

Desenvolvimento com Motores de Jogos II

Aula 02 - Entradas do jogador e movimento

Apresentação

Olá, pessoal! Sejam bem-vindos à segunda aula de nossa disciplina. Antes de iniciarmos, vamos recapitular o que vimos na aula passada? Então, recapitulando, aprendemos a criar objetos primitivos no Unity 3D e também a modificar sua posição, rotação e escala na cena.

Na aula de hoje, iremos aprender como funciona o sistema de entrada (Input) no Unity 3D, o conceito de eixos virtuais, como utilizá-los em scripts C# e como criar um objeto que se move na cena de acordo com os comandos do jogador. Vamos lá?



Fonte: Adaptado de <http://www.olhargeek.com.br/empresa-cria-luvas-magneticas-que-simulam-a-forca-de-star-wars>.

Objetivos

Ao final desta aula, você deverá ser capaz de:

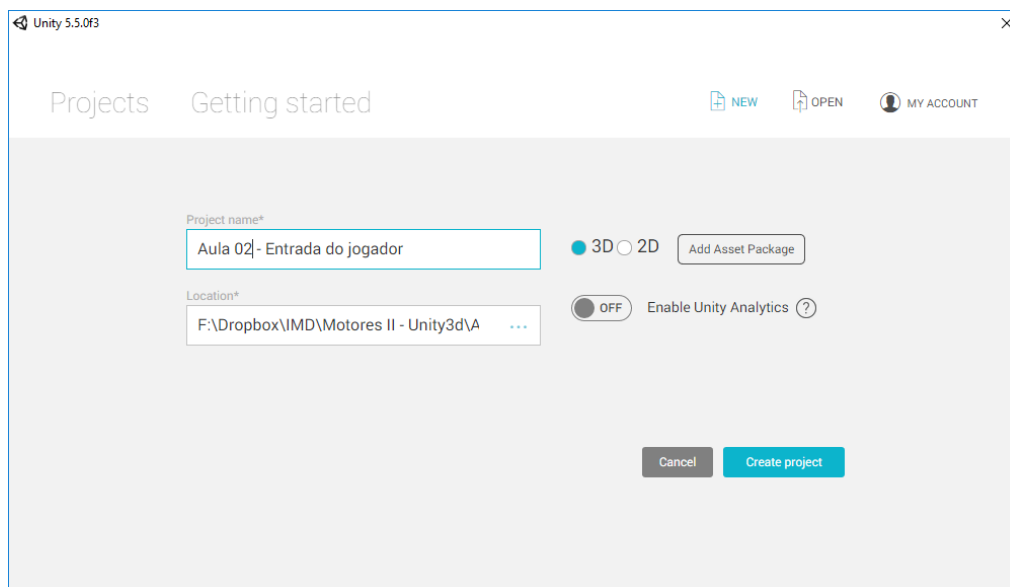
- Reconhecer os métodos básicos de entrada dos comandos do jogador (Input System)
- Compreender a configuração dos eixos virtuais
- Conhecer o movimento básico de objeto

Recebendo entrada do jogador

O Unity 3D suporta os tipos de entrada de dispositivos convencionais como teclado, mouse, joystick, etc. Além deles suporta entrada de diversos dispositivos mais avançados como touchscreen para dispositivos móveis. Inicialmente, nós vamos explorar os conceitos mais básicos de entrada pelo teclado/mouse usados a fim de criar jogos mais interativos, sendo esses conceitos elementos importantíssimos para a definição do modelo de jogos que pretendemos desenvolver, não é mesmo?

Antes de prosseguir, vamos criar um novo projeto chamado "Aula 02 - Entrada do jogador". Veja a **Figura 1**.

Figura 01 - Iniciando novo projeto



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Agora que criamos o projeto, veremos como um novo jogo no Unity vem configurado para tratar com as entradas do jogador através dos eixos virtuais, além de como modificá-los. Um pouco mais a frente, iremos aprender a criar eventos especiais nos quais o jogo reage a cada tipo de entrada.

Eixos virtuais

Os **eixos** são nomenclaturas especiais que cada projeto do Unity dá aos tipos de ações realizadas pelo usuário, por exemplo, mover horizontalmente, mover verticalmente, atirar, pular, entre outras.

Cada uma dessas ações tem um nome amigável e é mapeada para sua respectiva tecla do teclado, botão do joystick, etc. Assim você pode criar scripts que respondem, por exemplo, ao movimento no eixo horizontal (Horizontal), sem se preocupar se o jogador está usando um teclado, um joystick, o acelerômetro do celular ou qualquer outro dispositivo.

No que diz respeito aos scripts, cada eixo é acessado pelo seu nome. Apesar de você também ter a liberdade de especificar se deseja saber se uma tecla específica foi pressionada ou um botão específico do controle (joystick) foi ativado, é bem mais fácil criar scripts que simplesmente verificam se o jogador deseja atirar ou não. A fim de esse mapeamento de teclas->ações funcionar precisamos configurá-lo em cada projeto. Para facilitar, cada projeto do Unity já vem com uma configuração bem satisfatória, mapeando cada entrada dos dispositivos mais populares para sua ação correspondente.

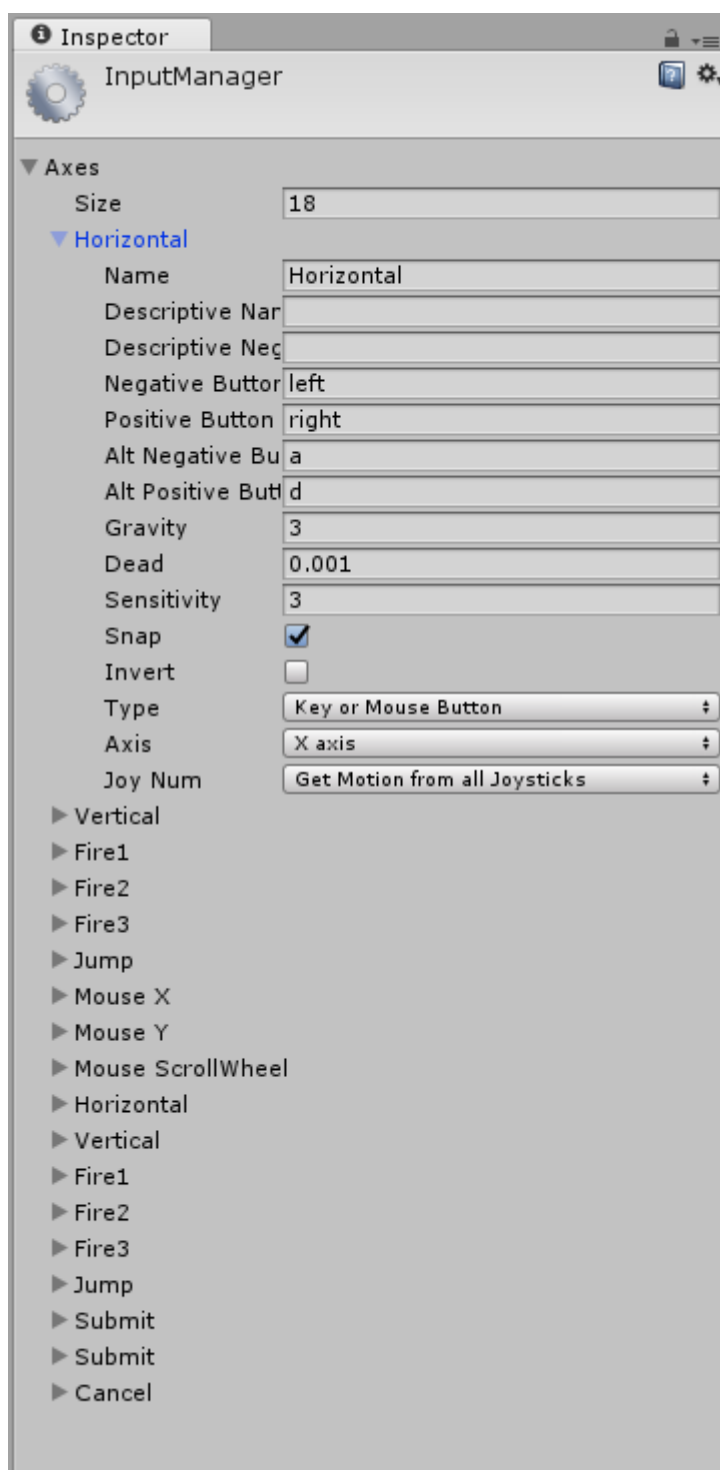
Alguns exemplos das entradas padrões de cada projeto criado no Unity são:

