

Desenvolvimento com Motores de Jogos II

Aula 11 - Rigidbody

Apresentação

Sejam bem-vindos à aula 11! Antes de avançarmos, vamos fazer um feedback da aula anterior? Na aula passada, vimos os parâmetros avançados do Material e nos aprofundamos no conteúdo correspondente a algumas propriedades relativas à cor e textura. Agora, falaremos sobre um assunto também relevante: a Física envolvida nos jogos. Em vários jogos em três dimensões, um dos fatores mais importantes para obtermos um maior realismo é a forma como os objetos da cena interagem fisicamente com outros. No Unity, existem diversas maneiras de adicionarmos comportamento físico realista aos objetos.

Preparem-se, pois, nesta aula, aprenderemos os conceitos básicos do uso da física 3D no Unity, explorando inicialmente um importante componente chamado **Rigidbody**.

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Aprender a trabalhar com física no Unity em 3D.
- Adicionar comportamentos físicos realistas aos objetos.
- Compreender o conceito do Rigidbody e suas propriedades.

Física no Unity

No Unity, existem dois motores de física (em inglês: Physics Engine), um para física 2D e outro para física 3D. Não é preciso ressaltar que cada um tem suas aplicações e normalmente em jogos 2D usamos o Physics Engine 2D e, em jogos 3D, o Physics Engine 3D. É importante observar que nada impede tecnicamente você utilizar os componentes de física 3D em jogos 2D ou vice-versa. Entretanto é preciso saber a diferença entre eles antes de utilizá-los de forma menos comum. Os conceitos básicos entre os dois são os mesmos, exceto que no 3D temos uma dimensão extra e ambos utilizam componentes diferentes.

Os nomes dos componentes de física em 2D têm escrito "2D" no final, enquanto os componentes 3D não têm essa terminação. Exemplo: Rigidbody é um componente para física 3D e o seu semelhante no mundo 2D se chama Rigidbody2D.

Com o Physics Engine 3D e seus componentes, você poderá deixar seus jogos mais dinâmicos e realistas. Para isso, na próxima seção abordaremos o conceito do componente **Rigidbody** e o modo como configurar suas propriedades, além de explorar suas relações com os colisores (Colliders).

Adicionando um Rigidbody

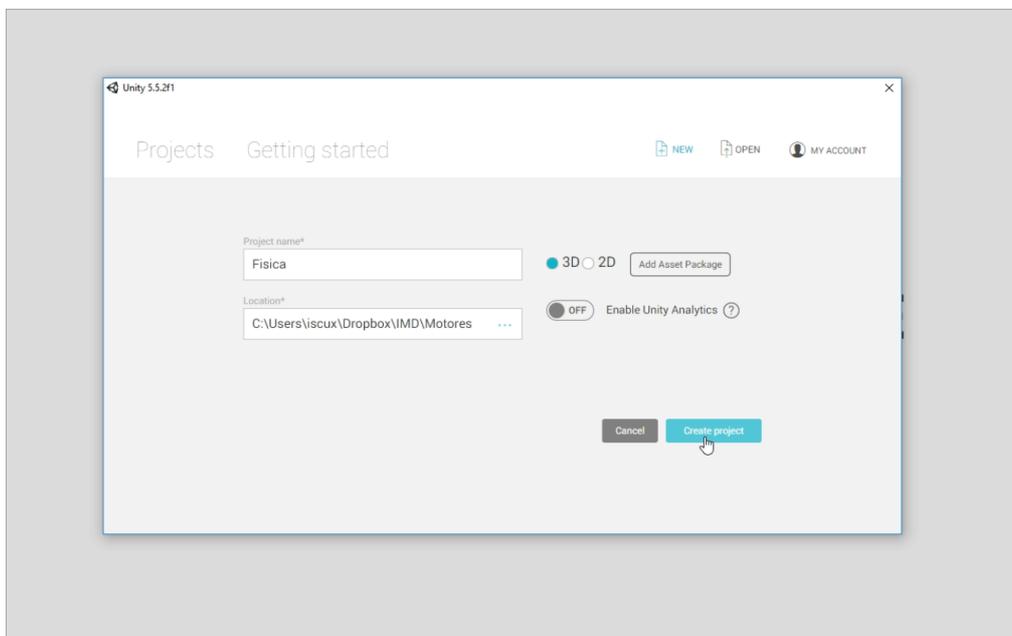
No Unity, o principal componente capaz de adicionar comportamento físico a um objeto é o **Rigidbody**. Você pode adicionar a qualquer GameObject um componente do tipo **Rigidbody** e esse GameObject ganhará imediatamente características as quais permitem que ele responda à gravidade da maneira esperada.

Além de reagir à gravidade, o **Rigidbody** também permite ao objeto ter interações com outros objetos que colidem nele, entretanto para isso é necessário ambos os objetos terem um colisor (Collider) como componente. Os objetos 3D

padrão do Unity, como Cube, Cylinder, etc., já são criados pela configuração padrão com um colisor.

Para estudar a física no Unity, criaremos um novo projeto chamado “Física”, conforme mostra a **Figura 1**.

Figura 01 - Novo projeto “Física”.

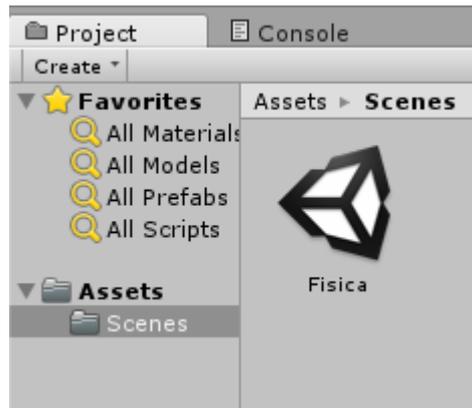


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Evite utilizar acentos nos nomes dos projetos e arquivos criados nos seus projetos do Unity. É comum acontecer problemas de codificação quando se move o projetos para outros discos, sistemas de arquivos, sistemas operacionais, transferência pela Internet, etc. Apesar de funcionar, muitos consideram uma boa prática utilizar somente caracteres sem acentos para esses elementos.

Com o projeto criado, adicione uma pasta chamada “Scenes” para guardar as cenas e salve a cena atual com o nome “Física”. Veja a **Figura 2**.

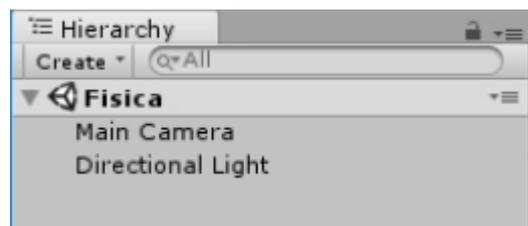
Figura 02 - Cena atual salva no projeto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Como toda nova cena do Unity, a nossa já vem com uma câmera (Main Camera) e uma Directional Light criada. Veja a **Figura 3**.

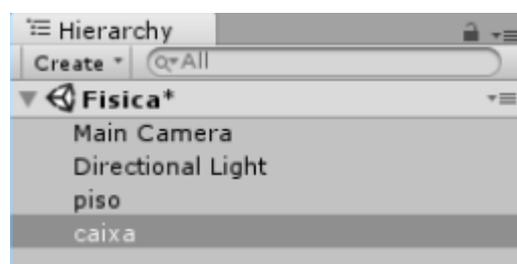
Figura 03 - Objetos padrão da cena.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

A fim de entender o comportamento do **Rigidbody**, criaremos uma simples cena com um piso e uma caixa flutuando acima dele. Para isso, crie dois Cubes na cena e os renomeie para “piso” e “caixa”, conforme exhibe a **Figura 4**.

