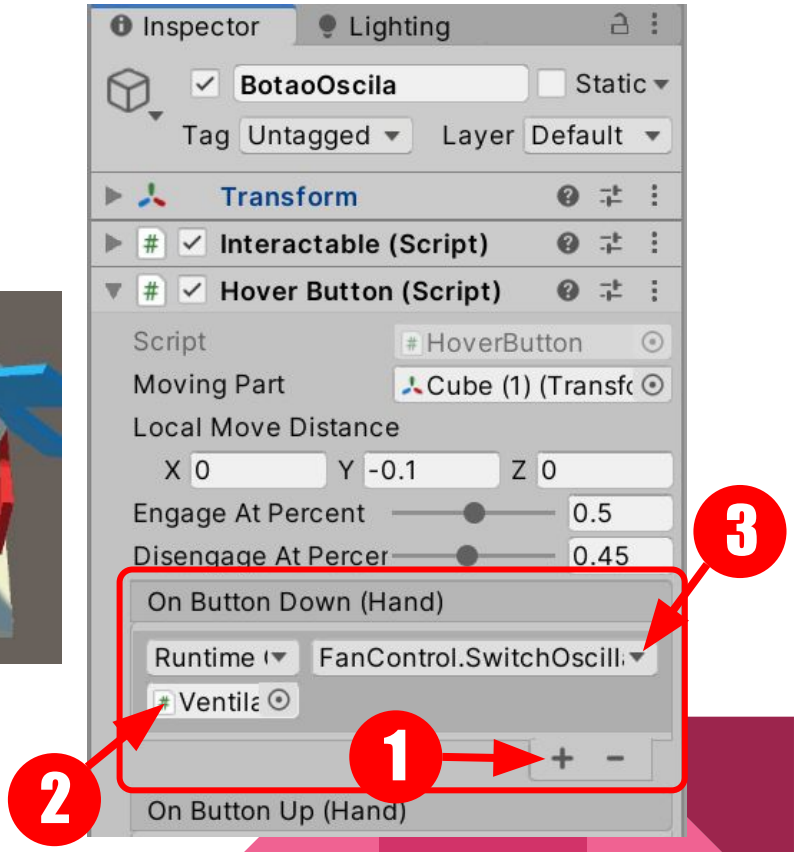
Em “On Button Down”, clicar em “+” (1)

Arrastar o objeto Ventilador da cena (2)

No dropbox, selecionar nosso script e método: FanControl > SwitchOscillator (3)

Dependendo de como o botão foi feito, pode ser necessário ajustar a sensibilidade, alterando os parametros “Engage/Disengage at Percent”





Power

Criar uma variável pública INT chamada “power” (1)

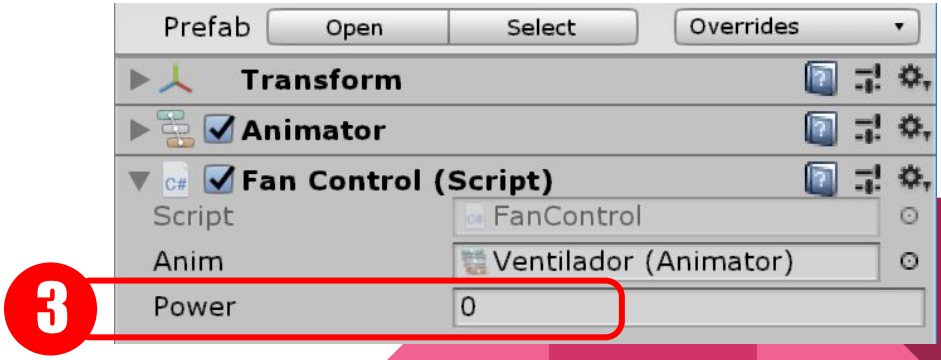
No método Update, alterar parâmetro “FanControl” com o valor de power (2)

Teste: Play, modificar o valor de power no Inspector (3) e observar a animação correspondente

```
public Animator anim;
private bool oscillator;
public int power;

void Update() {
    anim.SetInteger("FanControl", power);

    if (oscillator)
```