Atenção

- **Horizontal e Vertical** são mapeados para **W, A, S, D** e para as setas do teclado.
- **Fire1, Fire2, Fire3** são mapeados para Control, Alt e Shift e também para os três botões do mouse (pressionando o scroll temos um terceiro botão).
- **Mouse X e Mouse Y** são os movimentos do mouse.

O local onde verificamos/alteramos esse mapeamento é no “Input Manager”. Para acessar o Input Manager, busque o “Edit->Project Settings->Input” e um painel será aberto com um elemento chamado “Axes” no qual você pode clicar para expandi-lo e mostrar várias de suas opções. Dentre elas, clique em uma chamada “Horizontal”, expandindo-a também e revelando as propriedades dela. Veja na **Figura 2**.

Figura 02 - Input Manager.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Cada um dos “Axes” no Input Manager tem uma série de propriedades, as quais possuem funções distintas. Veja abaixo a lista.

▼	Name
	Nome usado para identificar o axis a partir de um script.
▼	Descriptive Name
	Descrição para o valor positivo desse axis. O valor positivo do axis Horizontal, por exemplo, é quando movimentamos para a direita, enquanto o negativo é para a esquerda.
▼	Descriptive Negative Name
	Descrição para o valor negativo do axis.
▼	Negative Button
	Botão utilizado para o axis na direção (valor) negativa.
▼	Positive Button
	Botão utilizado para o axis na direção (valor) positiva.
▼	Alt Negative Button
	Botão alternativo para a direção negativa.
▼	Alt Positive Button

Botão alternativo para a direção positiva.

▼ Gravity

Velocidade (em unidades por segundo) que o axis (eixo) volta para a posição neutra (zero) quando nenhum botão é pressionado.

▼ Dead

Tamanho da zona morta analógica no qual o dispositivo desconsidera qualquer tipo de movimento. Isso evita que ações indesejáveis sejam tomadas se um toque muito leve no controle ocorrer.

▼ Sensitivity

Velocidade (em unidades por segundo) que o axis se movimentará em direção ao alvo. Serve para dispositivos digitais, como o teclado, por exemplo.

▼ Snap

Se habilitado o valor do eixo vai ser imediatamente zerado caso seja pressionado o botão de direção oposta.

▼ Invert

Se habilitado, ocorre uma troca entre direções negativa e positiva. É útil quando, por exemplo, você deseja inverter os comandos de direção de um avião ou algo similar.

▼ Type

Tipo de entradas que irão controlar esse eixo.
▼ Axis
Eixos de um dispositivo conectado que irão controlar esse eixo virtual.
▼ Joy Num
Número do joystick que irá controlar esse eixo.

Quadro de notas

Para efeitos práticos iremos utilizar as configurações padrão do Unity para nossos projetos, mas você pode criar outros projetos e tentar modificar esses valores a fim de experimentar.

Usando eixos em scripts

Com o Input Manager configurado você pode fechar seu painel (clique com o botão direito sobre o seu nome e escolha a opção de fechar) para não ocupar espaço na sua área de trabalho. Lembre-se que essas configurações são válidas para cada projeto e se você criar um projeto novo elas são restauradas para os valores “padrão” nesse novo projeto. Isso é algo muito bom pois permite a você poder realizar vários testes em projetos separados.

Para utilizar os Inputs em um script você pode referenciá-los pelo nome utilizando o método **Input.GetAxis** (“nome”) assim:

```
float valorHorizontal = Input.GetAxis("Horizontal");
```


Isso fará com que a variável “valorHorizontal” receba o valor atual do eixo virtual Horizontal. Esse valor representa precisamente o quanto o jogador deseja se mover na horizontal. Valores positivos são para a direita e valores negativos para a esquerda (poderá alterar os sentidos, caso você tenha modificado no Input Manager).

Os valores são entre -1 e 1 e o valor neutro é 0. Existe uma exceção a isso relacionada a movimentos do mouse. Nesse caso os valores podem ser maiores que 1 e menores que -1 pois representam o quanto o jogador moveu o mouse desde o último frame.

Se você observou com cuidado viu que por “padrão” o Unity cria mais de um eixo virtual com mesmo nome. Por exemplo, existem dois eixos virtuais com o nome “Horizontal”. Isso é completamente possível e é um recurso muito poderoso, pois permite que você configure dois eixos virtuais para dispositivos diferentes, prevalecendo sempre o maior valor absoluto. Assim você pode criar um eixo Horizontal para o teclado e um outro com o mesmo nome para o joystick. Se o jogador estiver usando o teclado os valores retornados serão do teclado e se estiver usando o joystick os valores retornados serão desse dispositivo. Dessa forma você pode realizar a programação do jogo sem se preocupar se o jogador está utilizando um teclado ou um joystick para controlar o personagem.

Nomes dos botões

Para mapear uma tecla para um eixo virtual você precisa digitar o nome da tecla nas propriedades Positive Button ou Negative Button no eixo correspondente.