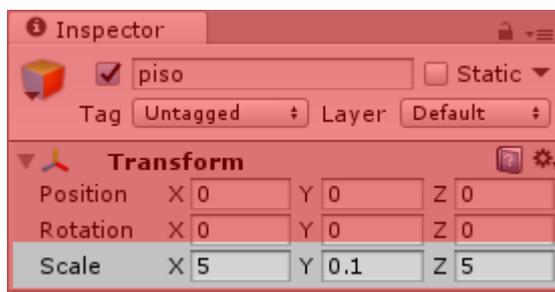
Figura 04 - Piso e caixa criados na cena e exibidos no Hierarchy.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Atualmente os dois cubos devem estar na posição 0,0,0 e com a mesma escala, portanto você deve estar vendo somente um deles. Clique no piso e modifique a sua escala para os valores X=5, Y=0.1, Z=5, através do Inspector. Veja o exemplo na **Figura 5**.

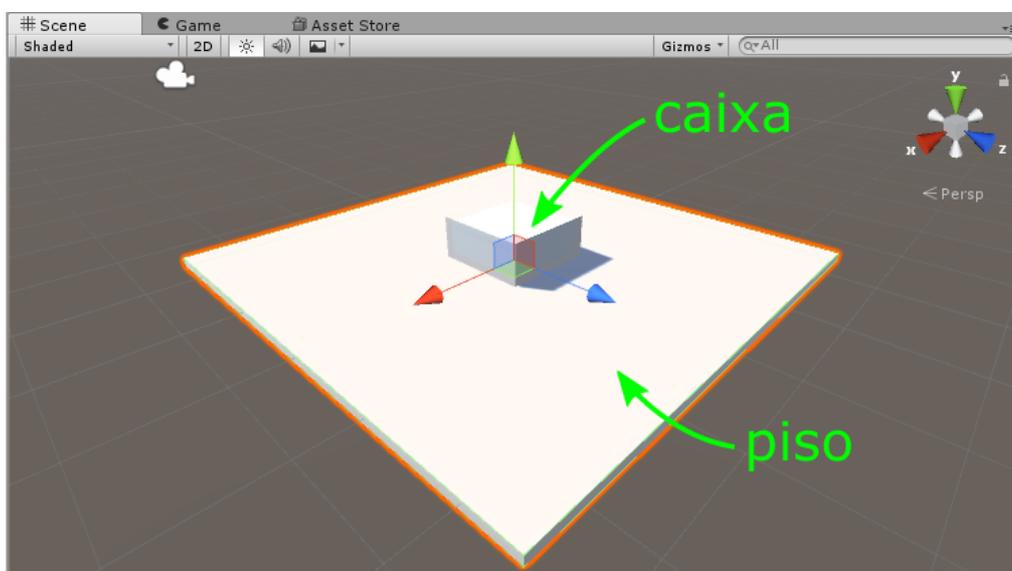
Figura 05 - Escala do piso modificada para 5, 0.1, 5.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Essa mudança o fará ficar maior na lateral e na profundidade, entretanto mais achatado na altura. Veja na **Figura 6** o resultado parcial da cena e repare que o piso está achatado e a caixa está ainda na posição original (dentro dele).

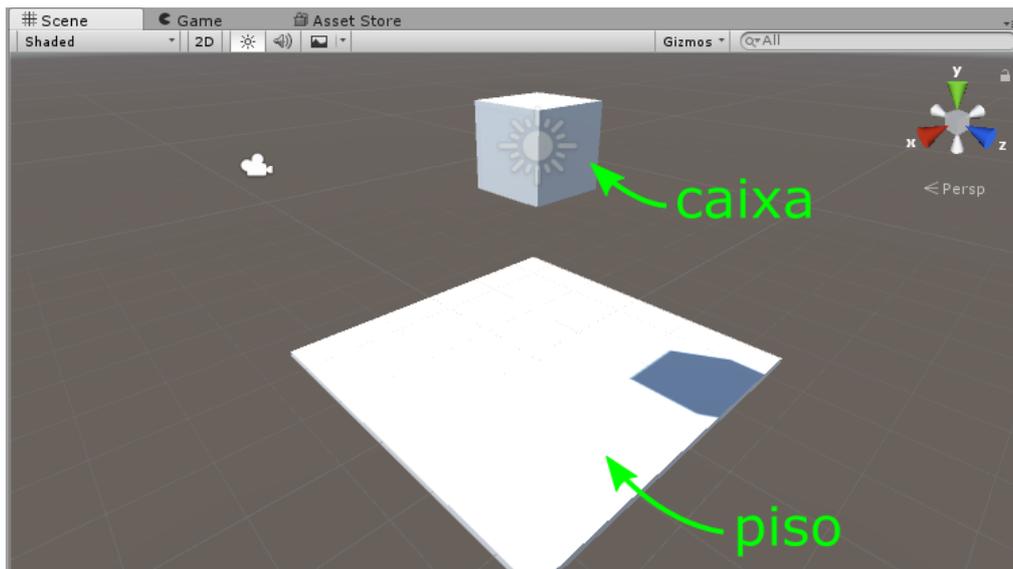
Figura 06 - Caixa ainda dentro do piso (já escalonado).



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Agora, clique na caixa e modifique sua posição no eixo Y para o valor 3. Isso fará com que ela fique acima do piso, flutuando no ar, conforme mostra a **Figura 7**.

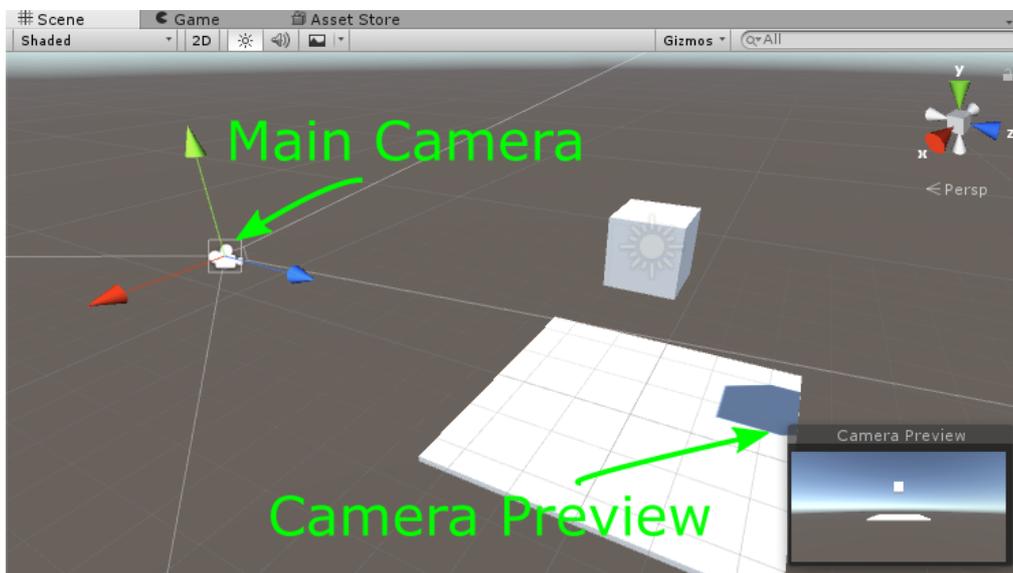
Figura 07 - Caixa acima do piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Posicione sua Main Camera de forma que o Preview mostre os dois objetos, como na **Figura 8**.

