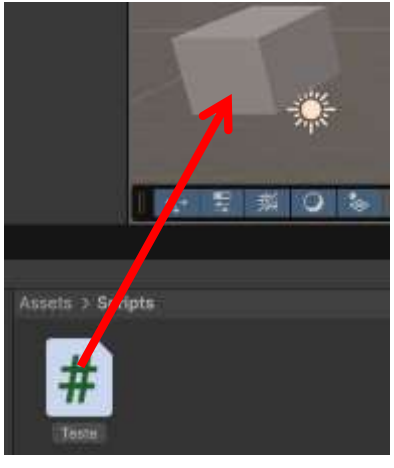
Introdução Ao Desenvolvimento de Jogos

Inserindo Scripts

Prof. Rone Ilídio da Silva

Inserindo e Associando Scritps

- O Unity utiliza C#
- Exemplo simples:
  - Crie um projeto novo
  - Em *Project*, crie uma pasta chamada Scripts
  - Selecione a pasta e com o botão direito clique em Create→Monobehaviour Script
  - Dê o nome de Teste
  - Insira um cubo na tela
  - Arraste o script criado até o cubo
- Dê 2 cliques no script para editar seu código



Script Básico

- Código de um script básico criado pelo Unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Teste: MonoBehaviour
{
        // Start is called before the first frame update
        void Start()
        {   }
        // Update is called once per frame
        void Update()
        {   }
}
```

Evento executado uma vez no início

Evento executado a cada frame

Escrevendo no console

- Modifique o método Start() e Update() para:

```
void Start()
{
	print("Texto exibido no console");
	Debug.Log("Debug também exibido");
}
void Update()
{
	Debug.Log("Debug também exibido");
}
```

Execute e veja o Console

Um pouco de teoria!

- “Teste” é uma classe filha de MonoBehaviour
- MonoBehaviour classe que possui métodos executado a partir de eventos nos objetos, como Start() e Update()
- Somente scripts com classes filhas de MonoBehaviour podem ser associadas a um GameObject

Introdução ao C#

- C# possui a mesma sintaxe C para comandos e do Java orientação a objetos

- A seguir, um visão geral da linguagem

Atributos

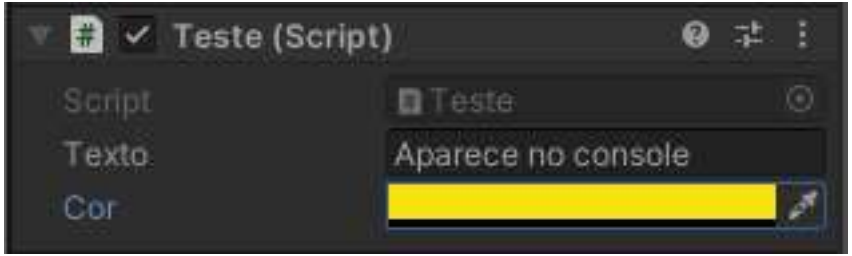
Atributos

- São as “variáveis” criadas dentro de um classe
- Atributos públicos podem ser acessados pelo *Inspector*
- Altere o código da classe Teste para o seguinte:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Teste : MonoBehaviour
{
    public string texto;
    public Color cor;

    // Start is called before the first frame update
    void Start()
    {
        Debug.Log(texto);
        GetComponent<Renderer>().material.color = cor;
    }
}
```

Selecione o cubo e veja no *Inspector*



Obs: só aparecem atributos públicos, privados não

Execute

Tipos Inteiros

Type	Size (in bits)	Range
sbyte	8	-128 to 127
byte	8	0 to 255
short	16	-32768 to 32767
ushort	16	0 to 65535
int	32	-2147483648 to 2147483647
uint	32	0 to 4294967295
long	64	-9223372036854775808 to 9223372036854775807
ulong	64	0 to 18446744073709551615
char	16	0 to 65535

Tipos de Ponto Flutuante

Type	**Size (in bits)**	**precision**	**Range**
float	32	7 digits	1.5×10^{-45} to 3.4×10^{38}
double	64	15-16 digits	5.0×10^{-324} to 1.7×10^{308}
decimal	128	28-29 decimal places	1.0×10^{-28} to 7.9×10^{28}