- Teclas normais: “a”, “b”, “c” ...
- Teclas numéricas (as que ficam acima das teclas normais): “1”, “2”, “3”, ...
- Setas do teclado: “up”, “down”, “left”, “right”
- Teclas do teclado numérico: “[1]”, “[2]”, “[3]”, “[+]”, “[equals]”
- Teclas de modificação: “right shift”, “left shift”, “right ctrl”, “left ctrl”, “right alt”, “left alt”, “right cmd”, “left cmd”

- Botões do mouse: "mouse 0", "mouse 1", "mouse 2", ...
- Botões do joystick (de qualquer joystick): "joystick button 0", "joystick button 1", "joystick button 2", ...
- Botões do joystick (de um joystick específico): "joystick 1 button 0", "joystick 1 button 1", "joystick 2 button 0", ...
- Teclas especiais: "backspace", "tab", "return", "escape", "space", "delete", "enter", "insert", "home", "end", "page up", "page down"
- Teclas de função: "f1", "f2", "f3", ...

Através de scripts C# você pode acessar o valor de uma determinada tecla com o método `Input.GetKey("nome da tecla")`, assim:

```
bool aPressionado = Input.GetKey ("a");
```

Observe que o método `Input.GetKey` retorna um valor booleano, ou seja, verdadeiro ou falso, indicando se a tecla está pressionada ou não.



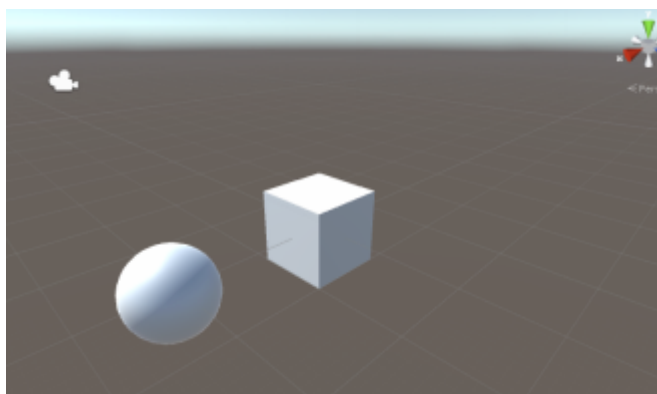
Fonte: Autoria própria.

Adicionando movimento em um objeto

Existem diversas maneira de se adicionar movimento aos objetos 3D no Unity a partir dos comandos do jogador. Vamos aprender as mais básicas inicialmente, que são comuns à maioria dos jogos.

Com seu projeto criado, adicionaremos um cubo e uma esfera na cena. Veja a **Figura 3**.

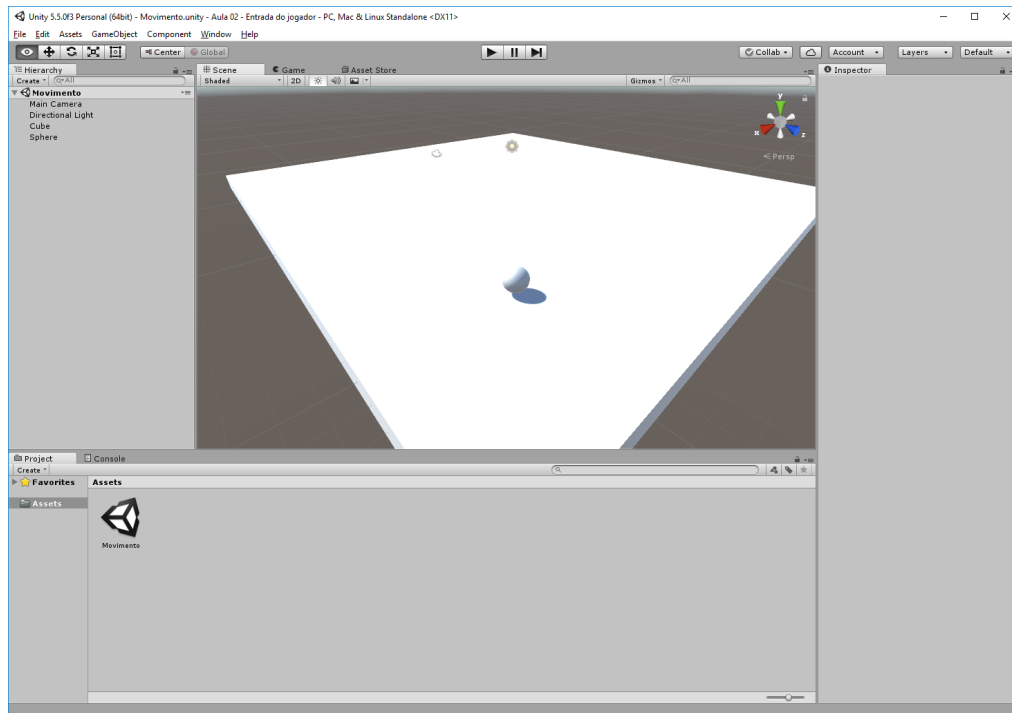
Figura 03 - Esfera e cubo adicionados



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017

Em seguida, utilizaremos as ferramentas estudadas para modificar suas dimensões e posições de modo que o cubo seja transformado em um “piso” para o nosso exemplo. Aproveite e salve sua cena com o nome “Movimento”. Posicione a esfera sobre esse piso como na Figura 4.

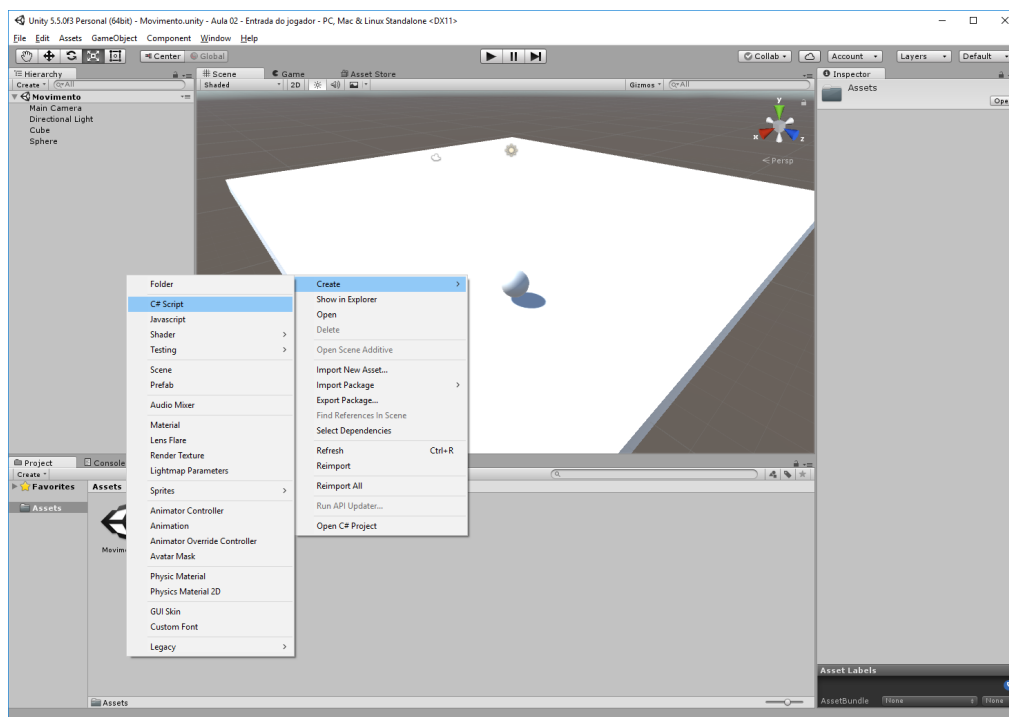
Figura 04 - Esfera sobre o piso.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Agora precisamos criar um script C# para podermos adicionar o comportamento de movimento na nossa esfera. Para isso clique em uma região em branco na aba "Project", que está devidamente com a pasta "Assets" selecionada, e escolha a opção Create->C# Script como na Figura 5.

