# Desenvolvimento com Motores de Jogos I

## Aula 04 - Detecção de colisão e o Motor de Física

Material Didático do Instituto Metrópole Digital - IMD
Versão 5.0 - Todos os Direitos reservados

# Apresentação

Olá, pessoal! Encontramo-nos novamente para a nossa quarta aula da disciplina autoproclamada a mais legal de todos os cursos do IMD: a de Motores de Jogos!

Em nossa última aula, avançamos um pouco mais com o nosso projeto e conseguimos colocar o nosso robô para se mover para cima, para baixo, para um lado e para o outro, acelerando “loucamente” e, inclusive, saindo da tela! Tiramos até a gravidade dele para que ele pudesse voar livremente. Mas isso não está certo, não é? O robô precisa obedecer à gravidade, se apoiar em um chão e andar sobre ele de modo adequado. E é aí que entram os colisores! Precisamos garantir que o Unity entenda quem deve colidir em quem, para garantir que haverá uma física adequada em nosso jogo.

A fim de compreender bem toda essa parte referente à física e aos colisores, precisamos aprender um pouco mais sobre o nosso motor de física! Outro assunto que discutiremos nesta aula é o modo como o Unity atua em relação à física e quais os principais aspectos de atenção indispensável para que, em nosso jogo, ela funcione de maneira adequada. Muitos dos conceitos vistos aqui serão também estudados em Física para Jogos, no avançado. Lembre-se disso! o/

Tudo pronto? Ready, set, GO!

## Objetivos

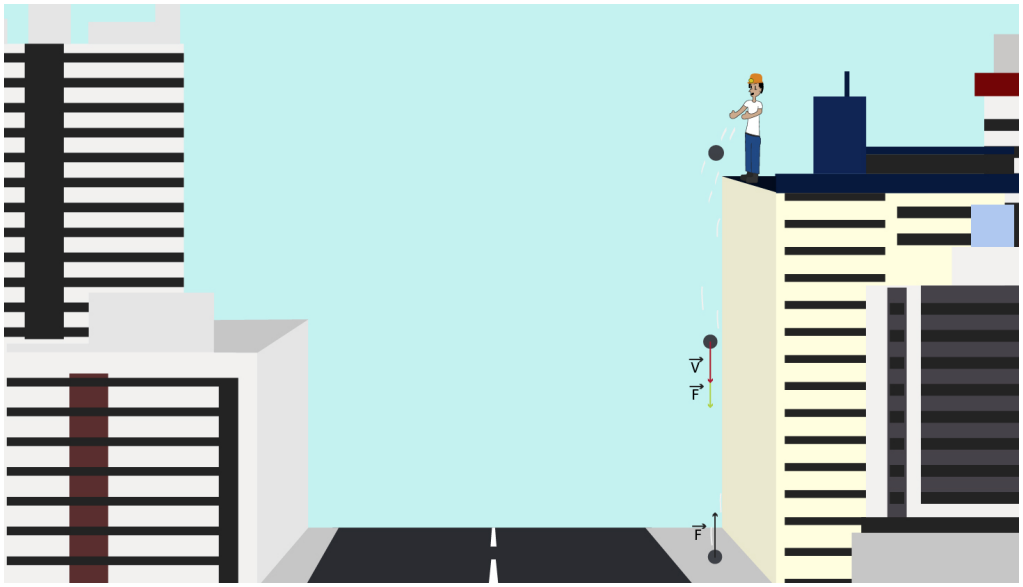
Ao final desta aula, você deverá ser capaz de:

- Entender melhor o funcionamento do motor de física no Unity;
- Compreender como se dá o funcionamento da física nos jogos e qual a sua importância para eles;
- Implementar Colliders, Effectors e Joints em 2D, no Unity;
- Utilizar materiais físicos para obter efeitos diferenciados em seus objetos.

# A Física nos Jogos

Para podermos entender bem o que faremos ao longo desta aula, começaremos nossa discussão com uma reflexão sobre como a física afeta a nossa vida e como afeta, naturalmente, elementos de jogos.

No mundo real, estamos na Terra, a qual possui uma força gravitacional conhecida em torno de 9,8m/s². De acordo com a Lei da Gravitação Universal, essa força gravitacional atrai objetos com massa. Sendo assim, se formos ao topo de um prédio e de lá soltarmos uma pedra, esta cairá, acelerando conforme as forças recebidas, até chegar a uma velocidade terminal e acertar o chão. O chão está fixo e retornará uma força sobre a pedra, na direção oposta à força exercida por ela sobre o chão, utilizando a energia da queda. Tudo isso pode ser calculado, demonstrado e ter parte de seus valores conhecidos por meio de fórmulas já vistas, capazes de aproximar bastante os valores esperados para a velocidade da pedra, a força do impacto, etc., dos valores reais que seriam medidos em um experimento como esse.




Nada disso soa estranho, correto? Uma pedra pesa, nós a soltamos do alto e ela cai. O chão a recebe, causa uma colisão e, depois, faz a pedra ficar lá, parada, quando toda a energia é dissipada. Isso é o mundo real. É o que crescemos vendo e experimentando, então, é bem simples de perceber! E é o que, usualmente, buscamos em nossos universos de jogos também.

Ao criar um novo elemento gráfico no computador, ainda não há nada disso. Se você criar um quadrado e, em seguida, colocar uma bola em cima, quando executar o seu código, nada se moverá. Eles não cairão, não se baterão e não terão qualquer comportamento físico. Isso tem um motivo simples: a física não está lá. Da mesma maneira que, ao criarmos um novo elemento gráfico estamos apenas acendendo pixels na tela para enganar o olho a achar que algo está ali, até mesmo com uma profundidade, em uma tela 2D, precisamos enganar o nosso cérebro também a fim de fazer a física parecer existir em um mundo de pixels acesos. E isso dá um trabaaaaaalho!

---

A física é uma ciência complexa, como vocês veem com mais detalhes na disciplina de Física para Jogos. E toda essa complexidade possui um alto custo para ser transmitida ao computador. É bem complicado simular tudo adequadamente, pretendendo fazer as coisas que ali estão serem reais e funcionarem exatamente como no mundo real. Ainda bem não ser isso o que, em geral, queremos! Lembra de eu ter falado sobre a física possuir modelos capazes de conseguir valores muito similares ou reais utilizando cálculos simples? Pois é! É assim que conseguimos resultados muito interessantes!

A gravidade da Terra não é precisamente 9,8m/s². Existe, além disso, a resistência do ar e as deformações que o objeto pode sofrer enquanto está submetido a uma força, de modo a alterar algumas de suas propriedades. Felizmente, tudo isso pode ser abstraído e simulado de uma maneira mais simples, fazendo-nos conseguir, com um custo computacional bem menor, um efeito bom o bastante para os nossos jogos. Arredondamos a gravidade, ignoramos a resistência do ar e, de repente, temos uma pedra caindo, parecendo suficientemente real. Colocamos um quadrado flutuando na parte inferior da tela e adicionamos a ele propriedades de colisão e BOOM! A pedra bate nesse quadrado voador, o qual podemos chamar de chão, e tudo parece estar funcionando! É justamente com essa mágica que queremos trabalhar. A ideia de utilizar física nos jogos, principalmente no caso de jogos 2D mais simples, como estamos trabalhando, é criar os melhores modelos, com abstrações às vezes absurdas, mas que alcançam o resultado desejado!