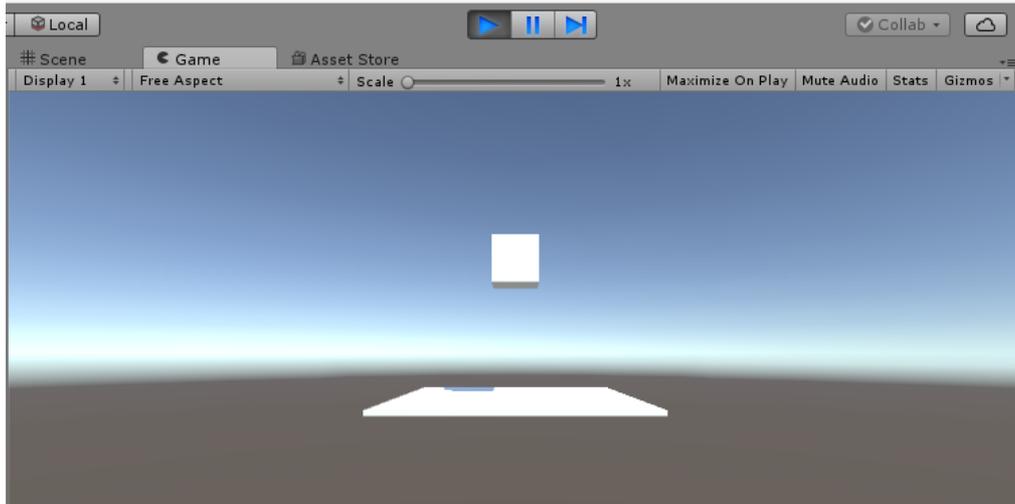
Figura 08 - Posicionamento da Main Camera.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Ao clicar no botão Play para iniciar o jogo, você verá que, mesmo com o jogo em execução, nenhum dos dois objetos se movem, isto é, eles se mantêm onde estão, conforme exibe a **Figura 9**.

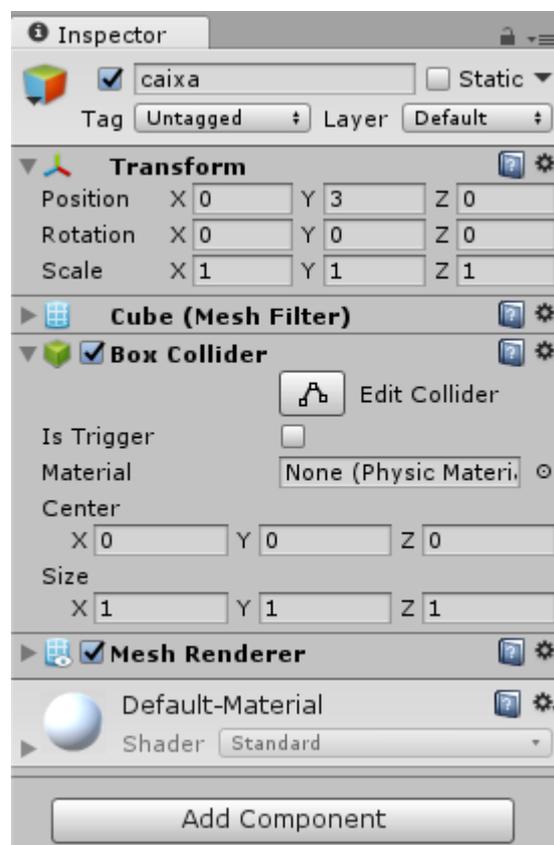
Figura 09 - Execução da cena com objetos sem Rigidbody.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Vamos, agora, adicionar um **Rigidbody** na caixa para que ela tenha uma reação física aos efeitos da gravidade. Clique na caixa e observe suas propriedades atuais no Inspector, como na **Figura 10**.

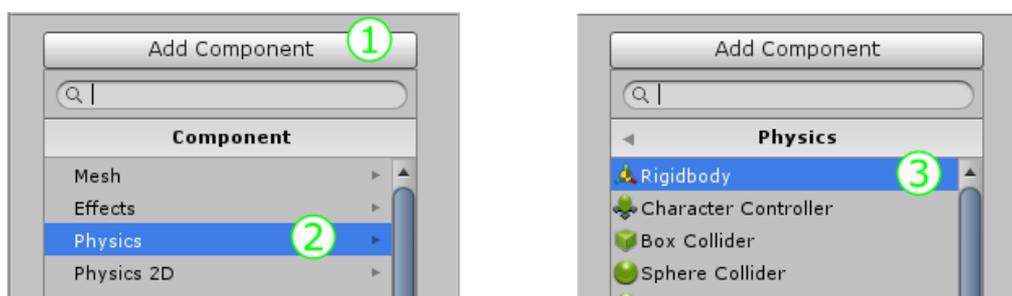
Figura 10 - Propriedades atuais da caixa.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Repare que um dos componentes já adicionados pela configuração padrão na caixa é um **Box Collider**. O **Box Collider** é um colisor especial que tem o formato de um cubo e por padrão tem as mesmas dimensões do Cube adicionado por você. Sem um collider, o **Rigidbody** ainda funciona com relação à gravidade, entretanto não interage ao colidir com outros objetos, por exemplo com o piso. Adicionaremos um componente **Rigidbody** clicando no botão “Add Component” e, depois, escolhendo a opção Physics -> Rigidbody, seguindo a sequência mostrada na **Figura 11**.

Figura 11 - Adicionando um Rigidbody na caixa.

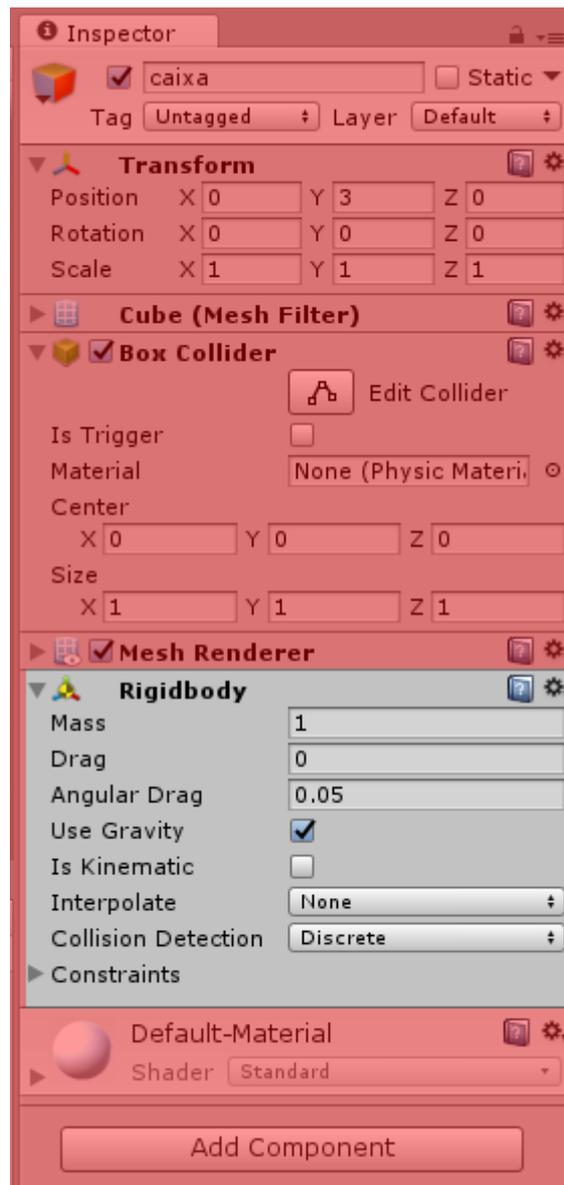


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Repare que, ao adicionar um **Rigidbody**, você escolhe inicialmente a seção “Physics”. Porém, também existe uma outra, chamada “Physics 2D”, a qual contém os componentes de física para cenas 2D. No nosso caso, usaremos o 3D que está na seção “Physics” e tem o nome “Rigidbody”.