Tipo decimal

- É um float com menor *range* e maior precisão.
- Utilizado sempre o sufixo m ou M
- Exemplo (escreva o código dentro do método Start()):

```
decimal d = 3.55M;//ou 3.55m
double x =  (double) d;
print("Valor de d: "+d);
print("Valor de x: "+x);
```

Tipo string

- Conjunto de caracteres como em qualquer linguagem
- Nome do tipo em minúsculo
- Utiliza aspas duplas “”

Operadores

Categoria (by precedence)	**Operadores**
Unary	+ - ! ~ ++x --x (T)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational	< > <= >= is
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||

Comandos

Comandos

- Idênticos ao Java/C:
  - if/else
  - for
  - while
  - do/while
  - switch
  - break
  - continue

Camando foreach

- Exemplo:

```
string [] nomes = {"Ana", "Maria", "José"};
foreach (string n in nomes){
    print(n);
}
```

Métodos

Métodos

- São funções declaradas dentro de uma classe
- Inicialmente, todos os métodos serão públicos
- Modifique o código de Teste.cs para:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Teste : MonoBehaviour
{
    public int fatorial(int n){
        int f = 1;
        for(int i=1; i<=n; i++){
            f = f * i;
        }
        return f;
    }
    void Start()    {
        int numero = 5;
        int fat = fatorial(numero);
        print("O fatorial de "+numero+" é "+fat);
    }
}
```

Tratando Eventos do Mouse

Tratando Eventos do Mouse

- Crie um projeto novo (ou utilize o anterior)
- Insira um cubo
- Insira um script (pode dar o nome de Teste)
- Associe o script ao cubo
- Modifique a classe Teste de acordo com o próximo slide

Tratando Eventos do Mouse

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Teste: MonoBehaviour
{
        // Start is called before the first frame update
        void Start()
        {   }
        // Update is called once per frame
        void Update()
        {   }
         void OnMouseDown(){
              Debug.Log("Clicou no Cubo");
              GetComponent<Renderer>().material.color = Color.red;
         }
}
```

Apague os conteúdos dos métodos Start e Update()

Trata o evento de clique do mouse (botão esquerdo pressionado)

Pega a referência do objeto

Execute e clique no cubo

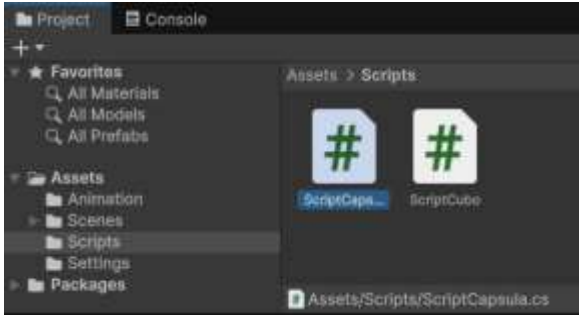
Tratando Eventos do Mouse

- Principais métodos para tratamento de eventos do mouse:
  - OnMouseDown: botão esquerdo pressionado
  - OnMouseUp: botão esquerdo liberado
  - OnMouseDrag: arrasto
  - OnMouseEnter: entrou com o ponteiro
  - OnMouseExit: saiu com o ponteiro
  - OnMouseOver: ponteiro sobre

Acessando Outras Classes

Acessando Funções em Outro Script

- Crie um novo projeto (ou mantenha o anterior apagando o cubo)
- Insira um cubo (novo)
- Associe ao cubo um novo script chamado ScriptCubo
- Insira uma cápsula
- Associe à cápsula um novo script chamado ScriptCapsula
- Veja os códigos:



ScriptCapsula

```
using UnityEngine;

public class ScriptCapsula : MonoBehaviour
{
    public void mudarCor()
    {
        GetComponent<Renderer>().material.color = Color.yellow;
    }
}
```