Melhor ainda: nem precisamos nos preocupar em implementar todos esses modelos, fazer todos esses cálculos ou otimizar isso a fim de ser feito de uma maneira computacionalmente barata. O nosso motor de jogos já faz tudo para nós!

Na verdade, essa parte de física é tão brutal que, usualmente, há um motor inteiro dentro do motor de jogos voltado simplesmente a ela. O chamado motor de física é uma parte crucial de todo e qualquer motor que deseje uma simulação minimamente realista da física, para garantir que o jogo possa responder de maneira adequada.

> **Motor de física**, do inglês *physics engine*, é um tipo de motor de jogo que simula a física de Newton em modelos 3D, usando variáveis como massa, velocidade, fricção e resistência ao ar, fazendo possível simular condições da vida real. Ele é usado principalmente em simulações científicas e em videogames.
>
> **Fonte:** Wikipédia. Disponível em: https://pt.wikipedia.org/wiki/Motor_de_f%C3%ADsica. Acesso em: 24 de jan. 2017

É claro que existem alguns jogos que não possuem física propriamente dita, como os RPGs de Tiles 2D (RPG Maker), mas, em compensação, temos, por outro lado, jogos sempre buscando uma perfeição física maior, investindo em motores de física cada vez mais precisos e potentes, abusando de todo o poder de processamento possível. PhysX, Havok Engine... Esse é um tema sobre o qual poderíamos conversar o dia inteiro, pois é muito interessante e completo! Há realmente bastante coisa sendo feita nessa área, com diversas novidades apresentadas constantemente e muita gente interessada em tudo.

Infelizmente, não temos tanto tempo e há ainda muita coisa para aprender sobre o motor de física 2D do Unity. Sugiro, a quem tem um interesse maior no assunto, que pesquise mais sobre esse tema. Garanto que não se arrependerá!

O mais importante de toda essa discussão é lembrar que a física, em jogos, por motivos óbvios, simplesmente não existe! Mas nós conseguimos, sim, utilizar modelos complexos e extremamente bem implementados para criar uma sensação de que ela está lá e “enganar”, da melhor maneira, o nosso jogador a acreditar ser possível encarar a física do jogo do mesmo modo que no mundo real! Ou não, não é? Afinal, às vezes, mudar a física do jogo já cria uma nova experiência de jogabilidade e pode ser um caminho interessante para se desenvolver novos jogos! Repetindo: tem muita coisa legal nessa área! ;)

# O Motor de Física do Unity

Como vimos, a implementação da física em um jogo não é uma coisa trivial. Existem diversos modelos conhecidos, com muitos detalhes, que precisam ser implementados de uma maneira a não se tornarem tão pesados para o processador e, ainda assim, parecerem reais ao usuário. Objetivando facilitar a nossa vida em todos esses aspectos, o Unity traz implementado um motor de física excelente, com diversas funcionalidades que aprenderemos ao longo desta aula, as quais podem nos ajudar muito quando criarmos os nossos jogos em 2D.

O interessante do motor de física do Unity é que, além de haver uma implementação muito cuidadosa de todos os elementos lá presentes, também há uma variedade muito grande de elementos! Colliders, Effectors, Joints e até o próprio Rigidbody: são todos elementos com funcionalidades diferenciadas, os quais, no entanto, podem, em conjunto, representar praticamente qualquer comportamento físico que você queira adicionar ao seu jogo! E isso é muito importante, pois adicionar modificações a comportamentos físicos, ou mesmo buscar adicionar ao jogo alguma funcionalidade física ausente no motor, via scripts ou código, não é uma boa ideia. Como já discutimos, essa é uma parte bem delicada para o processamento e quanto menos precisarmos mexer nela, melhor. Com o Unity, em 2D, posso garantir ser possível apenas utilizá-lo, sem precisar de modificação alguma! Oba! :D

Outro aspecto interessante trazido pelo Unity recentemente foi a separação do motor de física em dois. Temos o motor de física 3D e o motor de física 2D como dois motores realmente separados. Apesar de eles possuírem, quando faz sentido, os mesmos componentes e, em geral, apresentarem comportamentos iguais, esses motores não interagem entre si. O motor 2D trabalhará com componentes 2D, e o motor 3D com os componentes 3D. Não há como forçar uma interação.

---

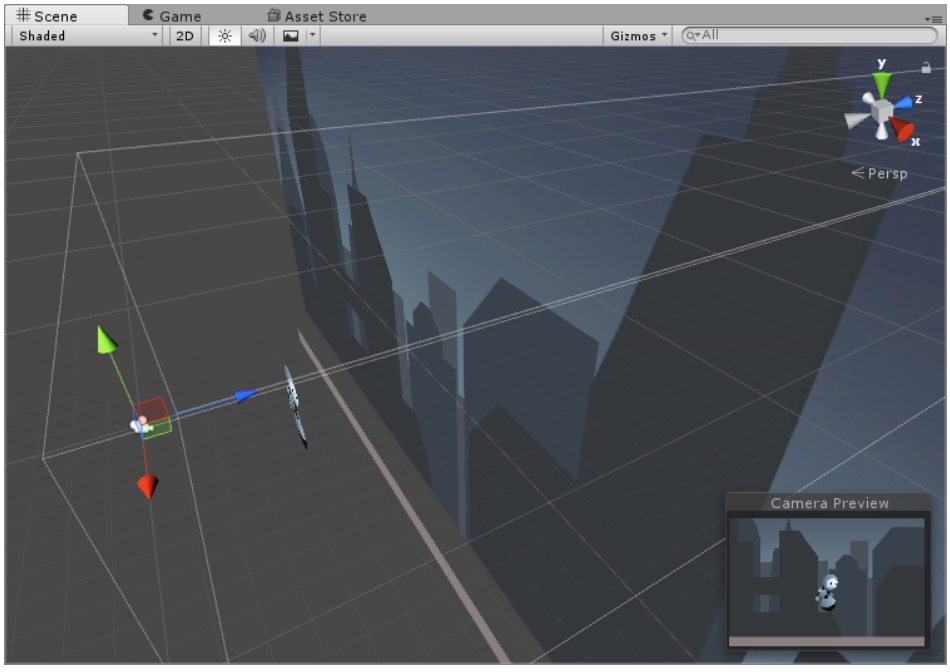
Apesar disso, é possível aos dois motores coexistirem em uma mesma cena. Podemos ter alguns componentes respondendo ao motor de física 2D, e outros respondendo ao motor de física 3D, no entanto, eles nunca poderão interagir entre si, independentemente de qual comportamento cada componente tiver. Um

componente 3D atravessará qualquer colisor 2D e vice-versa. Então, em geral, busque sempre utilizar só um dos motores! Eles foram separados por uma razão e você deve obedecer a esse comportamento.

Para o nosso caso, como trabalharemos com o desenvolvimento de jogos 2D, utilizaremos constantemente o nosso motor de física 2D. Em nossa primeira aula, vimos o motor 3D em ação quando tratamos de um exemplo rápido, mas não entramos em detalhes sobre ele e não entraremos mais. Precisamos deixar algo para o professor de Motores II poder ensinar também, não acha?