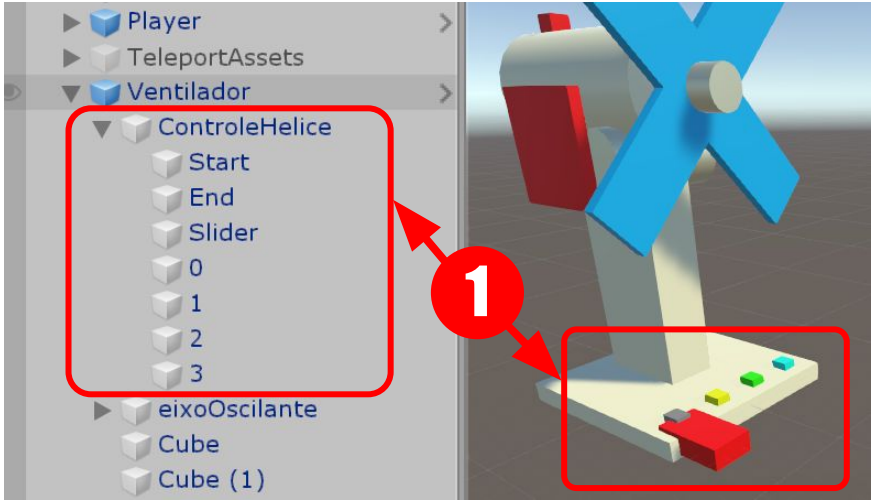


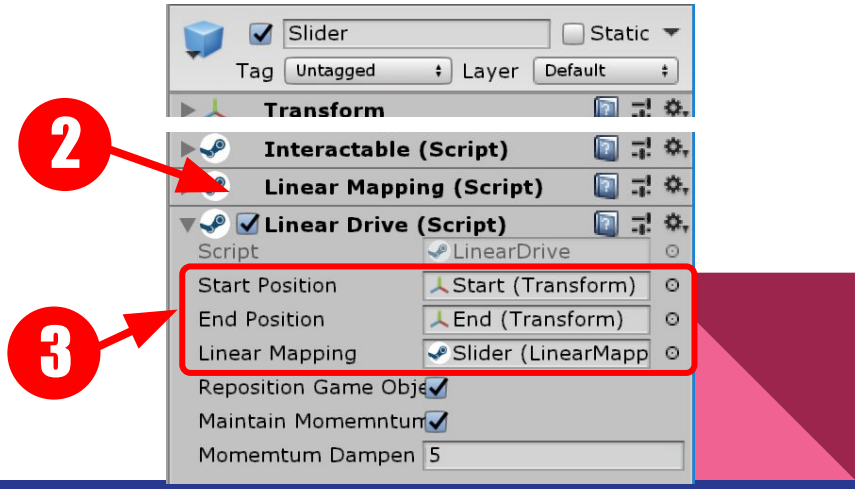
Controle Interativo

Usar o Linear Drive da aula 5 (1)

No controle de Slider, adicionar um componente Linear Mapping (2) e arrastar para o campo do Drive (3)

O LinearMapping converte um determinado movimento de Drive em um valor percentual (0.0 a 1.0)





Namespaces

Para acessar o valor do componente “LinearMapping” primeiro precisamos especificar em nosso script onde encontrar esta classe. Isto pode ser feito com o uso dos “namespaces”

Namespace é um modo de organizar coleções de classes com o uso de prefixos, como um sistema de endereços hierárquico.

Por exemplo, só conseguimos utilizar a classe Animator porque havíamos especificado onde encontrá-la (no namespace “UnityEngine”). Veja o que acontece quando comentamos o namespace.