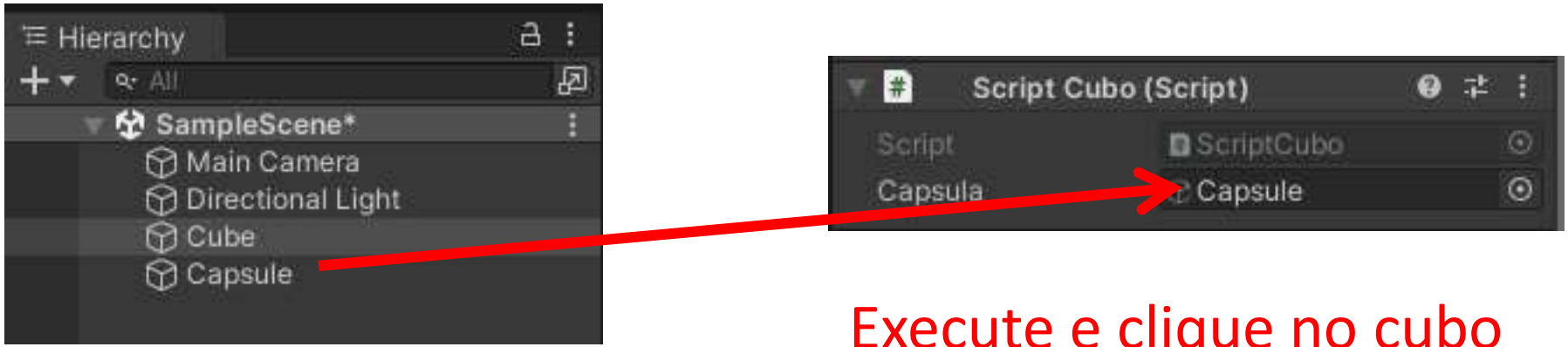
ScriptCubo

```
using UnityEngine;

public class ScriptCubo : MonoBehaviour
{
    public GameObject capsula;
    public void OnMouseDown(){
        capsula.GetComponent<ScriptCapsula>().mudarCor();
    }
}
```

Associe o Object Capsula em ScriptCubo

- Associe o objeto capsula ao atributo capsula em ScriptCubo
  - Selecione o Cubo

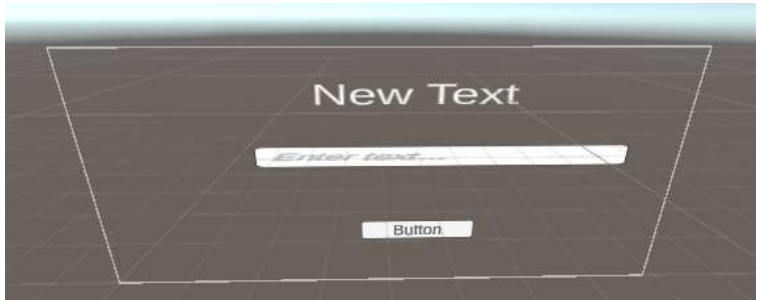


Execute e clique no cubo

Recebendo Dados do Usuário pela Interface 2D

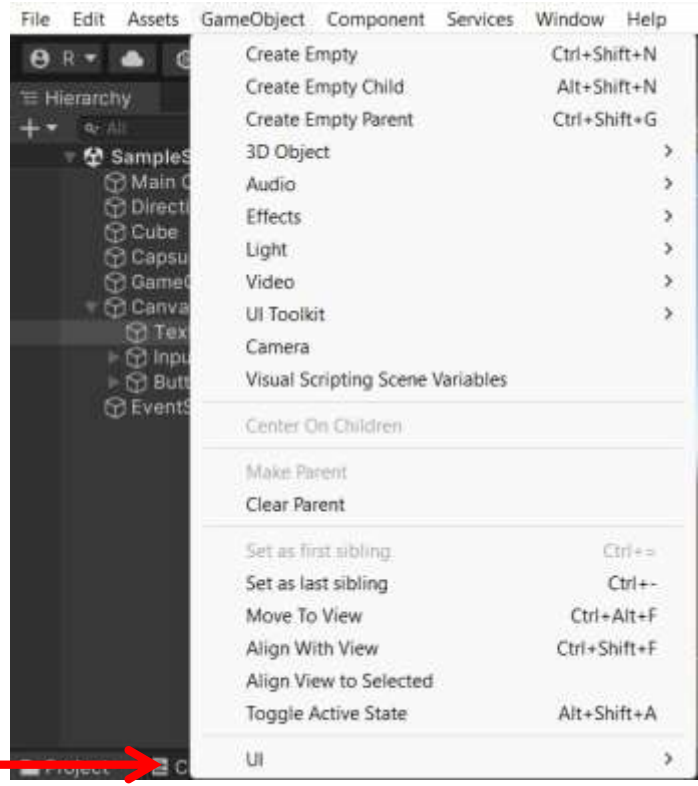
Recebendo Dados do Usuário pela Interface 2D