Falando do motor 2D do Unity, começaremos por uma questão muito importante: o motor de física 2D do Unity é 2D! Isso pode parecer bobagem, além de óbvio, mas não é! E eu explicarei o motivo, para vocês não acharem que estou zoando. Como vimos em aulas anteriores, o Unity, mesmo no modo 2D, ainda nos permite trabalhar com objetos em um mundo 3D. Podemos posicionar objetos em posições diferentes do eixo Z. Na verdade, para que o jogo funcione, isso precisa acontecer! A câmera, como vimos anteriormente, é, por padrão, posicionada no Z = -10. Os objetos são posicionados no Z = 0 e, com isso, ficam na frente da câmera e podem ser exibidos. Veja a **Figura 1**, na qual são mostrados os elementos deslocados no eixo Z.

**Figura 01** - Personagem, chão, fundo e câmera deslocados no eixo Z, mas retratados no mesmo plano no Camera Preview.

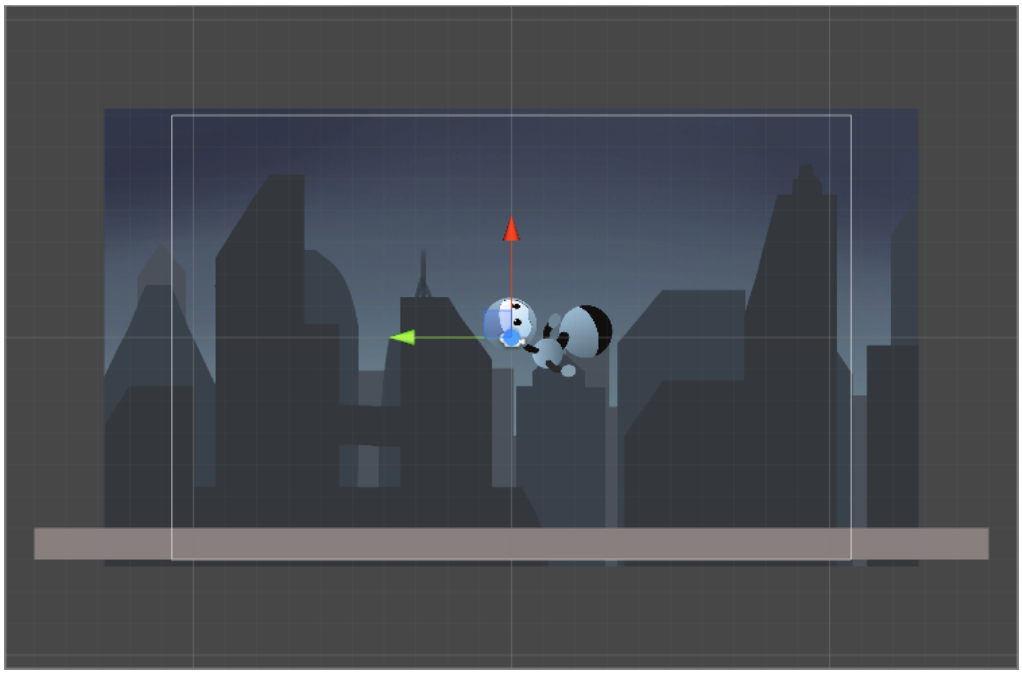



**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Sendo assim, nada impede que posicionemos o cenário no Z = 5, os objetos de frente no Z = 3, e o personagem no Z = 0 para ficar mais próximo à câmera. Nada impede, também, que façamos uma bobagem e coloquemos em Z um objeto não desejado, e nunca nem notemos, pois ele aparecerá do mesmo jeito, em 2D. Apesar disso tudo, como o nosso motor de física 2D é, de fato, 2D, tudo isso é ignorado na hora dos cálculos de física! O objeto que está no Z = 5 colide normalmente com o que está no Z = 3, e o personagem Z = 0 pode cair naturalmente em cima do chão Z = 4 sem passar por trás ou pela frente dele, de modo a não haver colisão. Essa facilidade é importante, apesar de usualmente passar despercebida. Há, também, todo o benefício dos custos computacionais dos cálculos feitos: uma dimensão a menos!

Outra coisa interessante nessa linha é que, por trabalhar apenas com os eixos X (vermelho, na **Figura 2**) e Y (verde, na **Figura 2**) para os cálculos físicos, o motor de física permite apenas ocorrerem rotações em torno do eixo Z, sendo este o eixo que fica perpendicular à tela do computador. Com isso, é possível rotacionar os objetos apenas de uma maneira a ser percebida pelo usuário. Veja na **Figura 2** a descrição desses eixos para facilitar o entendimento. Lembre-se que a referência dos eixos é o personagem, e não o mundo. Por isso o eixo Y (verde) está na horizontal e o X (vermelho) na vertical!

**Figura 02** - Personagem rotacionado em 90° em torno do eixo Z (eixo azul). Essa rotação, em 2D, é a única que pode ser percebida.



**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Então, aprendemos que o Unity tem dois motores de física diferenciados, que trabalharemos com o 2D e que o motor 2D já facilita a nossa vida ao trabalhar, de fato, em 2D. Além disso, identificamos a rotação única em torno do eixo Z. Sabendo disso tudo, podemos seguir ao próximo passo: entender melhor os elementos 2D trazidos pelo motor de física do Unity. Primeiramente, trataremos dos componentes os quais podem ser adicionados aos nossos elementos e, em seguida, conheceremos os materiais físicos que podemosde utiliza para simular alguns comportamentos no Unity.

Antes disso, que tal testarmos os nossos conhecimentos um pouco mais? Vamos lá!

## Atividade 01

1. Como o Unity diferencia a física 2D da física 3D?
2. É possível utilizar física 2D e 3D em uma mesma cena no Unity? É recomendado? Justifique.

## Os Componentes do Motor de Física 2D

O Unity trabalha com componentes 2D distintos, os quais podemos adicionar aos nossos elementos para que estes se comportem de maneira adequada. Dizemos que um objeto se comporta de maneira adequada quando segue corretamente as leis da física, como esperado no jogo. Isso inclui acelerar e frear do modo esperado, ser afetado por colisões e responder a elas de maneira adequada, ser afetado pela gravidade e outras forças... Enfim, são diversos comportamentos a serem seguidos e o Unity, com seus componentes, permite que possamos programar para cada um deles.

O Unity trabalha com vários componentes de tipos diferentes. Devido ao tempo e à praticidade, traremos apenas o conceito principal de cada um deles e falaremos um pouco de seus funcionamentos. Os detalhes de todos os componentes, como

sempre, podem ser encontrados no manual do Unity, referenciado nas leituras complementares para mais informações. Conversaremos aqui sobre os quatro tipos principais e deixaremos as suas derivações como tema de pesquisa para vocês!

## Rigidbody 2D

O componente Rigidbody 2D já é um conhecido nosso! Ele foi utilizado, na aula passada, para o nosso personagem poder receber uma força, baseado no Input do usuário. Isso é apenas uma das utilidades do Rigidbody 2D. Podemos dizer, de maneira geral, que **esse componente serve para adicionar um objeto, o qual antes era somente gráfico, ao motor de física**. Tal componente possui diversas propriedades relacionadas a como o objeto irá interagir com esse motor.

Um ponto interessante relacionado ao funcionamento do Rigidbody 2D é que, ao adicionar esse componente em um objeto, o RidigBody assume o controle do Transform (vetores de posição, rotação e escala, que já vimos, lembra?) e passa a alterar os seus valores conforme os cálculos físicos realizados pelo motor. Isso também acontece com os colisores, vistos a seguir, que estiverem ligados ao componente - o Rigidbody 2D será capaz de movê-los e gerenciá-los de acordo com o motor de física do Unity. Embora você ainda seja capaz de alterar o Transform do objeto diretamente, é importante saber que isso afeta bastante o nosso motor de física, pois essa movimentação é inesperada dentro da simulação e pode requerer a realização de novos cálculos.