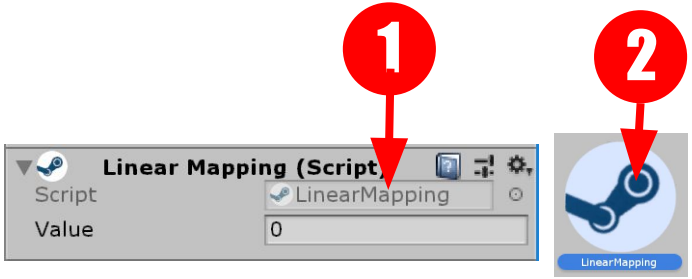
```
using UnityEngine;

Unity Script | 0 references
public class FanControl : MonoBehaviour
{
    public Animator anim;
```

```
//using UnityEngine;

0 references
public class FanControl : MonoBehaviour
{
    public Animator anim;
```

Descobrindo o namespace



Em Inspector, clicar em LinearMapping (1) e em Projects, selecionar o script (2)

Em Inspector, será exibido um preview do código. Observar o campo namespace (3)

No script referenciar o namespace no início (4) - ou alternativamente direto na variável (5)

```
Filename                    Assembly-CSharp.dll
//======= Copyright (c) Valve Corporation, All rights res
===============
//
// Purpose: A linear mapping value that is used by other co
//
//=================================
================================

using UnityEngine;
using System.Collections;

namespace Valve.VR.InteractionSystem
{
```

3

4

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Valve.VR.InteractionSystem;
```

ou 5

```
public bool oscillator;
public int power;
public Valve.VR.InteractionSystem.LinearMapping mapping;
```

Conversão de Valores

Adicione a variavel “mapping” (1)

Acrescente a linha (2) que converte o valor float 0.0 a 1.0 para valor Inteiro de 0 a 3

Salve o Script (CTRL+S)

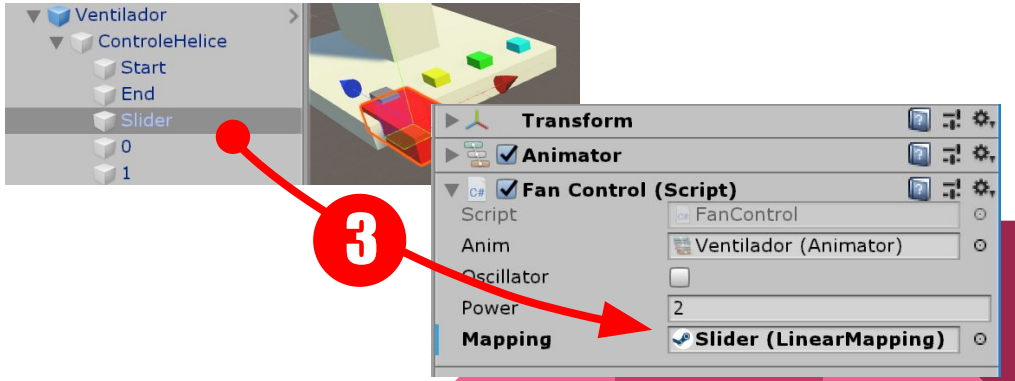
Arraste o Slider para o campo “Mapping” (3) e testar

```
public class FanControl : MonoBehaviour
{
    public Animator anim;
    public bool oscillator;
    public int power;
    public LinearMapping mapping;

    void Update()
    {
        power = Mathf.RoundToInt(mapping.value * 3.0f);
        anim.SetInteger("FanControl", power);
```

1

2



```
public class FanControl : MonoBehaviour
{
    public Animator anim;
    public bool oscillator;
    public int power;
    public LinearMapping mapping;

    void Update()
    {
        power = Mathf.RoundToInt(mapping.value * 3.0f);
        anim.SetInteger("FanControl", power);
```

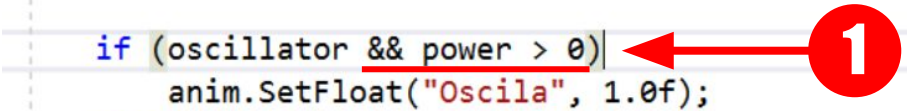
Ativando Oscilação com o Power

Adicionar nova condição para oscilação (1)

Testar e observar que agora a oscilação também é dependente da energia.