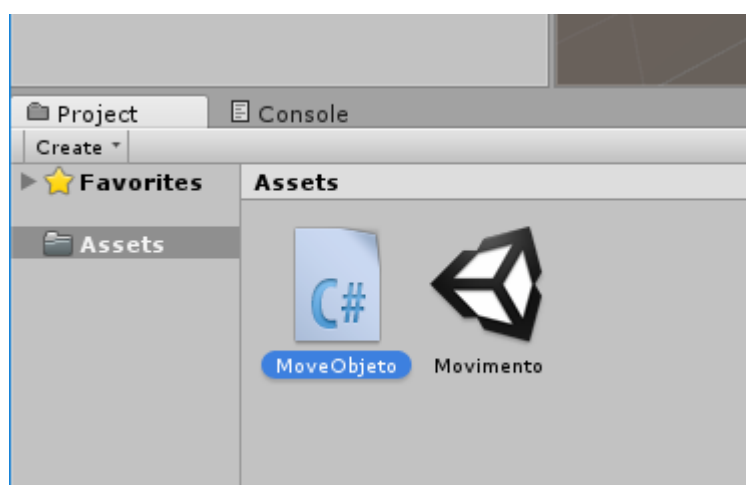
Figura 05 - Criando um script C#.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Nomeie o script para “MoveObjeto” e pressione Enter. Isso fará com que um script C# em branco (na verdade com um código padrão) seja adicionado nos seus Assets, os quais são os componentes do seu jogo. Na **Figura 6** você pode ver o script adicionado com sucesso na pasta Assets. Com o novo script criado clique duas vezes nele para abrir no MonoDevelop.

Figura 06 - Script C# adicionado ao projeto.



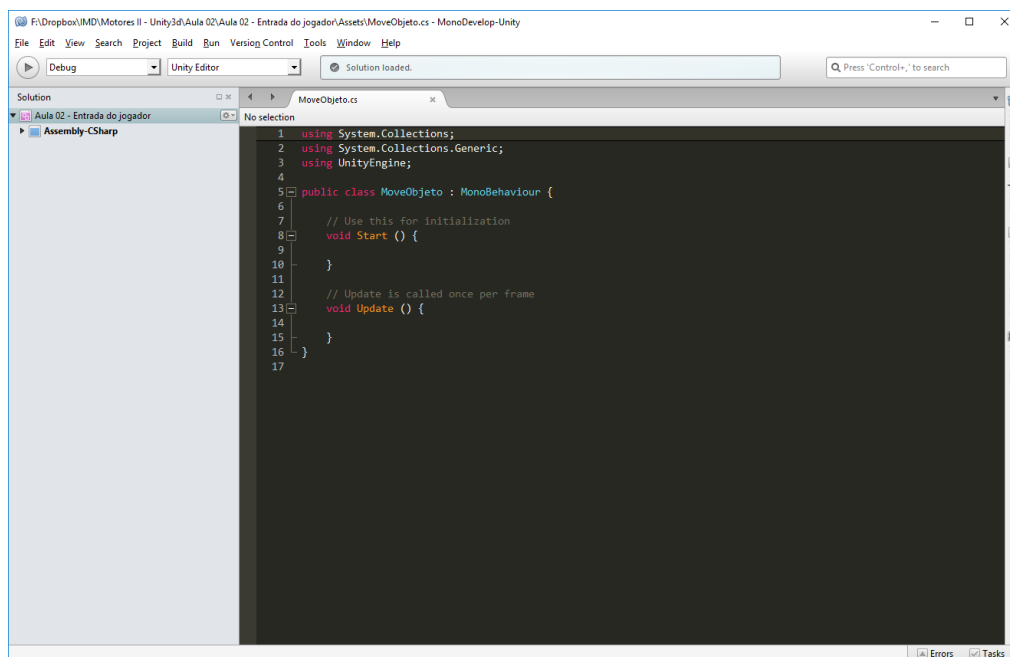
Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Atenção

O Unity suporta tanto o MonoDevelop quanto o Visual Studio como editor de código fonte. Nesse curso iremos utilizar o MonoDevelop mas todos os código apresentados também podem ser escritos no Visual Studio sem problema algum. Para modificar o editor padrão de códigos de Unity vá no menu Edit->Preferences, escolha a opção “External Tools” e lá a propriedade “External Script Editor” pode ser alterada entre o MonoDevelop e o Visual Studio (se esse estiver instalado na sua máquina). A escolha do editor é totalmente de sua livre opção.

Assim que você der um duplo clique no script MoveObjeto.cs ele abrirá no editor, como mostrado na **Figura 7**.

Figura 07 - Script aberto no editor MonoDevelop.

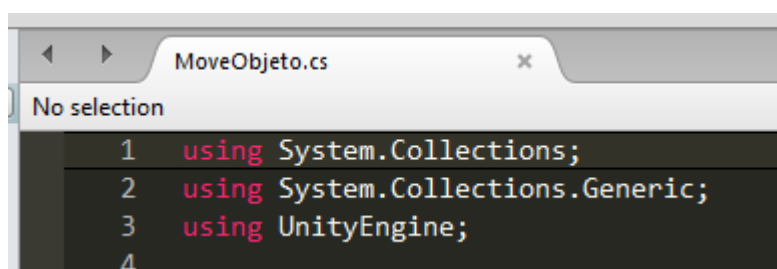


Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Observe que inicialmente o seu script já vem com algum código. Basicamente o Unity já adiciona três namespaces comumente utilizados com a diretiva “using”. Namespaces é um modo de organizar as classes em C# em grupos designados com um nome único, de modo a evitar ocorrerem problemas, como conflitos de nomes.

Usando a diretiva “using” em um arquivo .cs, você está informando ao compilador que deseja acessar as classes daquele namespace sem precisar digitar o caminho completo para se referenciar ao seu nome. Os namespaces adicionados são System.Collections, System.Collections.Generic e UnityEngine. Os dois primeiros permitem você utilizar funcionalidades da linguagem C# relativa ao uso de coleções, como listas, por exemplo. O UnityEngine adiciona os recursos específicos do motor de jogos Unity3D ao seu script para você poder utilizar suas funcionalidades. A **Figura 8** mostra em detalhes esses três namespaces.