- Crie um novo projeto (pode manter o anterior mas apague tudo)
- Insira os seguintes GameObjects:
  - GameObject/UI/Text – TextMeshPro
  - GameObject/UI/Input Fiel – TextMeshPro
  - GameObject/UI/Button – TextMeshPro
- Em Project, dê dois clique no objeto Canvas (atenção com o Zoom para conseguir ver o Canvas todo)
- A tela criada deve ficar como na figura:



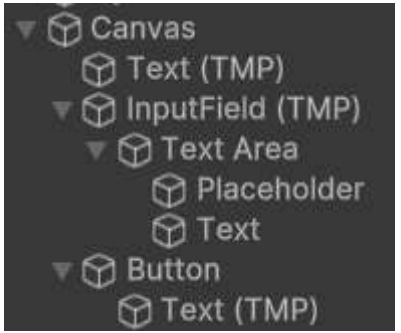
Recebendo Dados do Usuário pela Interface 2D

- Atenção: se a opção UI não aparecer no Menu GameObjet, siga os passos:
  - Crie um GameObject vazio (GameObject/Create Empty)
  - Selecione o objeto vazio
  - Clique no menu GameObject → UI deve aparecer no final



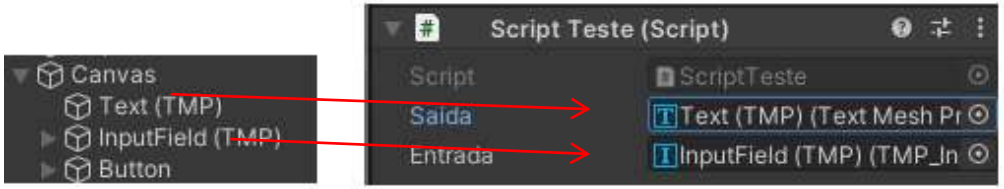
Recebendo Dados do Usuário pela Interface 2D

- Atenção para a hierarquia de objetos
  - Os componentes 2D são filhos de Canvas
  - O Button possuem um Text como filho
  - InputField possui dois Text como filho
    - Placeholder → dica quando o componente está vazio
    - Text → texto digitado pelo usuário



Recebendo Dados do Usuário pela Interface 2D

- Crie uma pasta chamada Script
- Crie um script chamado Script_Test
- Crie um GameObject vazio e dê o nome de _ButtonActions
- Associe Script_Test a _ButtonActions
- Insira o seguinte código em Script_Test (próximo slide)
- Selecione _ButtonActions e associe:
  - entrada ao InputTField (TMP)
  - saida ao Text (TMP)