```
void Update() {
    power = Mathf.RoundToInt(mapping.value * 3.0f);
    anim.SetInteger("FanControl", power);

    if (oscillator && power > 0)
        anim.SetFloat("Oscila", 1.0f);
    else
        anim.SetFloat("Oscila", 0.0f);
}
```



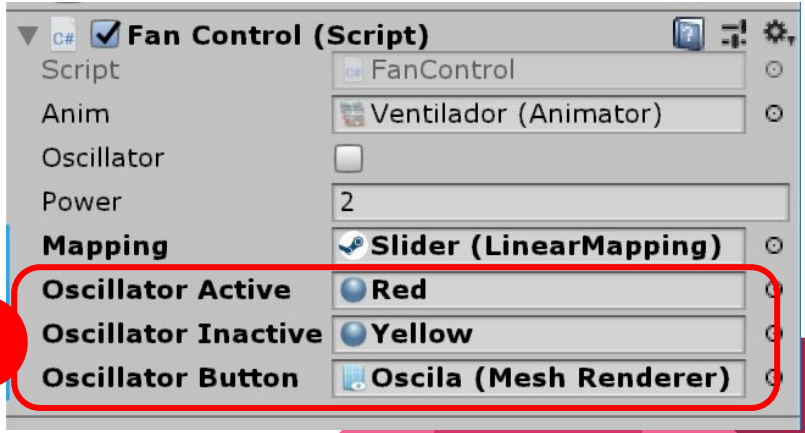
Material & User Feedback 1/2

Criar 2 variáveis tipo Material e 1 variável tipo MeshRenderer (1)

Arrastar materiais e o objeto “Oscila” para os campos no Inspector (2)

```
public LinearMapping mapping;
public Material oscillatorActive;
public Material oscillatorInactive;
public MeshRenderer oscillatorButton;
```

1

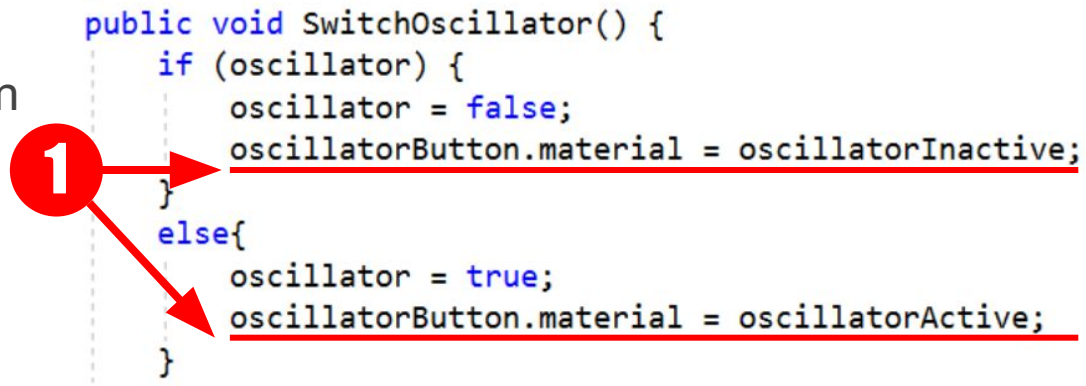


2

Material & User Feedback 2/2

Adicionar troca de materiais no botão de oscilação (1)

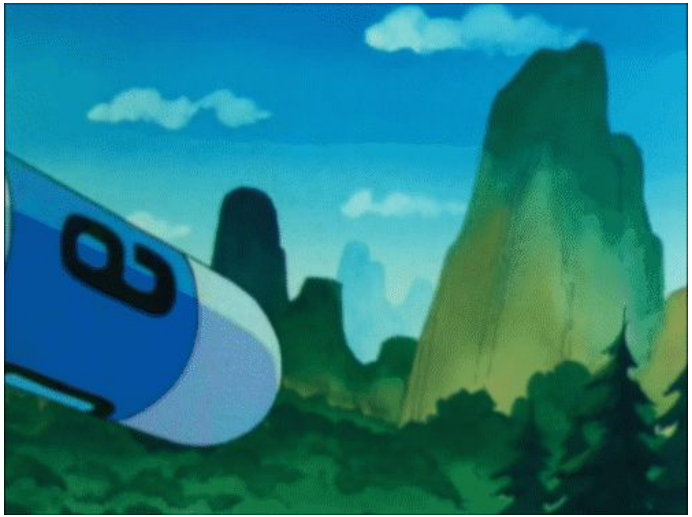
Testar controle de oscilação com power em diferentes valores



Bônus Script!

Objeto que instancia um Prefab ao colidir com outro objeto

(Capsulas Dragonball ou Pokebola)



CollisionInstance

Criar o script como ao lado (1).

Aplicá-lo em uma primitiva e adicionar também RigidBody (2)

ActivationObject : objeto que ativará o script (ex: Chão)

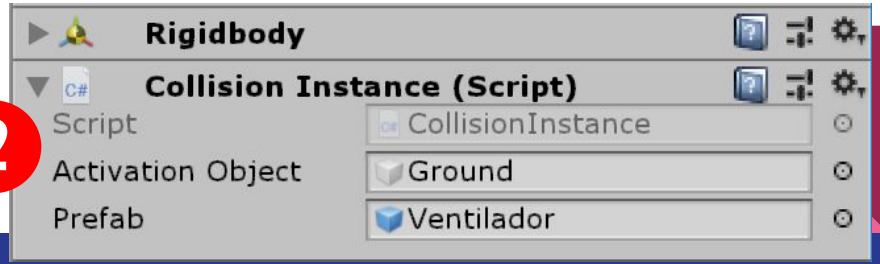
Prefab : qual objeto deverá ser instanciado

