

# Arquitetura da programação em Unity e primeiros passos

Flávio Roberto Dias Silva

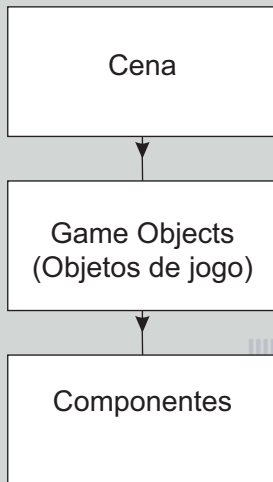
Encontro 1

**Desenvolvimento de jogos em Unity 3D**

Universidade Estadual do Oeste do Paraná



# Arquitetura da programação

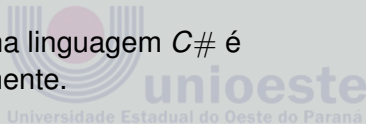


## Tipos de Componentes

Entre os principais tipos de componentes estão

- Transform
- Rigidbody
- CharacterController
- Colliders
- Camera
- Nossos Scripts

A programação do jogo que será feita na linguagem C# é adicionada a GameObjects como componente.



# Linguagem de programação dentro da Unity

`C#` é uma linguagem de programação orientada a objetos criada pela Microsoft, faz parte da sua plataforma .Net. A companhia baseou `C#` na linguagem C++ e Java.



# Opções de Layout

- 2by3
- 4Split
- Default
- Tall
- Wide



# Interface

- **Hierarchy:** Lista todos os gameObjects presentes na Cena Atual.
- **Scene:** Nessa aba temos uma representação visual da cena atual, sendo possível fazer edições nos gameObjects.
- **Game:** Essa aba mostra uma visão da camera de jogo, é a visão de como o jogo estará rodando para o jogador.
- **Project:** Essa aba mostra uma pasta no computador onde são guardados todos os elementos adicionais do jogo, imagens, modelos 3D, materiais, texturas etc...

- **Console:** Mostra mensagens de Debug da Unity.
- **Inspector:** Lista os Componentes do gameObject selecionado onde podemos editar alguns campos especificos dos componentes.



Existem vários caminhos para se criar um script em Unity. Uma delas é clicar com o botão direito na aba project ou em alguma de suas pastas e utilizar a opção Create -> Csharp Script.

O script criado dessa maneira vem com algumas linhas de código escrita.






O script adiciona duas bibliotecas: UnityEngine e System.Collections.

Uma classe com o nome escolhido pelo programador que utiliza a herança da classe MonoBehaviour.

Dois Metodos pertencentes a classe MonoBehaviour, são eles, Start e Update.

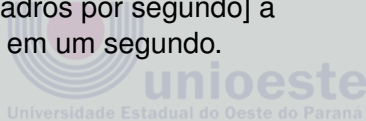


```
1   using UnityEngine;
2      using System.Collections;
3
4      0 references
5      public class UmScript : MonoBehaviour
6      {
7          // Use this for initialization
8          0 references
9          void Start()
10         {
11         }
12
13         // Update is called once per frame
14         0 references
15         void Update()
16         {
17         }
18     }
19
```

## Métodos

**Start:** A função Start é chamada no momento em que o objeto com o script entrar em cena. Se ele estiver presente na cena quando o jogo iniciar a função Start é chamada após a cena ser carregada.

**Update:** A função Update é chamada uma vez por quadro. Por exemplo: se o jogo roda a 30fps[quadros por segundo] a função Update será chamada 30 vezes em um segundo.



## Bibliotecas

**UnityEngine** é a biblioteca com os elementos internos da engine, como por exemplo as estruturas **Vector2**, **Vector3**, **Quaternion** e as classes **Input**, **Transform**, **Rigidbody**, **Collider**, **GameObject** entre muitas outras.

**System.Collections** é uma biblioteca nativa do *C#* responsável por coleções básicas com um conjunto de classes, estruturas e interfaces dedicadas a isso.



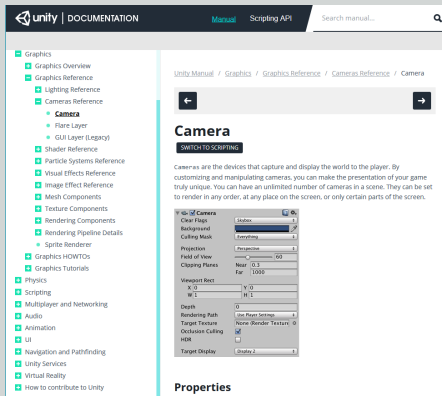
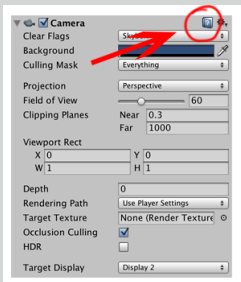
## A classe MonoBehaviour

A classe MonoBehaviour faz parte da biblioteca UnityEngine e é a classe responsável por tudo o que acontece em tempo de jogo, como por exemplo as atualizações a cada quadro, detecção de colisão, inserir e destruir objetos de jogo na cena, também entre muitas outras funções.