Dito isso, saiba que o Rigidbody 2D é o modo mais fácil de fazer um objeto passar a responder de maneira realista às forças físicas em geral. É através dele, principalmente, que ocorre a colisão com outros objetos (em conjunto com os colisores), pois as colisões podem resultar em forças as quais movem o objeto, e isso será mais uma das atribuições do Rigidbody 2D. Para simplificar: se você quer que um objeto se mova e obedeça a forças físicas, adicione a ele um Rigidbody 2D.

Perceba, no entanto, que colisões não são feitas pelo Rigidbody 2D e, sim, pelos colisores. A atribuição do Rigidbody 2D, nesse caso, é de gerenciar as forças resultantes das colisões detectadas e fazer o elemento que o tem reagir adequadamente, alterando seu Transform para haver uma movimentação.

A primeira propriedade de um Rigidbody 2D é o **Body Type**. Ela possui três valores, e a escolha de um deles afeta diretamente a interação do objeto com Colliders e a maneira com a qual o objeto se move e rotaciona. Note que isso tudo influencia diretamente no comportamento do Rigidbody 2D, e alterar algo durante a execução do jogo pode ser uma péssima ideia, devido a todos os cálculos e comportamentos a serem refeitos. Pense bem sobre o que você quer que o seu objeto faça e mantenha esse Body Type ao longo de toda a execução. As três opções são:

- **Dynamic** - Esse é o tipo de Body Type mais utilizado. É um objeto feito para se mover de acordo com a simulação. Possui a maior quantidade de propriedades físicas alteráveis e é afetado por forças e colisores em geral. Isso torna esse colider o mais caro e, também, o mais suscetível a problemas, caso você altere o Transform diretamente. **Evite a todos os custos fazer essa alteração manual.**

- **Kinematic** - Objeto que só deve se mover em situações específicas, de acordo com comandos do usuário, indicando ao Rigidbody 2D a posição/rotação do objeto, e não diretamente via Transform. Não é afetado por forças e não possui propriedades físicas de massa. Funciona como um objeto não movível em colisões (massa infinita) e não colide naturalmente.

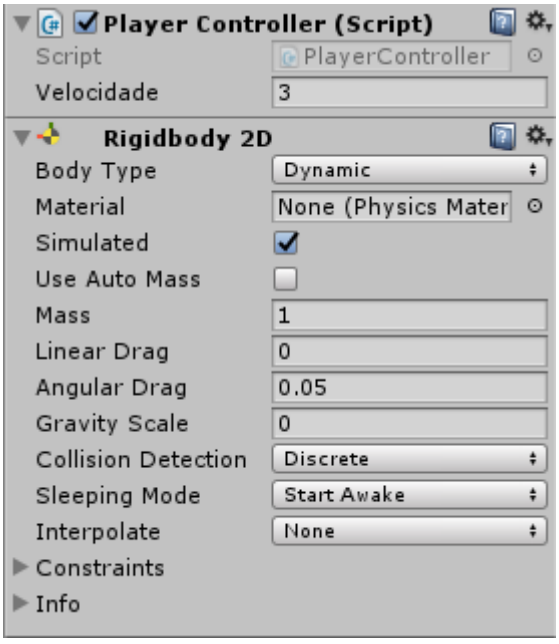
- **Static** - Objeto que não deve se mover de maneira alguma durante a simulação. Caso algum objeto entre em contato com ele, o Static funcionará como um objeto estático, com massa infinita. É capaz apenas de colidir com objetos do tipo Dynamic, já que não se move. Só possui propriedades de ativar ou desativar.

Em resumo, nós temos o tipo Dynamic, que se move com forças e reage à maior quantidade possível de outros elementos, o tipo Kinematic, que só se move quando é explicitamente dito para se mover em uma direção e não lida com colisões (exceto com objetos Dynamic), e o tipo Static, representando objetos que não devem se mover. Qual tipo utilizar dependerá diretamente do seu jogo e, mais especificamente, de qual objeto esse Rigidbody 2D estará ligado.

Você pode estar se perguntando o motivo de um componente responsável por fazer objetos responderem ao motor de física, ter uma configuração que o faz ficar estático. A resposta é simples. Alguns outros componentes de física dependem de um Rigidbody 2D estar ligado ao objeto para que possam funcionar. É através dele que esses componentes ganham acesso ao objeto e, então, é necessário adicionar um Rigidbody 2D de qualquer maneira. Se for num objeto que não deveria se mover, o adicionamos estático! Entende?

E as outras propriedades? Bom, vejamos a **Figura 3** com a lista de todas as propriedades presentes no Rigidbody 2D do nosso amigo robô, para podermos discutir cada uma delas rapidamente. Lembre-se que, ao alterar o **Body Type**, todas as propriedades disponíveis são também alteradas. Veremos aqui especificamente as propriedades da opção Dynamic, que é a mais comum e possui mais propriedades. Para mais detalhes sobre as outras opções, consulte o manual.

**Figura 03** - Rigidbody 2D do nosso player, com o script que o controla.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

A primeira propriedade, **Body Type**, já discutimos e vimos que escolhemos a opção correta para ela desde o começo, pois essa é a opção mais utilizada e selecionada por padrão pelo Unity. Seguindo adiante na lista, a segunda propriedade é a **Material**.

É possível criar materiais físicos, de modo a alterar as propriedades físicas de objetos. A propriedade **Material**, dos Rigidbody 2D, serve justamente para atrelar a um elemento físico um desses materiais. Veremos adiante que os Colliders em geral também possuem essa propriedade. Quem tem o material, na verdade, são os Colliders. Os Rigidbodys 2D apenas o tem para caso o Collider não especifique nenhum. Se o Collider não tiver o seu próprio, será utilizado o do Rigidbody 2D. Se nenhum dos dois tiver, não haverá material algum para aquele objeto.

Usa-se a propriedade **Simulated** a fim de ativar e desativar a simulação física para o objeto atual. É como se tivéssemos desabilitando o Rigidbody 2D daquele objeto. Mas atenção! **Não é a mesma coisa de desabilitar**. Quando estamos lidando com um Rigidbody 2D, ele é um elemento a mais que é criado dentro do motor de física e simulado. Se nós desabilitarmos esse Rigidbody 2D, ele será removido do motor. Caso o recriemos, será novamente adicionado. Isso consumirá tempo e memória. Quando simplesmente desabilitamos a opção Simulated, o objeto continua lá, mas passa a ser ignorado, facilitando toda a computação e devendo ser sempre a decisão tomada, quando necessário.

A **Use Auto Mass** é uma propriedade trazida pelo Unity para que a massa do objeto seja calculada automaticamente. Se marcarmos essa opção, o Unity procurará por Colliders atrelados ao objeto e, baseado na área desses Colliders e na densidade que eles possuem, calculará a massa aproximada dele. Isso tira um pouco da capacidade de personalizar o objeto que você tem, mas pode ser uma boa opção para se utilizar com efeitos como o de empuxo, por exemplo.