Figura 08 - Namespaces padrão de um script C# no Unity.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
```

Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Após os namespaces o Unity adiciona uma nova classe que tem o mesmo nome do arquivo que você criou (sem a extensão, claro). Você não pode modificar o nome dessa classe (MoveObjeto) a não ser que também renomeie o arquivo para que o mesmo tenha o mesmo nome dela. A classe adicionada herda de MonoBehaviour, que é uma classe base com diversas funcionalidades do motor de jogos Unity3D e que permite que a mesma seja associada a objetos na sua cena.

Dentro da classe MoveObjeto por padrão são adicionados dois métodos (Start e Update). Esses métodos são especiais e são executados pelo Unity durante a execução do seu jogo em momentos diferentes. Veja a **Figura 9** destacando o código citado.

Figura 09 - Esqueleto padrão de uma classe C# em um script no Unity 3D.

```
5 public class MoveObjeto : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
17
```

Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Vamos falar mais dos métodos especiais dos scripts no Unity no decorrer do curso, porém é importante conhecer de imediato esses dois adicionados por padrão.

O método Start() é executado somente uma vez durante o seu jogo (no início), já o método Update() será executado repetidamente durante o seu jogo (uma vez a cada frame). Desse modo, é possível adicionar dentro do método Start() todo o código desejado para inicializar um determinado objeto (por exemplo sua posição inicial, quantidade de vida inicial de um personagem, etc.) que contém esse script. No método Update() colocamos todo o código executado durante o jogo, como a verificação acerca de o jogador ter pressionado algum comando para mudar a posição do objeto, sendo essa verificação justamente o que precisamos agora.

Assim, verificaremos inicialmente, dentro do método Update(), se o jogador deseja se movimentar no eixo horizontal. Para isso, utilizaremos o Input.GetAxis("Horizontal") a fim de obter um valor que representa o quanto ele deseja se mexer nesse eixo (valor de -1 a 1). Veja na **Figura 10** como obter esse valor e armazenar em uma variável float.

Figura 10 - Obtendo a intenção de movimento do jogador no eixo horizontal.

```
11
12    // Update is called once per frame
13    void Update () {
14        float movimentoHorizontal = Input.GetAxis ("Horizontal");
15    }
16 }
17
```


Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Lembre-se de não ser necessário mais de um comando para saber se o jogador deseja movimentar o objeto para a esquerda ou para a direita, já que o eixo virtual “Horizontal” retorna um valor float de -1 a 1 no qual os valores negativos indicam a esquerda e os positivos a direita. Se o jogador estiver utilizando um teclado, basta pressionar as teclas A, S ou as Setas para a direita ou esquerda a fim de o valor correto ser retornado. Se o jogador estiver utilizando um controle (joystick), ele pode utilizar o analógico do lado esquerdo para isso ou as setas digitais do controle. No caso do controle analógico, os valores representam o quanto ele moveu o analógico (se moveu metade para a esquerda, por exemplo, o valor retornado será -0.5 e se moveu todo para a esquerda será retornado -1).

Antes de mover nosso objeto vamos exibir o valor obtido no Unity para verificar se tudo está funcionando corretamente até agora. A forma mais prática de fazer isso é utilizando o Console de Debug.

Para isso digite o seguinte comando ainda dentro do Update() e logo abaixo do Input.GetAxis():

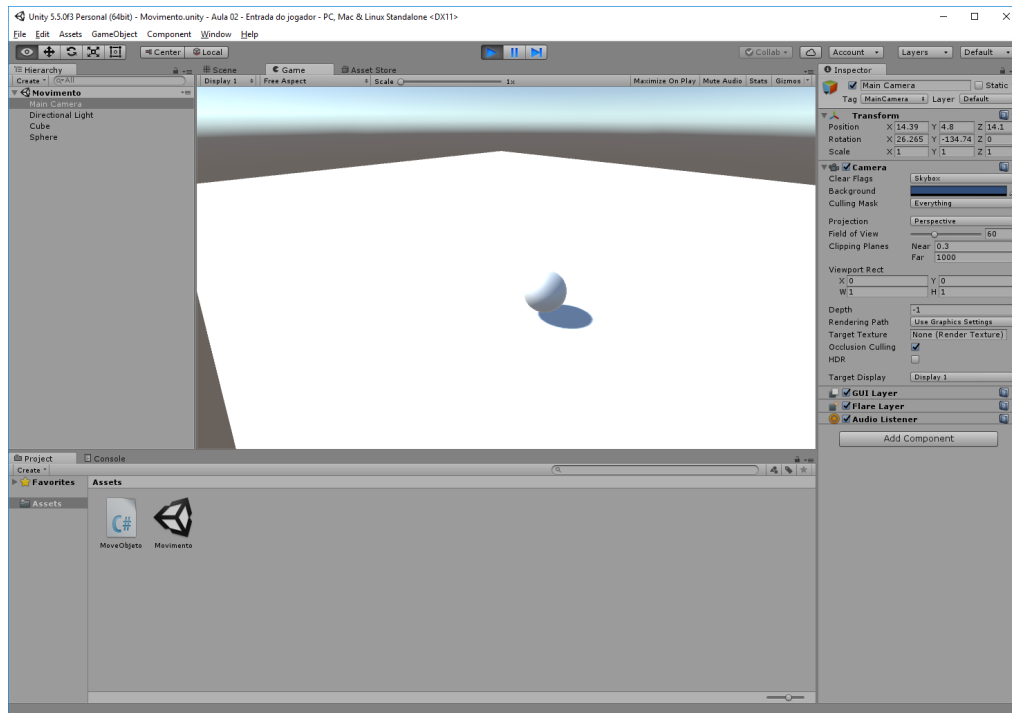
```
Debug.Log ("Valor horizontal: " + movimentoHorizontal.ToString ());
```

Seu código final do método Update() deve ficar assim:

```
1 void Update () {  
2     float movimentoHorizontal = Input.GetAxis ("Horizontal");  
3     Debug.Log ("Valor horizontal: " + movimentoHorizontal.ToString ());  
4 }
```

Salve o script no MonoDevelop por meio do atalho Control+S ou do menu File->Save, volte ao Unity, pressione “Play” e veja o que acontece, como na **Figura 11**.

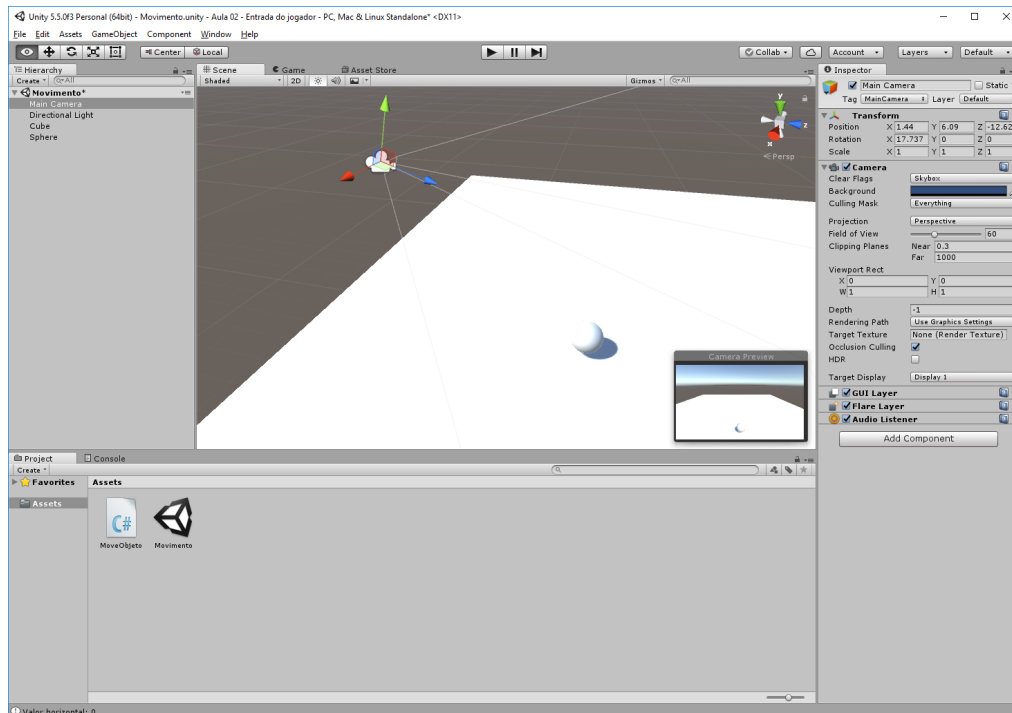
Figura 11 - Tentativa de executar o script C# criado.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Se não existe nenhum erro no script, o botão Play deve ter ficado azul (indicando que o jogo está rodando) e a primeira coisa que você deve ter notado é o foco da janela ter ido para a aba "Game" e lá ter sido exibida a visualização da câmera. Se ela não estiver muito bem posicionada, interrompa a execução do jogo (clicando novamente em Play), mova a câmera para ter uma perspectiva geral da cena e inicie novamente o jogo (Play). Veja na **Figura 12** um exemplo de posição de câmera. Lembre-se de que todo projeto no Unity já vem com uma câmera criada chamada "Main Camera" e ela pode ser acessada na aba Hierarchy.

Figura 12 - Posicionamento da câmera.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Um recurso muito útil do Unity é haver, enquanto você está com a câmera selecionada, uma pequena visualização exibida no canto inferior direito mostrando exatamente como a câmera está “observando” a cena.

Voltando ao movimento do objeto, se você lembrar bem, implementamos a função Update() do nosso script para ela obter repetidamente (uma vez por frame) o valor do eixo horizontal e exibi-lo no Console de Debug. Veja que ao lado da aba Project existe uma outra aba, chamada “Console”. Clique nela para ver se esse valor está aparecendo (**Figura 13**).

Figura 13 - Aba Console.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

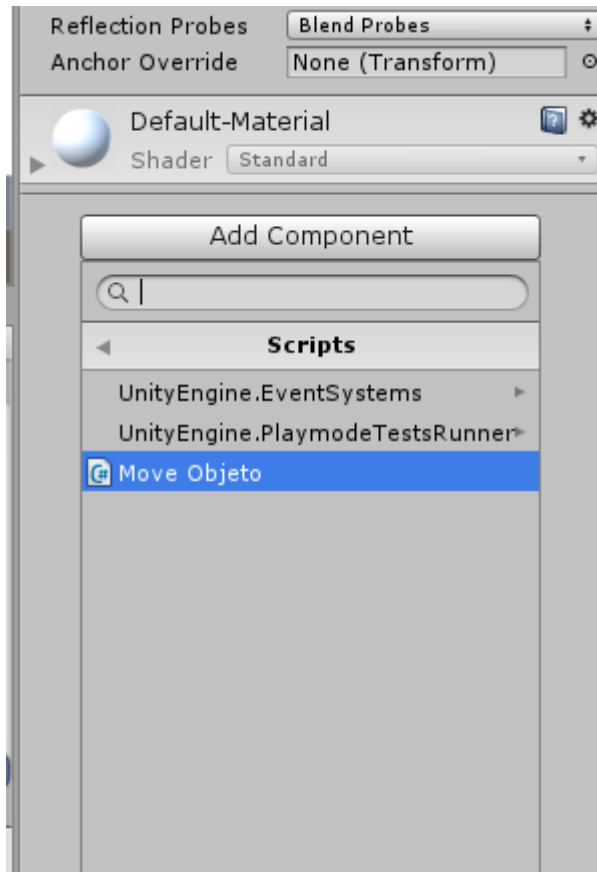
É... Parece que nada está funcionando, pois o Console não mostra nada! Não se preocupe, está tudo sob controle. Acontece que criamos um script C# no nosso projeto, porém não o associamos a nenhum objeto da cena. Dessa forma, esse script só existe como um Asset, mas não está sendo utilizado na cena atual do jogo. Para corrigir esse problema, adicionaremos o script na nossa esfera.

Atenção

Para realizar qualquer mudança permanente na cena, você DEVE interromper antes a execução do jogo. Se não fizer isso, todas as mudanças realizadas serão perdidas assim que você interromper a execução posteriormente.

Para adicionar o script na esfera, selecione-a no Hierarchy e, no Inspector, localize o botão "Add Component" (deve ser a última opção). Clique nele, escolha a opção "Scripts" e, em seguida, deverá aparecer uma opção chamada "Move Objeto", referente ao nosso script. Selecione essa opção como mostrado na **Figura 14**.

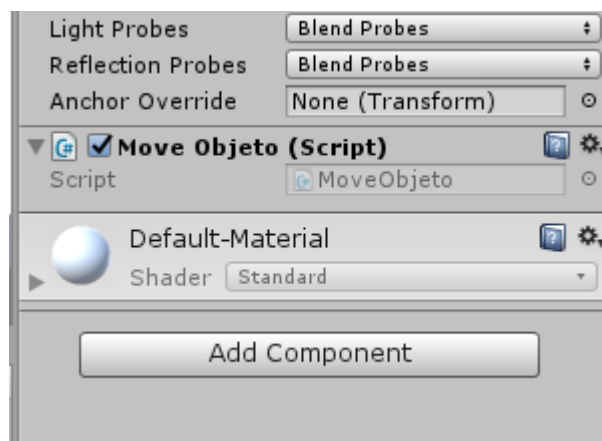
Figura 14 - Adicionando um script criado à um objeto.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Na **Figura 15** você vê o script adicionado com sucesso na esfera.

Figura 15 - Script MoveObjeto adicionado com sucesso na esfera.

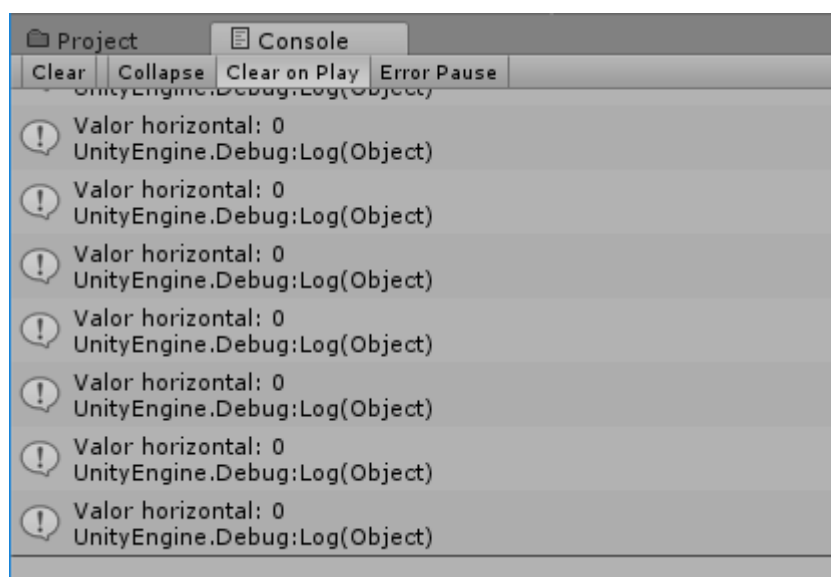


Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Tente novamente iniciar o jogo. Play!

Se tudo der certo, você verá na aba Console uma série de textos nos quais estará escrito “Valor horizontal: 0”, como na **Figura 16**. Isso significa que o método Update() está sendo chamado corretamente na instância do script o qual está associado à esfera a cada frame, mas o valor lido do eixo horizontal é zero. Claro, não pressionamos nenhuma tecla!

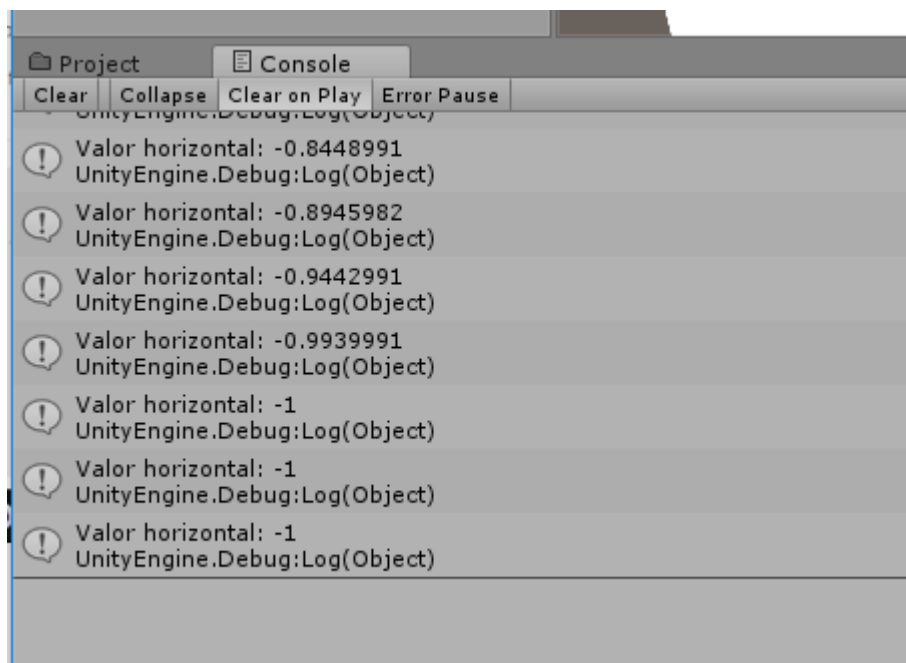
Figura 16 - Console exibindo o valor lido no eixo horizontal.



Fonte: Captura de tela do Unity <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Experimente agora, com o jogo ainda rodando e com a janela do Unity em primeiro plano, mantendo pressionada a tecla “A” do teclado. Observe na **Figura 17** o que acontece com o Console.

Figura 17 - Console exibindo os valores lidos no eixo horizontal.

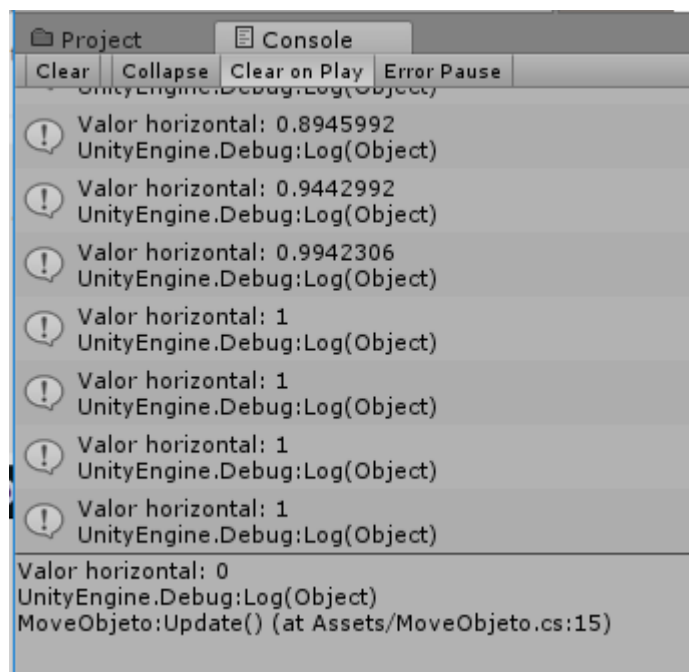


Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Que estranho! Os valores lidos serem negativos faz todo sentido, já que a tecla "A" representa uma intenção de movimento para a esquerda. Porém, você faz alguma ideia sobre esses números decimais aparecerem antes do valor -1 e se manterem estáveis? É simples. Lembra do Input Manager e das configurações dos eixos virtuais? Pois bem, por padrão, o Unity configura os eixos com uma certa sensibilidade (Propriedade Sensitivity igual a 3). Isso significa que quando você pressiona a tecla "A" o sistema de entrada "leva" o eixo horizontal do valor neutro (zero) até -1 de maneira gradual, fazendo o movimento nos jogos ser mais suavizado. Se você desejar, pode aumentar ou diminuir o valor da propriedade Sensitivity do eixo Horizontal para ver como ele se comporta. Vamos deixar com esse valor padrão por enquanto.

Da mesma forma que pressionar A leva o valor Horizontal para -1, pressionando D esse valor vai gradualmente para 1, como visto na **Figura 18**.

Figura 18 - Valor Horizontal para a tecla D pressionada.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Tudo certo agora! Vamos utilizar esse valor para mover a esfera para direita/esquerda de acordo com o desejo do jogador.

Para o jogo e dê um duplo clique novamente para abrir o script no editor. No método Debug(), logo abaixo do comando Debug.Log adicione o seguinte:

```
1 Vector3 mover = new Vector3 (movimentoHorizontal, 0f, 0f);  
2 transform.Translate (mover);
```

Esses dois comandos são bem simples. Inicialmente criamos uma nova variável do tipo Vector3 para representar o quanto desejamos mover a esfera a cada frame. Usamos um Vector3 pois, como estamos em três dimensões, os movimentos podem acontecer em todas as três direções ao mesmo tempo e cada componente do Vector3 (x, y e z) deve guardar o quanto desejamos mover o objeto em cada uma das direções. Repare que inicializamos o Vector3 chamado “mover” com os valores x = movimentoHorizontal, y = 0f e z = 0f. Dessa forma, garantimos que o valor movimentoHorizontal afetará somente o eixo X.

O comando transform.Translate(mover); utiliza, para alterar a posição atual da esfera, o Vector3 criado anteriormente. Todo script associado a um objeto visível no Unity tem uma variável chamada “transform”, a qual provê diversos tipos de

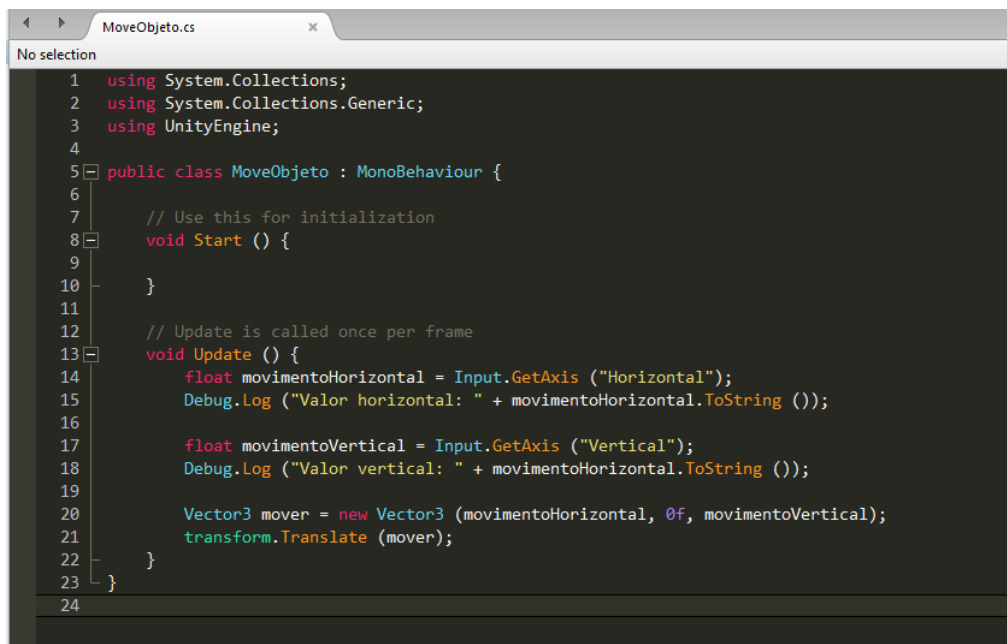
operações que modificam sua posição, rotação e escala. Estamos utilizando o método `Translate` que recebe um `Vector3` e move o objeto nas direções x, y e z na quantidade indicada nos valores das componentes do `Vector3`.

Agora, faremos o mesmo para o eixo Vertical e modificaremos a criação do `Vector3` "mover" a fim de utilizar o valor desse eixo para a componente Z. No Unity, o componente X representa as laterais, e Z representa as direções para frente e para trás. O componente Y é para cima e para baixo, e não queremos mover o objeto nessas direções por enquanto. O seu código final do método `Update()` deve ficar assim:

```
1 void Update () {
2     float movimentoHorizontal = Input.GetAxis ("Horizontal");
3     Debug.Log ("Valor horizontal: " + movimentoHorizontal.ToString ());
4
5     float movimentoVertical = Input.GetAxis ("Vertical");
6     Debug.Log ("Valor vertical: " + movimentoHorizontal.ToString ());
7
8     Vector3 mover = new Vector3 (movimentoHorizontal, 0f, movimentoVertical);
9     transform.Translate (mover);
10 }
```

Na **Figura 19** você pode ver como ficou o código completo do script no Editor.

Figura 19 - Código do script de movimento no editor.



Fonte: Captura de tela do Unity. <https://unity3d.com/pt/>. Acesso em: 24 de fev de 2017.

Finalizamos a primeira parte de nossos estudos sobre as entradas do usuário e movimento, tudo bem até aqui? Você deve ter notado que vários aspectos ainda não estão ideais, como a velocidade do movimento, a falta de reação física da esfera, a câmera travada a qual eventualmente faz o objeto controlado sair do campo de visão, etc. Não se preocupe, pois retomaremos esses tópicos na próxima aula e realizaremos vários ajustes. Aguenta firme, ainda temos muito a aprender! Espero que tenham gostado! ;)

Resumo

Na aula de hoje, estudamos uma pequena amostra do sistema de entrada de comandos do usuário através de eixos virtuais e vimos uma introdução acerca de como movimentar objetos no Unity de acordo com as entradas do jogador. Ainda aprenderemos vários outros mecanismos de entrada e movimento, para você criar jogos bem mais interessantes.

Leitura complementar

O sistema de entradas do Unity é muito poderoso. Mesmo que nas próximas aulas vejamos mais mecanismos de entrada e movimento, existe uma quantidade muito grande de possibilidades e é importante estudá-las através de outras fontes. Abaixo segue uma lista de alguns recursos que você poderá usar para aprender mais sobre o Unity e o desenvolvimento de jogos em geral.

- Aprenda com o Unity (oficial). Disponível em:
<<https://unity3d.com/pt/learn>>
- Manual do Unity (oficial). Disponível em:
<<https://docs.unity3d.com/Manual/>>

Autoavaliação

1. Faça alterações nas propriedades dos eixos virtuais do Unity para criar sistemas de entradas personalizados de acordo com o que você considera adequado.
2. Adicione um componente chamado “Physics > Rigidbody” na esfera do nosso exemplo e observe o que acontece com ela.
3. Antes de aplicar o `transform.Translate(mover)`, tente modificar esse `Vector3`, multiplicando-o por um float entre 0, por exemplo: `mover = mover * 0.2f`. Observe o que ocorre com o movimento da esfera.

Referências

UNITY TECHNOLOGIES. 2016 (C). **Unity 3D Online Tutorials [online]**. Disponível em: <<https://unity3d.com/pt/learn/tutorials>> [Acessado em 16 de novembro de 2016].