# Referencias

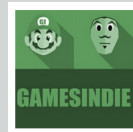
Podemos acessar a documentação de um componente da Unity clicando na interrogação posicionada a direita do nome do componente.



## Referencias no Youtube



Canal WeMakeAGame ->  
Paulo.



Canal GamesIndie -> Bruno.

São dois canais dedicados a ensinar a programação de jogos com Unity 3D.



## Levantamentos a cerca de POO

**POO:** Programação Orientada a Objetos.

**O que é um Objeto?** Objeto é a instancia de uma classe.

**O que é uma classe?** Uma classe, em geral, é um conjunto de Métodos(Funções) e propriedades(atributos) que definem um comportamento no desenvolvimento de um software.



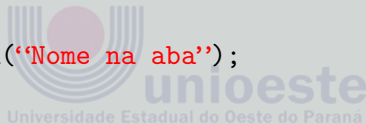


## Levantamentos a cerca de GameObjects

**GameObject:** É uma classe da biblioteca UnityEngine com métodos e propriedades voltadas a programação do jogo. Todo elemento que consta na aba Hierarchy é um objeto do tipo GameObject.

**Importante:** A partir de um script qualquer, para encontrar um GameObject presente na cena temos de usar um dos métodos estáticos presentes na classe GameObject, por exemplo:

```
GameObject G = GameObject.Find("Nome na aba");
```



## Primeiro Exercícios:

**Exercício:** Crie um script na Unity, clique com o direito na aba Project, criar um novo C# script. No script digite `GameObject`. e observe outras funções de busca por `GameObject` na programação em Unity.

**Exercício:** Observe os componentes na aba Hierarchy e tente encontra-los via Script, para certificar-se que ele foi encontrado utilize

```
print(nomeDaSuaVariavelDoTipoGameObject);
```

ou

```
Debug.Log(nomeDaSuaVariavelDoTipoGameObject);
```

## Encontrando Componentes

Com um GameObject selecionado na aba Hierarchy os componentes do mesmo aparecem na aba Inspector. Você pode selecionar um componente de um gameObject através de uma variável que guarda o GameObject e a função GetComponent. Sendo G a variavel que guarda o seu GameObject temos:

```
NomeDoComponent componente =  
    G.GetComponent<NomeDoComponent>();
```

onde NomeDoComponent representa o tipo de variável que em geral coincide com o nome do componente.

## Exercícios:

**Exercício:** Selecione componentes utilizando a função GetComponent, use o print ou Debug.Log para verificar que ele realmente foi selecionado.

**Exercício:** Modifique algumas propriedades de alguns dos componentes selecionados e verifique o efeito.



## Observação

**Observação:** Toda variável do tipo `GameObject` tem uma propriedade `transform` que guarda o transform associado ao `gameObject`.

Toda variável do tipo `Transform` tem uma propriedade `gameObject` que guarda o `gameObject` associado ao `transform`.



## Filiação:

Podemos criar parentescos entre `GameObjects/Transforms` utilizando a aba Hierarchy. Para isso basta arrastar um `GameObject` para cima de outro na aba Hierarchy.

Quando um `Transform` é filho de outro, todas as transformações (posição, rotação e escala) feitas no objeto pai também ocorrerão no objeto filho.



## Filiação:

**Observação1:** As transformações por filiação levam em consideração a distancia que a origem do objeto filho está da origem do objeto pai.

**Observação 2:** Quando um objeto é filho de outro dentro da cena o transform da aba Inspector mostra a posição relativa do filho em relação ao pai e não mais a posição que ele ocupa no mundo.



## Exercicio

**Exercicio:** Monte uma cena de testes. No menu suspenso GameObject adicione um terreno. No terreno adicione uma forma geométrica, pode ser um cubo, capsula, esfera etc... .

**Exercicio:** Coloque a Camera como filha da geometria escolhida no exercicio anterior.

**Exercicio:** Movimente a geometria na aba Scene e observe a visão da camera acompanhando na aba Game.





## A classe Input

**Input**: É uma classe da biblioteca UnityEngine responsável pelas entradas de informações, leitura do teclado, do mouse, do toque do celular, do joystick, etc...

Inicialmente utilizaremos as funções `GetKey`, `GetKeyDown`, `GetKeyUp`. As tres são funções que retornam um valor do tipo **bool**, verdadeiro quando apertamos uma tecla e falso quando não apertamos.