A propriedade **Mass** permite a você indicar a massa do Rigidbody 2D. Perceba que essa opção é desabilitada caso o cálculo automático de massa seja habilitado na propriedade anterior.

A propriedade **Linear Drag** indica qual o atrito que afetará a movimentação de posição do objeto, enquanto a propriedade **Angular Drag**, por sua vez, indica o atrito para rotação do objeto.

A propriedade **Gravity Scale** indica como a gravidade afetará esse objeto. A gravidade em si é definida nas configurações de física do motor, as quais podem ser acessadas pelo menu Edit -> Project Settings -> Physics 2D, juntamente com diversas outras propriedades do motor de física que podem ser alteradas. Vale a pena conferir por lá!

## Atenção!

É interessante notar que a **Gravity Scale** pode receber um valor negativo também, fazendo com que o objeto seja afetado pela gravidade ao contrário. Isso já é uma opção para se pensar em colocar no seu jogo!

A propriedade **Collision Detection** indica a maneira como o Unity deve checar colisões. Não entraremos em muitos detalhes nessas opções, mas saiba que elas afetam dois pontos. A detecção de colisão contínua pode evitar problemas de um objeto atravessar o outro por tê-lo "pulado", mas tem um custo bem mais caro que a detecção de colisão discreta. Para entender melhor a diferença, imagine uma bala de rifle sendo atirada contra uma janela fina de vidro. Pode ser que, devido a velocidade da bala, num instante T0 ela esteja antes da janela, e no instante T1 depois desta, sem nunca ter estado em cima/no meio dela. Nesse caso, a colisão não seria detectada de maneira discreta.

A propriedade **Sleeping Mode** define o modo como o objeto fica enquanto não está sofrendo nenhuma força, para economizar um pouco de processamento. Com o modo Start Awake, ele começa checando se há algo por perto e somente depois dorme, economizando recursos. No modo Start Asleep, ele começa sem checar nada e fica aguardando alguém interagir com ele, para voltar a consumir recursos e calcular o que deve ser feito. Com o valor Never Sleep, ele nunca será desabilitado.

A propriedade **Interpolate** indica como o objeto deve se mover entre cada atualização do motor de física. Se for escolhida a opção None, nenhuma interpolação será feita. Com a opção Interpolate, a movimentação do objeto será suavizada, baseada em posições anteriores. Com a opção Extrapolate, a posição também será suavizada, mas baseada em uma predição de onde o objeto deverá estar no próximo frame.

As últimas propriedades que temos dizem respeito a restrições. O conjunto de propriedades dentro da aba **Constraints** indica se deve haver algum congelamento do nosso Rigidbody 2D em relação à movimentação ou rotação. Freeze Position X ou Y indica que o objeto não deverá se mover naquele eixo de maneira alguma. Freeze

Rotation Z indica que o objeto não deverá rotacionar. Isso é importante para evitar, por exemplo, ao nosso amigo robô cair com a cara no chão à medida que uma força o empurre.

Por fim há a aba **Info**. Apesar de não ser uma propriedade, pois não tem nenhum valor alterável, essa aba é muito útil para podermos, ao dar play em nosso jogo, acompanhar informações importantes sobre o nosso corpo rígido em tempo real e, com isso, debugar alguns problemas que possam surgir sem que saibamos precisamente como. É possível observar informações, como a velocidade nos dois eixos, a posição, a rotação, se o objeto está acordado ou não, entre outras. Ou seja, essas diversas informações importantes contribuem para entendermos melhor o que o motor está gerando para nós.

Assim, concluímos o primeiro e mais complexo dos componentes físicos estudados nesta aula. Agora conheceremos os outros componentes disponibilizados pelo Unity, para podermos, enfim, adicionar um tanto de física ao nosso projeto!

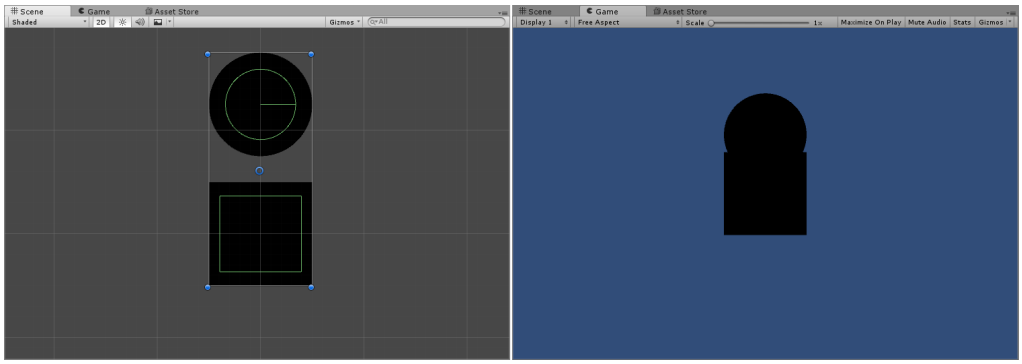
---

## Conhecendo os Colliders 2D

Os Colliders 2D são os objetos responsáveis por tornar possíveis as colisões entre objetos no mundo físico simulado de nosso motor. Eles são áreas fechadas que definem o espaço a ser ocupado por um objeto em nosso motor físico. É por meio do tratamento dessas áreas que o motor de física consegue detectar se dois objetos encostaram ou não.

Para que as colisões aconteçam adequadamente, os objetos devem ter colisores que representem bem a sua forma. Caso isso não aconteça, elas podem ocorrer de maneira estranha e, principalmente, passar informações visuais erradas ao usuário, dando a impressão de que um objeto está dentro de outro, ou algo do tipo. Conhecemos esse efeito como clipping. Isso pode ser evitado utilizando a forma correta de colisor. O Unity nos dá cinco opções de colisores 2D: Circle Collider 2D, Box Collider 2D, Polygon Collider 2D, Edge Collider 2D e Capsule Collider 2D. Cada um deles deve ser utilizado para cobrir uma forma diferente. Vejamos, na **Figura 4**, uma caixa com um Box Colider e um círculo com um Circle Collider. No Unity, os Colliders são representados com linhas verdes para termos uma referência visual de como eles estão posicionados.

**Figura 04** - À esquerda, objetos com Colliders (em verde) menores que seus formatos. À direita, os objetos em colisão inadequada devido ao tamanho dos Colliders.



**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Vemos, na **Figura 4**, o posicionamento dos Colliders sendo realizado de maneira errada, justamente para exemplificar os efeitos que isso pode causar visualmente em seu jogo. Perceba a importância de ajustar as propriedades de cada um dos Colliders, para que sempre passem a melhor impressão visual ao usuário, realizando as colisões adequadamente.

Já em relação às propriedades dos Colliders, elas estão em quantidade bem menor que no Rigidbody 2D. Algumas variações existem entre as diversas formas, mas, em geral, todas elas possuem as mesmas propriedades. Caso estejam ligados a um objeto que tenha um Rigidbody 2D com a opção **Use Auto Mass** ativa, como vimos anteriormente, a primeira propriedade do Collider 2D é a **Density**. Essa propriedade indica a densidade tida pelo objeto e será utilizada para calcular a massa total, com base na área.

Já a propriedade **Material**, como já discutimos, indica o material físico utilizado por esse Collider. Se nenhum for escolhido, ele utilizará o do Rigidbody 2D atrelado. Se também não houver, nenhum material será utilizado.