```
public class CollisionInstance : MonoBehaviour
{
    public GameObject ActivationObject;
    public GameObject prefab;

    void OnCollisionEnter(Collision collision){
        if (collision.gameObject == ActivationObject) {
            Instantiate(prefab,
                gameObject.transform.position,
                gameObject.transform.rotation);
            Destroy(gameObject);
        }
    }
}
```

1

2



Recapitulando...

Criamos nossos primeiros componentes por meio de scripts. Você aprendeu a usar funções básicas do Visual Studio e rudimentos de programação.

Lembra-se dos vínculos entre componentes discutidos na aula anterior ? Nesta aula aprendemos como permitir vínculos com nossos componentes simplesmente ao expor variáveis públicas.

Cultive a prática de criar essas referências, ao invés de tentar fazer tudo via código. É muito importante criar componentes que podem ser reutilizados, tanto em seus projetos, como por outras pessoas de seu grupo de trabalho. Portanto evite criar relações diretas entre componentes dentro do código !

Recapitulando...

O Script FanControl foi colocado no objeto Ventilador e precisa do vínculo com o Animator, para ter acesso ao controle das animações deste objeto.

Criamos uma booleana para armazenar se a oscilação está ligada ou não. O código verifica a cada quadro o estado desta booleana e altera o parâmetro “Oscila” do Animator se seu estado for *true*. Para ativar a booleana por meio do botão criado em aula anterior, que já continha o componente Hover Button, bastou selecionar na função OnButtonDown o objeto Ventilador e o método SwitchOscillator.

Recapitulando...

Depois utilizamos o slider interativo para controlar a velocidade do ventilador. Para isso adicionamos um Linear Mapping vinculado ao Linear Drive, assim podemos converter a posição do slider em um valor de 0 a 1. No Script FanControl do Ventilador expomos uma variável pública para vincular esse Linear Mapping do slider, convertemos o mapeamento de 0 a 1 para 0 a 3 e usamos o valor para controlar a variável FanControl do Animator. Agora o slider controla a velocidade.

A última ação foi alterar a cor do botão da oscilação e para isso apenas expomos variáveis de material e do objeto utilizado como botão e alteramos a função que controla a oscilação para alterar os materiais de acordo.