## A classe Input

Podemos utilizar

```
bool apertando = Input.GetKey('w');
```

e também

```
if(Input.GetKeyDown('w')){  
    // Suas Linhas de comando  
}
```



## A classe Input

**GetKey:** É verdadeiro enquanto o botão está pressionado.

**GetKeyDown:** É verdadeiro apenas quando se pressionou o botão.

**GetKeyUp:** É verdadeiro apenas quando se soltou o botão (tirou o dedo do botão).



**Observação** Também podemos utilizar como argumento da função `GetKey` e suas variantes um valor do tipo `KeyCode` que é um enumerador dos possíveis valores que podem ser utilizados como argumentos na função.

Por exemplo:

```
Input.GetKeyUp(KeyCode.W);
```

ou

```
Input.GetKeyUp((KeyCode)97);
```



## pré-definições da classe Input

A classe Input conta com pré-definições para acessar os botões por nomes dados pelo usuário e acessa-los na forma de eixos em vez de simplesmente botões.

As pré-definições se encontram no menu Edit -> Project Settings -> Input .  
As pré-definições de Input aparecerão no Inspector.



Observe que nas pré-definições existe uma opção para Positive Button e Negative Button. Essas opções nos permitem definir a leitura de eixos por meio da função:

```
Input.GetAxis (“nome do Eixo”);
```

Essa função retorna valores entre  $-1$  e  $1$  que dependem da intensidade que você acessa os inputs positivo e negativo que podem ser: movimentos do mouse, eixos de joystick ou mesmo teclas do teclado.



As prédefinições também podem ser utilizadas da forma padrão, retornando verdadeiro ou false e tem equivalentes as funções `GetKey`.

Temos:

`Input.GetButton("nome do comando")`; retorna verdadeiro quando estamos segurando o botão apertado.

`Input.GetButtonDown("nome do comando")`; retorna verdadeiro no momento em que apertamos o botão.

`Input.GetButtonUp("nome do comando")`; retorna verdadeiro no momento em que soltamos o botão que está apertado.

**Exercício:** Utilizando os levantamentos a cerca das classes `Input` e `Transform` faça a movimentação da geometria inserida no terreno com as teclas AWS D.





- [1] Barnes, D. J., Kölling, M.(2009), Programação Orientada a Objetos com Java. Uma introdução prática usando BLUEJ, 4ªed., Pearson Prentice Hall.
- [2] Battaiola, A. L. (2000). Jogos por Computador ? Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação In: XIX Jornada de Atualização em Informática. Curitiba: SBC, Julho/2000, v. 2. pp. 83 - 122
- [3] Battaiola, A. L.; Elias, N. C.; Domingues, R.G. et al (2002). Desenvolvimento de um Software Educacional com Base em Conceitos de Jogos de Computador In: XIII Simpósio Brasileiro de Informática na Educação. São Leopoldo: SBC, 2002, pp. 282-290.

- [4] Crua, E. W. G.; Bittencourte, J. R.(2005) Desenvolvimento de Jogos 3D: Concepção, Design e Programação. Anais da XXIV Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, pp.1313-1356, São Leopoldo, Brasil, Julho de 2005.
- Deitel, H. M., Deitel, P. J.(2010), Java: Como programar, 8ªed., Pearson Prentice Hall. Rio de Janeiro IMPA.
- [5] Freeman, E., Freeman, E.(2007), Use a Cabeça Padrões de Projetos, 2ªed., Rio de Janeiro Altabooks.
- [6] Sintes, A.(2002), Aprenda Programação Orientada a Objetos em 21 dias, São Paulo Makron Books.
- [7] Stellman, A.; Greene, J.(2011), Use a Cabeça! C#, Rio de Janeiro, AltaBooks.

- [8] Unity Technologies (2016)(A). Unity 3D User Manual [online]. Disponível em: [\[http://docs.unity3d.com/Manual/index.html\]](http://docs.unity3d.com/Manual/index.html) [[Acesso em 18/04/2016]
- [9] Unity Technologies (2016)(B). Unity 3D Community Forum [online]. Disponível em: [\[http://forum.unity3d.com/\]](http://forum.unity3d.com/) [[Acesso em 18/04/2016]
- [10] Unity Technologies (2016)(C). Unity 3D Online Tutorials [online]. Disponível em: [\[https://unity3d.com/pt/learn/tutorials\]](https://unity3d.com/pt/learn/tutorials) [[Acesso em 18/04/2016]
- [11] Unity Technologies (2016)(D). Unity 3D Community Wiki [online]. Disponível em:

[[http://wiki.unity3d.com/index.php/Main\\_Page](http://wiki.unity3d.com/index.php/Main_Page) ][Acesso em 18/04/2016]