Em seguida, temos a propriedade **Is Trigger**. Ela é muito importante e é utilizada para criar objetos que funcionam apenas como gatilho para eventos, mas não possuem nenhuma função física propriamente dita. Quando essa opção é ativa, colisões com esse objeto dispararão um evento, porém não serão calculadas fisicamente. Ou seja, é possível atravessar o objeto, mesmo com o colisor, disparando somente um evento quando isso acontecer. Essa propriedade é útil para diversas coisas, como criar moedas coletáveis no Mario, por exemplo.

A última opção que os Colliders compartilham, antes das opções de tamanho e posicionamento individual que cada forma possui diferenciada, é a opção **Used by Effector**. Essa opção indica que o Collider é utilizado por um Effector para que este possa realizar as suas ações. Assim, é necessário que o Collider tenha essa opção marcada, ou o próprio Unity exibirá um erro dizendo que há um Effector sem nenhum Collider atrelado a ele no objeto. Mas o que é um Effector? Vamos descobrir!

Ah! Ia esquecendo! Os Colliders também apresentam a possibilidade de abrir as informações através da aba **Info** dentro do próprio componente. Uma ótima ferramenta de depuração!

---

## Trabalhando com Effectors 2D

Effectors são objetos físicos capazes de influenciar uma área com forças físicas provenientes de origens diversas, de acordo com cada tipo de atuador. É possível utilizar esses objetos para gerar zonas com empuxo para líquidos, simular efeitos de explosões ou simplesmente gerar uma área com atração/repulsão. Os efeitos são variados, mas todos são obtidos a partir desse componente!

Para adicionar um Effector a um GameObject qualquer, é necessário que o objeto possua um Collider 2D com a opção Used By Effector marcada. Feito isso, basta adicionar o Effector como um componente de física 2D normal. É importante também para alguns atuadores que o colisor ao qual ele está associado seja do tipo Trigger, para garantir que o efeito acontecerá se qualquer parte dos objetos estiver em contato. Caso o Collider não seja um Trigger, o Effector ainda irá funcionar, mas só atuará quando houver o contato do objeto com a borda do colisor, em vez de funcionar para qualquer local da área marcada pelo colisor.

O Unity disponibiliza cinco Effectors diferentes para o motor de física 2D. Vejamos a seguir, na **Tabela 1**, quais são esses Effectors e o que eles são responsáveis por fazer. Não entraremos em muitos detalhes desses componentes, mas apresentaremos cada um deles para que vocês saibam qual utilidade é mais adequada quando desejarem adicionar algum dos efeitos demonstrados em seus jogos. As configurações de propriedades desses Effectors podem ser encontradas também no manual do motor de física do Unity que, mais uma vez, referenciaremos na leitura complementar.

| Effector | Utilização |
|---|---|
| Area Effector 2D | Exerce uma força em uma área à medida que os objetos entram em contato com ela. É possível configurar o ângulo em que a força será exercida, a magnitude da força e a sua variação, além do atrito que ela gerará com o objeto. |
| Buoyancy Effector 2D | O atuador de empuxo simula comportamentos simples de líquidos dentro do motor de física. É possível configurar a densidade do líquido, o atrito gerado, o fluxo e qual a localização da superfície para finalizar as forças. |
| Point Effector 2D | O atuador de pontos gera uma força de atração/repulsão ao redor de um ponto, de acordo com a magnitude configurada (positiva ou negativa). É possível escolher a maneira com que a força atuará, se constante, em escala linear ou quadrática. Também há uma distância máxima de atuação da força que é possível configurar. Funciona bem para explosões! |
| Platform Effector 2D | Como o nome indica, esse atuador simula diversos comportamentos de plataformas, como colisões de um lado só, remoção da fricção lateral, etc. Diferentemente do padrão, esse atuador precisa que o colisor ao qual está atrelado não seja do tipo Trigger, para que haja a colisão física necessária. |
| Surface Effector 2D | Esse atuador gera forças tangentes aos corpos que estão em cima de seus colisores para que estes adquiram alguma velocidade. Funciona como se fosse uma esteira, se movendo para carregar algo de um ponto a outro. Assim como no Platform Effector 2D, é necessário que o Collider ao qual este atuador está ligado não seja do tipo Trigger, para que haja a colisão física e, devido a isso, possa haver a aplicação da força. |

**Tabela 1** - Tipos de Effectors e seus funcionamentos.
**Fonte:** Elaborada pelo autor.

Também é importante notar que em todos os atuadores é possível definir qual é a fonte da força e qual é o alvo da força, se o Rigidbody 2D ou o Collider do objeto. Caso o alvo seja diretamente o Rigidbody 2D, este será simplesmente empurrado

por aquela força na direção que ela indica. Já no caso do alvo ser o Collider, a força empurrará fisicamente o Collider, podendo gerar, por exemplo, rotações. É como se a primeira maneira representasse uma força como a gravidade e a segunda um empurrãozinho!

---

## Trabalhando com Joints 2D

O Unity também apresenta, em seu motor de física 2D, um conjunto de tipos de Joints - ou juntas, em português - que podem ser utilizados para criar sistemas complexos, ao conectar-se umas às outras, o tanto quanto se queira. Pense nas juntas como sendo um objeto que exerce a função de um conector entre dois outros objetos, por exemplo a dobradiça de uma porta, permitindo que ela se mova no arco, ou os amortecedores de um carro, permitindo que os pneus e o chassi trabalhem de maneira um pouco mais independente. É possível criar desde interações simples, como uma porta que abre e fecha, até algo mais complexo, como uma corrente balançando (com as juntas agindo como os elos) ou uma máquina em funcionamento.

Esses componentes possuem diversas especificidades e tomariam duas ou três aulas para que pudéssemos discuti-los detalhadamente. Mais uma vez, como fizemos com os atuadores, listaremos todos os que estão disponíveis dentro do motor de física 2D, juntamente às suas utilizações, e então deixaremos a vocês, à medida que forem necessitando no desenvolvimento de seus jogos, a tarefa de se aprofundar em cada um. O mais importante nessa parte da aula é que aprendam a escolher seguramente, dentre os diversos tipos de juntas disponíveis, qual será a capaz de resolver o problema que vocês estão enfrentando no desenvolvimento. Vejamos a **Tabela 2** com todos os Joints 2D disponíveis no Unity.

| **Joints 2D** | **Funcionamento** |
|---|---|
| Distance Joint 2D | Liga dois objetos que possuam Rigidbody 2D e os mantêm a uma distância fixa, definida no componente. |
| Fixed Joint 2D | Une dois objetos de maneira fixa, como uma barra de ferro. Os dois objetos se mantêm sempre a uma mesma distância e ângulo. |

| Joints 2D | Funcionamento |
|---|---|
| Friction Joint 2D | Reduz a velocidade dos dois objetos conectados a zero. |
| Hinge Joint 2D | Cria um ponto ao redor do qual um objeto com Rigidbody 2D pode rotacionar ao redor. Pense nisso como o pivô de um alicate, por exemplo, o qual permite que as duas partes abram e fechem ao redor dele. |
| Relative Joint 2D | Mantém um objeto com Rigidbody 2D distante do outro com uma distância fixa. Imagine isso como se um objeto seguisse o outro. Os dois estarão sempre a uma distância fixa um do outro. |
| Slider Joint 2D | Permite que objetos possam deslizar em torno de uma linha, como portas automáticas em um shopping center. |
| Target Joint 2D | Liga um objeto com Rigidbody 2D a um alvo. Pode ser utilizado para pegar objetos, por exemplo. |
| Spring Joint 2D | Conecta dois objetos através de uma mola. |
| Wheel Joint 2D | Simula o comportamento de uma roda com molas. |

**Tabela 2** - Diversas juntas disponíveis no motor de física 2D do Unity.
**Fonte** - Elaborada pelo autor.

Perceba que a grande maioria das juntas é utilizada para ligar dois objetos que possuem Rigidbody 2D, com poucas exceções. A conexão muitas vezes vai trabalhar com objetos do tipo Static, mas que necessitam de um Rigidbody 2D para que haja a conexão. A partir da utilização desses nove tipos de Joints é possível gerar comportamentos diversos para suprir as necessidades do desenvolvedor. Lembrando sempre que é possível combinar mais de uma junta para montar a sua máquina maléfica de dominação do planeta em seu jogo, com diversas partes móveis, conectadas e trabalhando!