Depois de adicionar o **Rigidbody**, o Inspector do GameObject na caixa deve mostrar algo como o que é exibido na **Figura 12**, agora tanto com o **Box Collider** quanto com o **Rigidbody** adicionados como componentes.

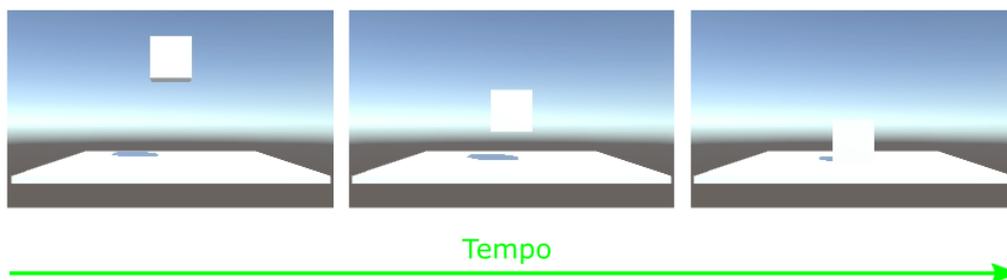
Figura 12 - Visão do Inspector da caixa com Rigidbody.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Antes de investigar as propriedades do **Rigidbody**, observaremos o que a sua adição na caixa modifica no comportamento da cena durante a execução do jogo. Clique no botão Play  para iniciar o jogo e repare que a caixa agora não fica parada, mas cai até atingir o piso, como representado em três momentos na **Figura 13**.

Figura 13 - Queda da caixa com Rigidbody até atingir o piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

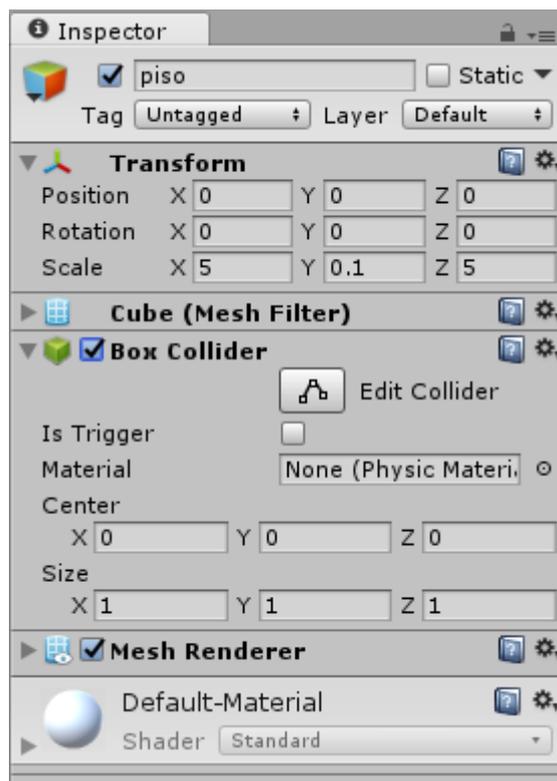
Legal, não é mesmo? Com o uso de Rigidbody, você pode facilmente adicionar um comportamento de reação à gravidade em objetos do seu jogo.

Algo interessante de se observar é o fato de o piso não reagir à gravidade, justamente por não ter em si o componente **Rigidbody**. Além disso, a caixa só cai até o piso devido ao fato do **Rigidbody** dela “notar” que o piso (assim como a caixa) tem um componente do tipo **Box Collider** e, portanto, esse componente detecta a colisão com o piso, parando a queda.

Objetos com e sem Colisores

Faremos, agora, uma pequena modificação a fim de desabilitar o **Box Collider** do piso. Para isso, clique no piso e observe no Inspector seus componentes. Veja a **Figura 14**.

Figura 14 - Componentes atuais do piso.

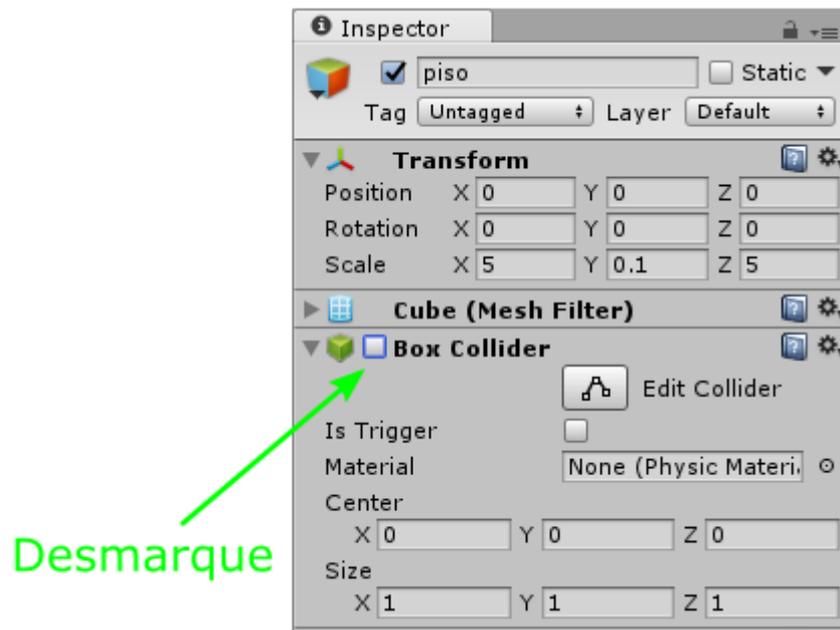


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Repare: o piso, apesar de ter um **Box Collider**, não tem um **Rigidbody**, porque não o adicionamos. Na verdade, isso está certo, pois não queremos ver o nosso piso caindo com a gravidade, porém desejamos que ele seja visto pelos outros objetos na cena como um objeto no qual pode haver colisões. Por esse fator, a caixa só cai até colidir com ele.

Agora, faremos um teste e desabilitaremos o **Box Collider** do piso. Para isso, com o piso selecionado, desmarque a opção que ativa o componente **Box Collider**, a qual fica ao lado esquerdo do seu nome, assim como mostra a **Figura 15**.

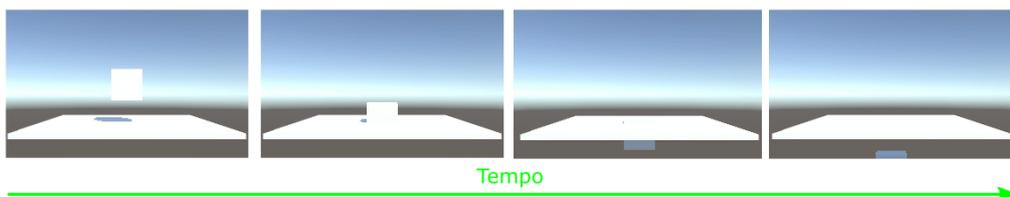
Figura 15 - Desabilitando o Box Collider do piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Execute novamente o jogo e verá que a caixa cairá, como da vez anterior. Entretanto, agora ela não parará no piso, e sim passará por dentro dele, ignorando completamente sua existência, e continuará caindo indefinidamente, como representado nos quadros da **Figura 16**.

Figura 16 - Caixa caindo e passando pelo piso que está com o Box Collider desativado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Agora, nosso piso não oferece resistência alguma à queda da caixa. Você pode se perguntar em que casos isso seria útil, e a resposta é: depende do tipo de cena que você está fazendo no seu jogo. Evidentemente, objetos como um piso precisam do colisor ativo, entretanto você talvez esteja querendo criar objetos a fim de representarem água, por exemplo, os quais devem permitir a outros objetos passarem através deles.