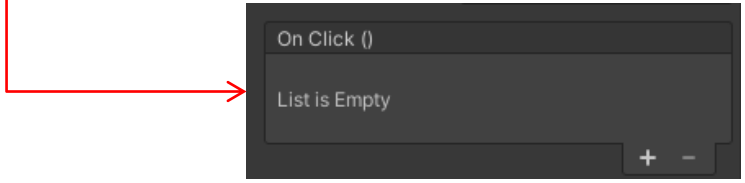
Script_Test.cs

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
public class Script_Test : MonoBehaviour {
    public TextMeshProUGUI saida;
    public TMP_InputField entrada;
    public void click(){
        saida.text = entrada.text;
        Debug.Log(“Funcionou!");
    }
}
```

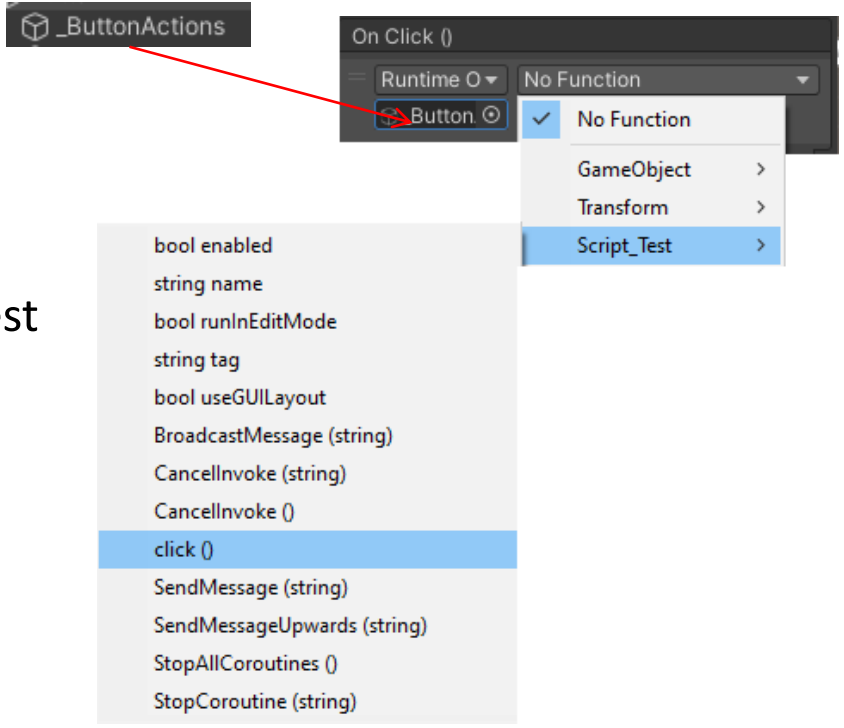
Atenção com using

Script_Test.cs

- Selecione o objeto do botão
  - Adicione um elemento à lista
  - Associe o objeto _ButtonActions ao evento OnClick()
  - Escolha o método click() de Script_Test



Execute

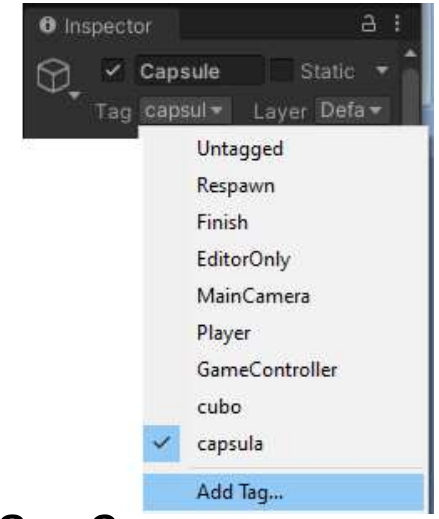


Utilizando TAGs

TAGs

- Tags são textos associados a GameObjects
- Podem ser utilizadas para definir categorias ou classes de objetos
- Exemplo:
  - A tag “player” é associada a todos os jogadores
  - A tag “parede” é associada a todos os objetos que não podem ser ultrapassados.
- O texto da tag pode ser acessado via código

TAGs



- Crie um novo projeto
- Insira 10 cubos
- Selecione um deles
  - No Inspector , crie as tags 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9
- Associe um cubo a cada uma das tags
- Crie um script chamado ScriptCubo
  - Código no próximo slide

ScriptCubo

```
using UnityEngine;
using TMPro;
public class ScriptCubo : MonoBehaviour {
    public TextMeshProUGUI display;
    public static string dado = “”;
    public void OnMouseDown(){
        dado = this.tag;
        display.text += dado;
    }
}
```

TAGs

- Associe o ScriptCubo a cada um dos cubos
- Adicione um GameObject/UI/Text – TextMeshPro
- Ajuste a câmera e a posição do TextMeshPro
- Associe o TextMeshPro ao atributo display de cada um dos scripts nos cubos

Repita para todos os cubos e execute