## Utilizando a Constant Force 2D

Por fim, mas não menos importante, temos o último componente do motor de física 2D do Unity que veremos nesta aula: a força constante! É um componente simples, mas muito útil em diversas situações.

Esse componente, como o nome diz, simplesmente adiciona uma força constante a um objeto que possua um Rigidbody 2D. Podemos utilizar essa força, por exemplo, para impulsionar carros de corrida, ou aviões, ou qualquer coisa que possua uma aceleração constante ao longo do tempo e não só um grande impacto inicial, como um chute a uma bola de futebol.

O componente Constant Force 2D aplica a força a cada update do motor gráfico do Unity com o mesmo valor e a mesma direção (afinal é uma força constante, não é?). As propriedades que possui são simples:

- **Force X e Y:** Indica a força aplicada ao objeto a cada iteração.
- **Relative Force X e Y:** A força aplicada a cada iteração em relação às coordenadas do Rigidbody 2D.
- **Torque:** O torque aplicado ao Rigidbody 2D a cada iteração.

E com isso, finalizamos toda a parte de componentes do motor de física 2D do Unity. É coisa para caramba, não? O bom de ter tanta coisa assim é que não nos limitamos no que queremos desenvolver! Não temos como ver tudo funcionando com explicações detalhadas, claro, mas é possível que tenhamos a completa noção de como tudo isso funciona para podermos utilizar e desenvolver os nossos jogos sem problema algum.

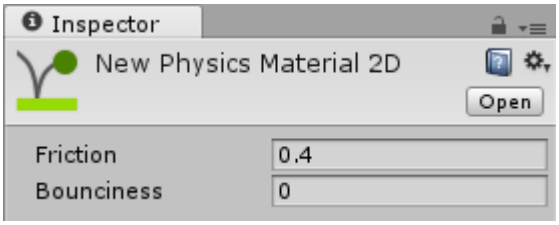
Vamos agora conhecer os materiais físicos 2D e como eles podem afetar o nosso jogo!

# Materiais Físicos 2D

A última coisa nova que conheceremos na aula de hoje serão os materiais físicos 2D. Esses materiais, utilizados nos Colliders e nos Rigidbodies 2D, como vimos anteriormente, podem alterar as propriedades desses objetos de modo a torná-los mais, ou menos, escorregadios, “grudentos” ou até mesmo criar uma sensação de cama elástica.

Para criar um novo material 2D, devemos, primeiramente, criar uma nova pasta em nossos assets para guardar esses materiais, pois será um asset como qualquer outro. Feito isso, basta escolher o menu Asset -> Create -> Physics Material 2D. Criado o novo material, ele aparecerá em nossa pasta de Assets para que alteremos seu nome e suas propriedades. A **Figura 5** mostra as propriedades de um Physics Material 2D.

**Figura 05** - Propriedades de um Physics Material 2D.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Existem apenas duas propriedades, mas que são suficientes para alterar bastante o comportamento de um objeto, de acordo com os valores que assumirem. A propriedade **Friction** indica a fricção que o objeto irá exercer quando houver um atrito com outro objeto. Quanto mais próximo de 0, mais escorregadio o material vai parecer. Da mesma maneira, quanto maior o valor, mais grudento e de difícil movimentação o material vai fazer o objeto parecer.

Já a **Bounciness** indica a propriedade elástica do objeto. Quanto mais perto de 0, mais rígido será o objeto, quanto mais perto de 1, mais o objeto irá quicar. Na verdade, um Bounciness com o valor de 1 indica exatamente que o objeto irá bater e retornar sem que haja qualquer perda de energia. Um valor maior que 1 fará com que o objeto ganhe energia na colisão. No entanto, muito cuidado com isso, uma vez que o objeto poderá juntar energia a ponto de sair voando!

Após criado e configurado o material, basta selecioná-lo em um dos campos de **Material** que vimos anteriormente, seja no Rigidbody 2D ou no Collider 2D que adicionarmos aos nossos objetos da cena.

Para praticar um pouco tudo isso que vimos hoje, vamos adiante desenvolver um pouco mais o nosso projeto, fazendo o nosso amigo robô parar de voar e finalmente tocar o solo!

## Atividade 02

---

1. Quais os cinco componentes físicos que o Unity disponibiliza em seu motor de física?
2. A utilização de materiais físicos modifica de alguma maneira o comportamento de um objeto? Como? Quais os parâmetros que podem afetar o comportamento?

## Adicionando Física ao Projeto DMJ I

---

Agora, continuaremos o desenvolvimento do nosso projeto do joguinho do robô! Àqueles que não estão em dia com o projeto (que feio!), é importante lembrar que vocês podem baixar os recursos do que foi desenvolvido na última semana neste link.

Para começar, vamos abrir o projeto e a nossa cena já desenvolvida, com o nosso robô voador. Em seguida, vamos começar devolvendo a gravidade que foi roubada de nosso amigo. Para fazer isso, selecionemos o robô (player) na aba Hierarchy e então modifiquemos a propriedade **Gravity Scale** de nosso Rigidbody 2D de volta para 1. Com isso o nosso robô voltará a cair quando clicarmos em Play!

No entanto, o coitado irá cair... e cair... e cair... para sempre! Isso acontece pois não há ainda uma colisão dele com o chão, correto? Então, para que possamos detectar essa colisão, como vimos no decorrer da aula, precisamos adicionar Colliders em nossos componentes para que eles possam deixar o motor de física do Unity fazer a magia dele! Primeiramente, vamos adicionar um Box Collider ao nosso chão, uma vez que ele é um grande quadrado. Se você quiser utilizar um outro