Existem outras maneiras de permitir que objetos passem por dentro de outros sem desabilitar completamente o seu colisor. Não se preocupem agora! Nas próximas aulas, falaremos mais sobre isso e sobre outras funcionalidades dos colisores.

Veja agora um vídeo que demonstra os efeitos de um Rigidbody adicionado a um objeto e o modo como ele interage com os outros objetos da cena.

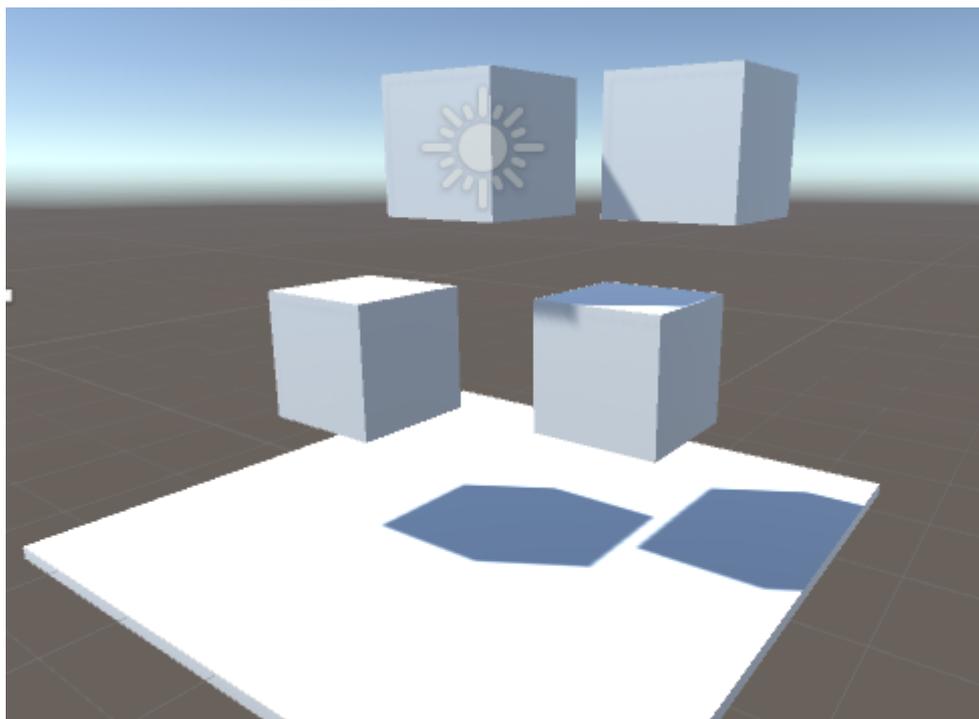


Vídeo 01 - Rigidbody Gravidade

Atividade 01

1. Modifique a sua cena habilitando novamente o colisor do piso e, em seguida, o rotacione 45 graus no eixo X. O que ocorre quando você executa o jogo?
2. Adicione outras caixas na cena, todas com Rigidbody, e as posicione em locais diferentes, porém próximas umas das outras, similar à imagem abaixo. O que acontece quando você executa o jogo?

Figura 17 - Caixas com Rigidbody posicionadas em locais diferentes na cena.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Propriedades do Rigidbody

Já vimos que, quando adicionamos um componente do tipo **Rigidbody** em um GameObject, este ganha um comportamento físico, reagindo a coisas, tais como gravidade e colisão.

As propriedades do **Rigidbody** são:

Mass

A massa do objeto em quilogramas.

Drag

O quanto que a resistência com o ar afeta o objeto, podendo ser de zero (nenhuma resistência) até Infinity (resistência total, o objeto para de se mover imediatamente).

Angular Drag

O quanto que a resistência do ar afeta a rotação do objeto, ou seja, se for zero não tem nenhum efeito e se for maior que zero o objeto aos poucos desacelera a sua rotação.

Use Gravity

Determina se o objeto reage à gravidade (marcado) ou não (desmarcado).

Is Kinematic

Se estiver marcado, significa que o objeto não será movido pelo sistema de física do Unity, podendo ter sua posição e rotação modificada somente se um script manipular diretamente o seu Transform. Isso é útil se você deseja animar um Rigidbody que tem HingeJoint anexada. Nesta aula, não exploraremos esse modo.

Interpolate

Suaviza o movimento do Rigidbody. Use somente se você estiver observando movimentos abruptos na sua animação. Pode ser None (nenhuma interpolação), Interpolate (suaviza baseado no frame anterior), Extrapolate (suaviza baseado em uma estimativa do próximo frame).

Collision Detection

Usado para prevenir que objetos se movendo muito rápido não tenham colisões detectadas por passar através de outros entre dois frames. Pode ter os valores:

- * Discrete: detecção normal entre todos os objetos (padrão).
 - * Continuous: detecção de colisão com maior precisão, mas muito mais lenta. Só use se você tiver problemas com o modo Discrete.
 - * Continuous Dynamic: usado somente para objetos de movimento muito rápido.
-

Constraints

Restringe o movimento do Rigidbody: Freeze Position determina que eixos não podem se mover, e Freeze Rotation determina que eixos não podem girar. É útil, por exemplo, se você estiver fazendo um jogo de plataforma em 3D e não deseja que os objetos se movimentem no eixo Z (profundidade).

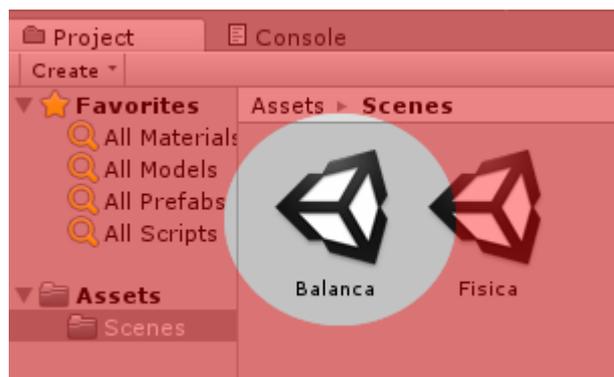
A seguir, exploraremos detalhadamente três dessas propriedades: **Mass, Drag e Angular Drag**.

Mass

Mass, em português, significa “Massa” e é justamente o que seu nome indica, ou seja, a massa do objeto em quilogramas. Essa propriedade pode ser ajustada modificando a forma como os corpos reagem a colisões. Assim, os corpos com mais massa acumulam mais energia do que os com menos massa, quando estão em uma mesma velocidade.

Para testar a propriedade **Mass**, criaremos uma balança no Unity. Ainda no projeto Fisica, crie uma cena através do menu File -> New Scene e salve-a na pasta Scenes, com o nome “Balanca”. Veja a **Figura 18**.

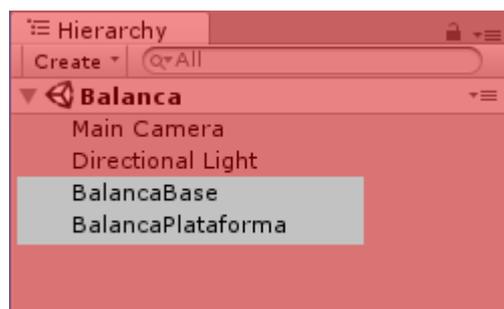
Figura 18 - Cena “Balanca” na pasta Scenes.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Inicialmente, adicionaremos na cena dois cubos, de modo a criar nossa balança. O primeiro chamaremos de BalancaBase e o segundo, de BalancaPlataforma, como visto no Hierarchy mostrado na **Figura 19**.

Figura 19 - Duas partes da balança renomeadas no Hierarchy.



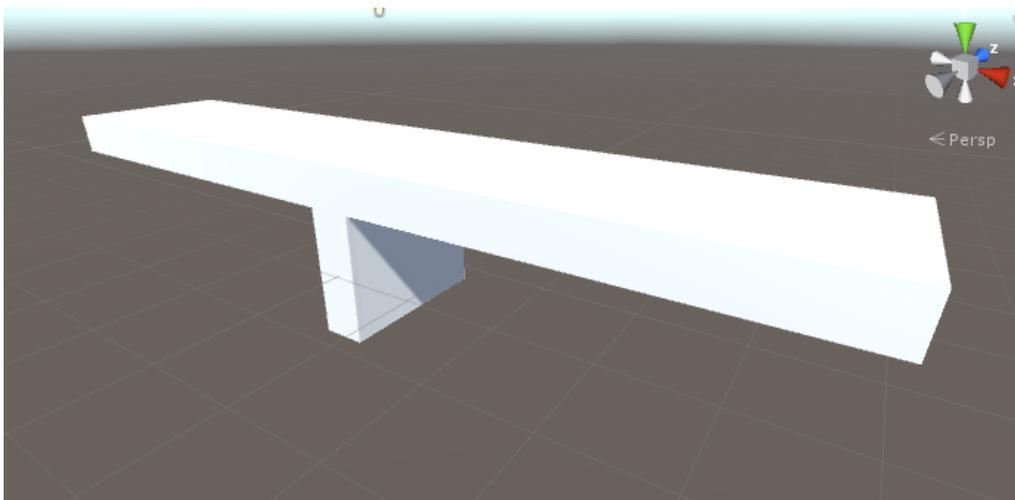
Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Os dois devem ter as seguintes propriedades no Transform:

- BalancaBase:
 - Position: 0, 0, 0
 - Rotation: 0, 0, 0
 - Scale: 0.5, 1.5, 2
- BalancaPlataforma:
 - Position: 0, 1, 0
 - Rotation: 0, 0, 0
 - Scale: 10, 0.5, 2

Essas configurações do Transform dos dois cubos devem gerar uma cena como a vista na **Figura 20**, na qual os dois objetos formam um “T” que será utilizado como uma balança simplificada.

Figura 20 - Balança formada por dois objetos.



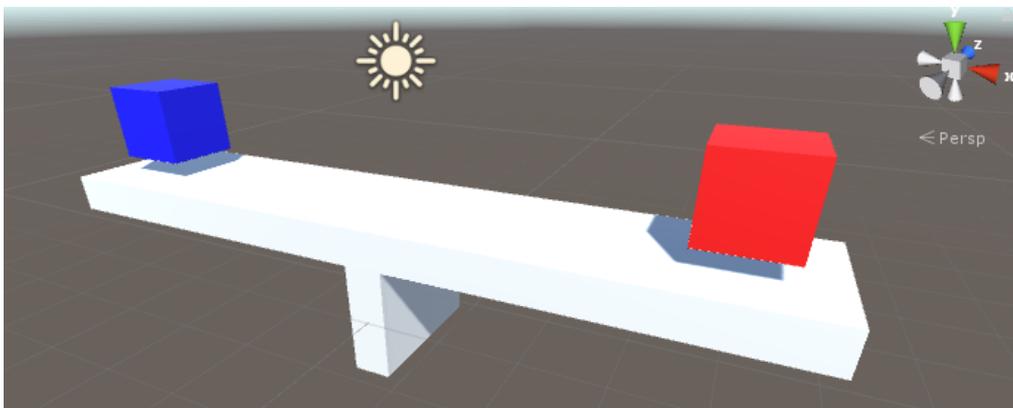
Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Crie agora dois outros cubos chamados de Peso1 e Peso2, respectivamente, e os configure com as seguintes propriedades:

- Peso1:
 - Position: -4, 2, 0
 - Rotation: 0, 0, 0
 - Scale: 1, 1, 1
- Peso2:
 - Position: 4, 2, 0
 - Rotation: 0, 0, 0
 - Scale: 1, 1, 1

Como você já aprendeu a criar Materials, adicione no seu projeto dois Materials, um de cor sólida Azul e outro de cor sólida Vermelho, e aplique ao Peso1 o Azul e ao Peso2 o Vermelho. Sua cena deve estar como a vista na **Figura 21**.

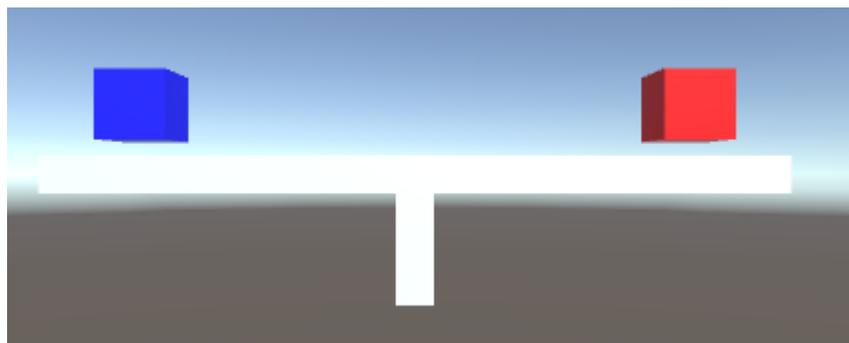
Figura 21 - Cena com os dois pesos adicionados com cores diferentes.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Se você pressionar o botão Play  agora, verá que nada acontece e obterá um resultado estático, sem movimento, conforme a **Figura 22**. Inclusive, verá que os pesos estão posicionados um pouco acima da BalancaPlataforma, flutuando no ar.

Figura 22 - Nenhum movimento na nossa balança.

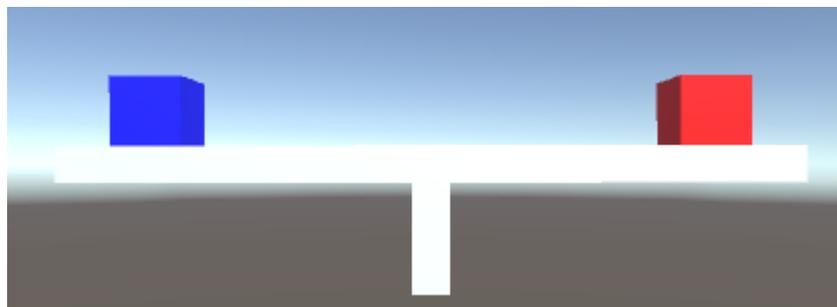


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Como você já deve ter notado, esse comportamento é esperado, pois ainda não adicionamos os Rigidbody nos objetos. Vamos fazer isso agora.

Primeiro, adicione um componente **Rigidbody** (Add Component -> Physics -> Rigidbody) no Peso1 (Azul) e configure sua propriedade **Mass** para 2 (2kg). Depois, adicione um **Rigidbody** no Peso2 (Vermelho) e configure sua propriedade **Mass** para 2 (2kg), ou seja, a mesma massa que o Peso1. Clique em Play e veja um resultado como o da **Figura 23**.

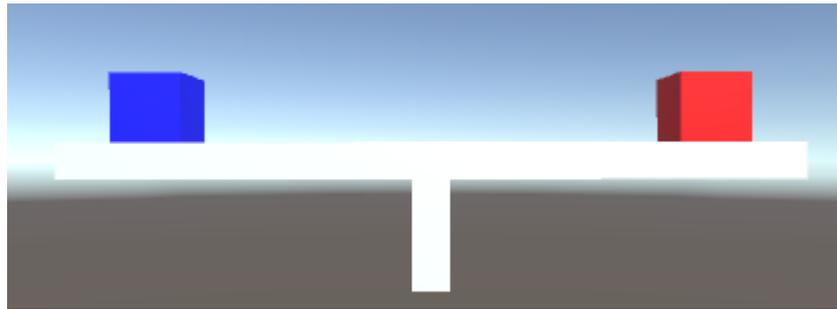
Figura 23 - Peso1 e Peso2 com a mesma massa (2kg).



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Repare que agora os dois pesos já estão tocando a balança, pois eles reagiram à gravidade e caíram até entrar em contato com a BalancaPlataforma, entretanto essa plataforma não se move. Você pode imaginar que o motivo da plataforma não se mexer é devido à massa dos dois pesos serem iguais, correto? Vamos testar. Pare a execução da cena e modifique a massa do Peso2 (Vermelho) de 2 para 20, ou seja, 10 vezes maior. Execute novamente e veja o resultado como o da **Figura 24**.

Figura 24 - Dois pesos com massas diferentes.

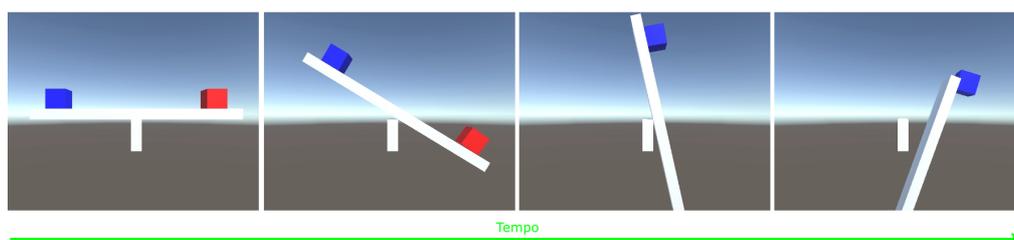


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

A mesma coisa: a balança não se mexe. O que será isso? É de se esperar que a nossa balança minimalista pendesse para o lado do peso com maior massa, correto? Na verdade, o motivo disso é o objeto BalancaPlataforma não ter um **Rigidbody**, apesar de ter um colisor fazendo os pesos pararem nele, ou seja, BalancaPlataforma não reage às colisões modificando a sua posição, rotação ou escala. Ela é efetivamente um bloco estático. Para corrigir isso, clique na BalancaPlataforma e adicione um **Rigidbody**, mantendo as configurações originais. Não precisa adicionar **Rigidbody** no objeto BalancaBase, pois realmente não queremos que ele se mova.

Execute novamente a cena e veja que, agora sim, nossa balança está se comportando da forma esperada. Na **Figura 25**, você pode ver esse comportamento em alguns quadros.

Figura 25 - Comportamento da balança com os pesos que possuem massas diferentes e também com a plataforma com Rigidbody.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Para criar um balança mais realista, a plataforma não poderia simplesmente se desprender da base. No entanto, esse simples exemplo ilustra bem as reações entre os objetos em cena com o uso do **Rigidbody**, no qual a propriedade **Mass** é configurada com valores diferentes para cada objeto.

Assista ao vídeo no qual é mostrada a criação de uma cena que representa uma balança virtual a ser utilizada para testar diferentes configurações de massa em objetos com Rigidbody.



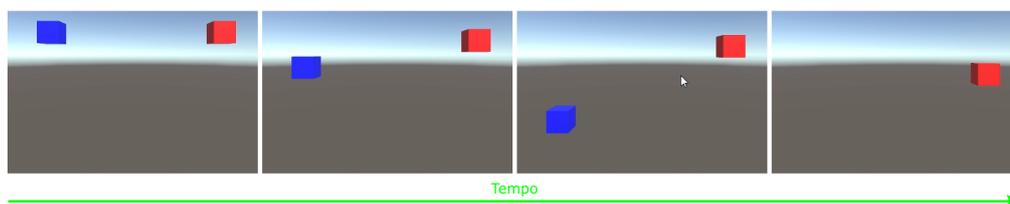
Vídeo 02 - Balança

Drag

A propriedade **Drag** representa o coeficiente linear de arrasto com o ar do objeto. Não se assuste, é bem simples. Basicamente o **Drag** informa quanto de resistência com o ar afeta o objeto, podendo ter valores de zero até infinito. Zero significa que o objeto não tem nenhuma resistência com o ar, e infinito significa que ele tem uma resistência infinita, ou seja, o objeto para de se mover imediatamente. Para configurar a propriedade **Drag** com o valor infinito, basta digitar "Infinity" no seu valor.

Para testar essa propriedade, crie dois cubos idênticos, adicione um **Rigidbody** em cada um e modifique, somente em um deles, o **Drag** de zero para 10. Execute a cena e verá que o cubo com Drag=0 cairá com uma velocidade maior do que o cubo com Drag=10. Veja a **Figura 26**.

Figura 26 - Objetos em queda com o Drag=0 para o Azul e Drag=10 para o Vermelho.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Agora, veja um vídeo que demonstra os efeitos causados em um objeto quando modificamos a propriedade **Drag** do seu Rigidbody.



Vídeo 03 - Drag

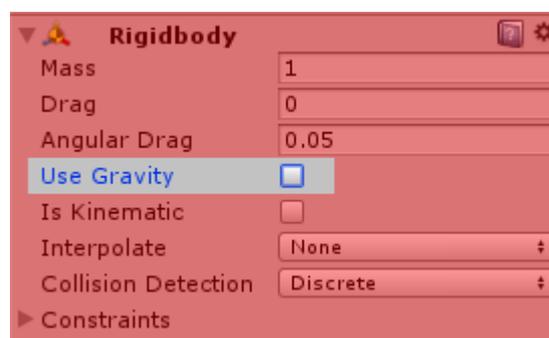
Angular Drag

A propriedade **Angular Drag** indica quanta resistência o objeto com um **Rigidbody** tem à rotação proveniente de um torque, ou seja, se um objeto está girando em seus eixos na cena e tem um **Angular Drag** diferente de zero, ele reduz aos poucos essa velocidade de rotação até chegar em zero.

Um objeto pode iniciar um movimento de rotação dinamicamente em uma cena por vários motivos. Por exemplo: um cuco pode ser atingido por outro objeto em uma de suas quinas e, assim, começar a girar, ou em uma queda ele pode atingir uma plataforma em um determinado ângulo, também iniciando uma rotação, etc.

Para testar essa propriedade, iniciaremos a rotação em um cubo em um de seus eixos através de um script. Para isso, crie uma nova cena chamada "AngularDrag", adicione somente um simples cubo no centro dela (Position=0, 0, 0) e adicione nesse cubo um **Rigidbody**. Desmarque a opção "Use Gravity" desse **Rigidbody**, fazendo-o não reagir à gravidade, pois não queremos que o objeto caia e saia da nossa visão. As configurações do **Rigidbody** desse cubo devem ser como as exibidas na **Figura 27**.

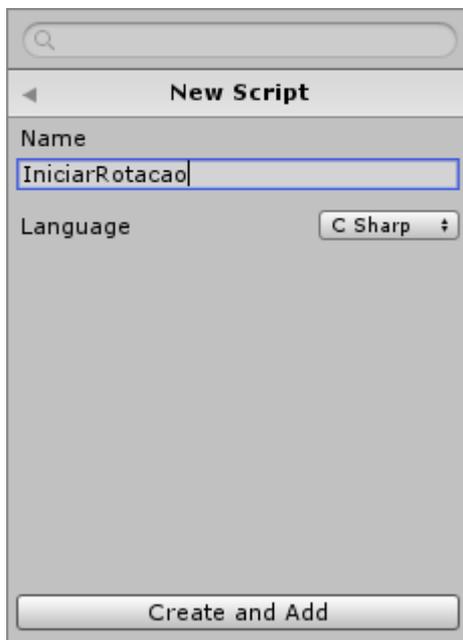
Figura 27 - Rigidbody em objeto com a opção "Use Gravity" desmarcada.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Adicione no cubo um novo script (Add component -> New script), nomeie esse script de IniciarRotacao e clique em "Create and Add", como na **Figura 28**.

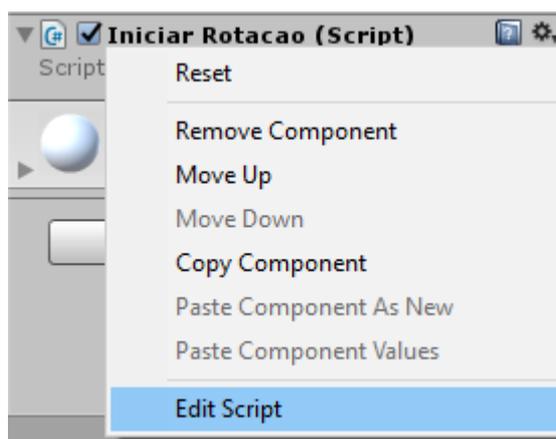
Figura 28 - Adicionando script de iniciar rotação no cubo.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Clique no novo script IniciarRotacao com o botão direito do mouse e escolha a opção Edit Script, como mostra a **Figura 29**, para abrir o MonoDevelop com o script aberto.

Figura 29 - Opção de editar o script IniciarRotacao.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

O MonoDevelop iniciará com um script no qual há somente os métodos Start() e Update(), ambos vazios. Como queremos apenas iniciar um movimento de rotação, utilizaremos o método Start() para aplicar um torque no cubo que tem esse script. Para isso, precisaremos primeiro obter o **Rigidbody** do cubo e guardar sua referência em uma variável na qual executaremos o método AddTorque.

Para obter o **Rigidbody** do objeto e armazenar sua referência em uma variável, execute os comandos abaixo:

```
1 Rigidbody meu_rigidbody;  
2 meu_rigidbody = GetComponent<Rigidbody> ();
```

O comando GetComponent objetiva procurar no objeto que tem o script associado um componente de determinado tipo. Como estamos querendo buscar um componente do tipo **Rigidbody**, colocamos entre < e > esse tipo e executamos o comando com “()” no final, de modo que ele fique assim: GetComponent<Rigidbody> (); a variável meu_rigidbody guarda, então, uma referência ao **Rigidbody** associado ao cubo.

O objeto **Rigidbody** do Unity é extremamente poderoso e tem vários métodos que podem ser executados para se adicionar forças físicas a objetos na cena através de scripts. Como estamos querendo iniciar uma rotação adicionando um torque, usaremos o método AddTorque().

O método AddTorque pode ser executado de várias formas nas quais você pode passar como parâmetro, por exemplo, um Vector3 com os valores de X, Y e Z correspondentes a quanto de torque você deseja adicionar em cada eixo.

No nosso caso, escolheremos que o cubo gire no eixo Z, mas poderia ser em qualquer um dos eixos, ou até em vários ao mesmo tempo. Executaremos o método AddForce utilizando sua versão que recebe como parâmetro os valores dos 3 eixos separadamente, como 3 floats. Então, o comando ficará assim:

```
1 meu_rigidbody.AddTorque (0f, 0f, 10f);
```

Repare que os dois primeiros parâmetros do AddForce, chamados na referência do **Rigidbody** do cubo, são 0f e 0f. Eles representam quanto de torque queremos adicionar nos eixos X e Y, isto é, nenhum, portanto correspondem a 0f (adicionamos

o "f" no final do zero para indicar que se trata de um float, e não de um inteiro no C#). O terceiro parâmetro é 10f e é referente ao eixo Z.

Dessa forma, o método adicionará um torque de valor 10 no eixo Z somente no início da execução da cena, já que ele está no método Start() e, a partir daí, o **Rigidbody** se responsabilizará por manter o objeto girando. O script IniciarRotacao deve ficar como é mostrado na **Figura 30**.

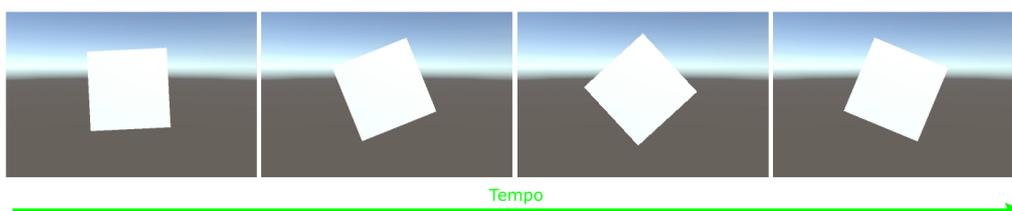
Figura 30 - Script IniciarRotacao.

```
IniciarRotacao.cs
IniciarRotacao ▶ Start ()
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class IniciarRotacao : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9         Rigidbody meu_rigidbody;
10        meu_rigidbody = GetComponent<Rigidbody> ();
11
12        meu_rigidbody.AddTorque (0f, 0f, 10f);
13    }
14
15 }
16
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Volte agora ao Unity, pressione Play  e veja que o cubo começará a girar no eixo Z, como exibe a **Figura 31**.

Figura 31 - Cubo girando no eixo Z devido ao torque aplicado pelo script IniciarRotacao.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 03 de abril de 2017.

Como o **Angular Drag** por padrão vem configurado para 0.05, o cubo iniciará a rotação no eixo que escolhermos, mas aos poucos diminuirá a velocidade de rotação até parar por completo. Se você aumentar o **Angular Drag**, digamos para 2, ele parará mais rápido e, se você colocá-lo em zero, o cubo continuará girando indefinidamente.

Veja um vídeo no qual é demonstrado o que acontece quando mudamos a propriedade **Angular Drag** do Rigidbody de um objeto na cena.



Vídeo 04 - Angular Drag

Resumo

Nesta aula, aprendemos a adicionar e a configurar reações físicas em objetos no Unity, utilizando o componente Rigidbody. Vimos como adicionar o componente, configurar a propriedade **Mass**, **Drag** e **Angular Drag**, fazendo suas colisões e reações à gravidade serem personalizadas. Para testar as mudanças na propriedade **Angular Drag**, criamos também um script que adiciona um torque a um objeto na cena.

Leitura Complementar

O estudo do componente **Rigidbody** se complementa com o estudo de Colliders, portanto abaixo segue uma lista de links para você conhecer mais sobre os dois assuntos:

- <https://docs.unity3d.com/Manual/class-Rigidbody.html>
- <https://docs.unity3d.com/Manual/PhysicsOverview.html>
- <https://docs.unity3d.com/Manual/class-BoxCollider.html>

Autoavaliação

1. Modifique a cena da balança e coloque a massa do Peso1=2 e Peso2=20. Agora, coloque a BalancaPlataforma com a massa 100. Execute a cena e veja o que acontece quando você aumenta a massa da BalancaPlataforma.
2. Na cena do AngularDrag, modifique o script IniciarRotacao e adicione valores positivos também para o AddTorque nos eixos X e Y. Observe o que acontece ao executar a cena.

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].

UNITY TECHNOLOGIES. 2016 (C). Unity Manual - Prefabs [online]. Disponível em: <https://docs.unity3d.com/Manual/Prefabs.html> [Acessado em 16 de novembro de 2016].

STOY, C. 2006. Game object component system. In Game Programming Gems 6, Charles River Media, M. Dickheiser, Ed., Páginas 393 a 403.

MARQUES, Paulo; PEDROSO, Hernâni - C# 2.0 . Lisboa: FCA, 2005. ISBN 978-972-722 208-8

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Manual [online]. Disponível em: <https://docs.unity3d.com/Manual/index.html> [Acessado em 16 de novembro de 2016].