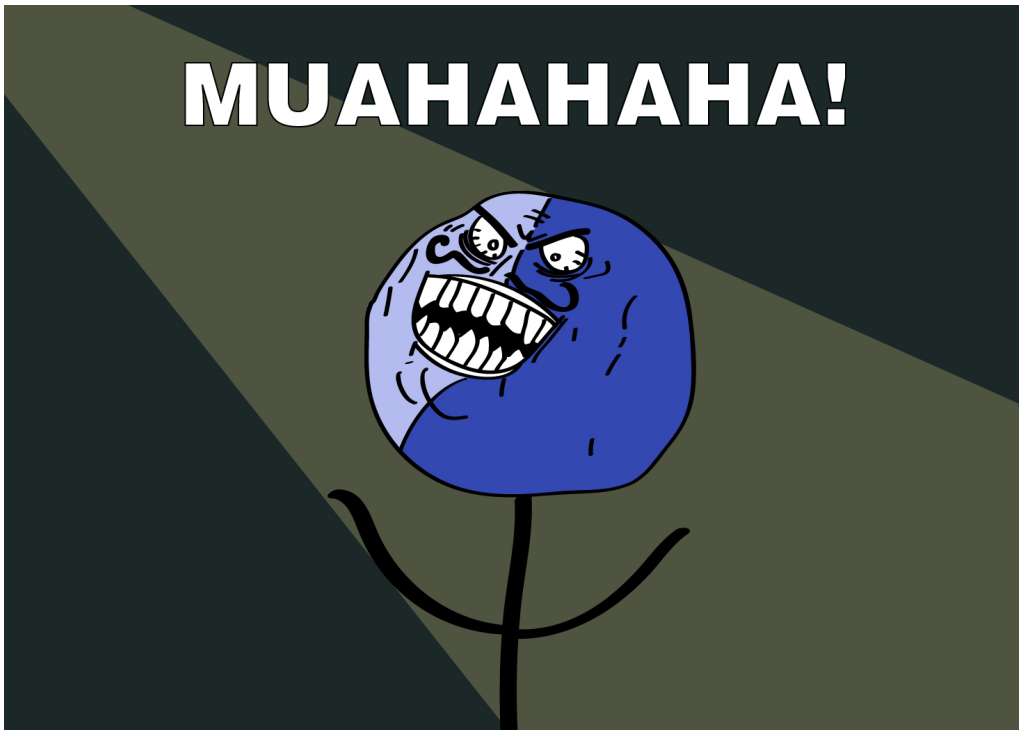
colisor para isso, também temos outras opções, também vistas ao longo da aula. Fique à vontade! Só tenha cuidado para não deixar nenhuma ponta sobrando ou algo do tipo.

Para adicionar o Box Collider ao chão, selecione o chão na aba Hierarchy e em seguida clique em Add Component -> Physics 2D -> Box Collider 2D. Lembre-se de selecionar sempre a opção de 2D! Não queremos misturar os dois motores em uma mesma cena, certo? :)

Agora que o nosso chão já está colidindo adequadamente, vamos ao próximo objeto que precisa de uns colisores: o robô! E para o robô, vamos fazer algo um pouco diferente, adicionaremos dois colisores!

- Oi? Dois? Para quê, tio? Vocês questionam.

E eu respondo: - Cenas dos próximos capítulos! Muahahaha!



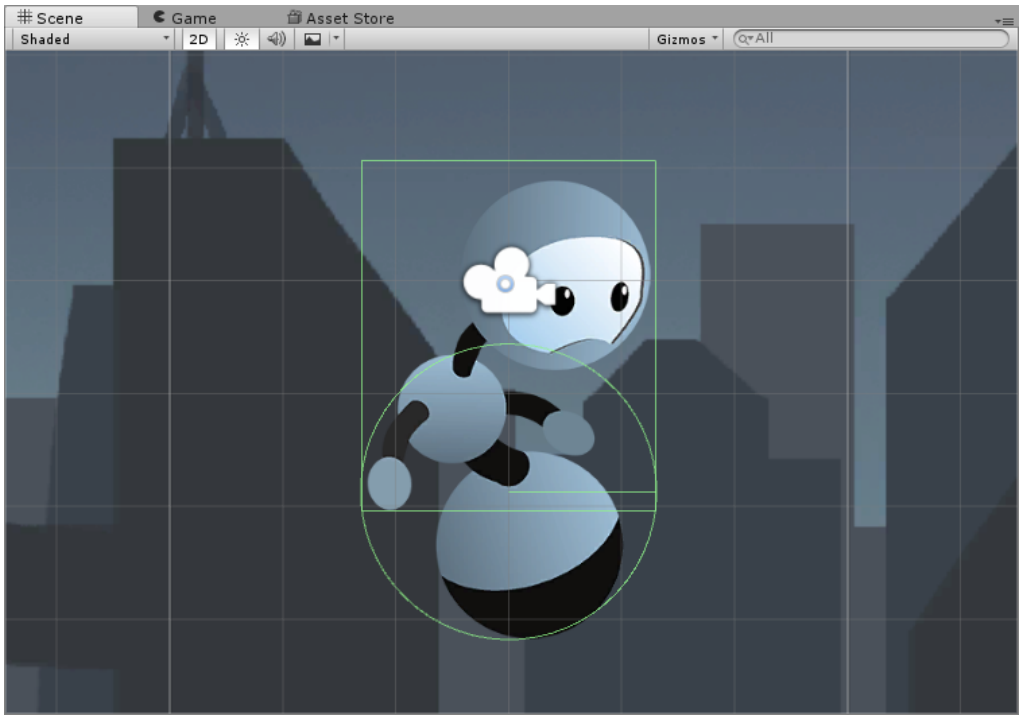

Bem, falando sério! É realmente para utilizarmos nos próximos capítulos. Usualmente, quando vamos iniciar uma ação de pulo, é interessante ter um colisor à parte para trabalhar com essa questão do pulo. Então, já se preparando para a aula em que discutiremos essa importante mecânica do 2D, vamos adicionar dois

colisores ao nosso amigo robô: um responsável pelas colisões do corpo e outra simplesmente pelos pés! Se bem que ele nem tem pés, não é? Então pela bolinha de baixo do corpo!

---

No caso, utilizaremos para o corpo um Box Collider que seja capaz de englobá-lo em sua totalidade, e para a parte de baixo um Circle Collider que fique centralizado no robô e termine exatamente no fim do sprite (para evitarmos aquele clipping terrível que discutimos anteriormente). Para facilitar a visualização do que vocês devem fazer, vejam a **Figura 6**, ela exibe o nosso robô com os dois colisores já posicionados em seus locais corretos!

**Figura 06** - Robô com os dois Colliders 2D que serão responsáveis por seu corpo.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Perceba que o Circle Collider está abaixo do BoxCollider e também está cobrindo as duas pontas do Box Collider de maneira a evitar que o personagem fique "preso" pelas pontas do box collider em algum objeto quando estiver caindo. Perceba também que o círculo do colisor se encerra bem junto ao sprite do boneco.

Com isso, o nosso robô será capaz de cair ao iniciarmos o jogo e de parar quando tocar o chão! E aí poderemos movê-lo para um lado e para o outro no chão, até que... boom! Bateu de cara no chão! Pobre robozinho! Ainda bem que não

quebrou (ainda, pois mais adiante adicionaremos animações a ele)!

Vimos também, nesta aula, como evitar que isso aconteça, lembra? Não? Vou deixar você voltar um pouco e pesquisar, então.

Sério! Vai lá. Não espera eu dizer de novo não, vai!

---

Muito bem! Na parte de Rigidbody 2D falamos que podemos adicionar uma constraint de rotação para que o personagem não rotacione (e caia de cara). Vamos adicionar essa restrição ao nosso personagem e tcharãã! Agora sim, ele cai e anda pelo chão tranquilamente! Que bonitinho, amigo robô! Mas às vezes ainda parece que ele está bem lento, não é? Na verdade, se você colocou uma velocidade menor que 5, mais ou menos, é possível que ele nem ande!

**Vídeo 01** - Robô Funcionado

Isso acontece devido ao arrasto dele no chão ser grande o bastante para que a força a qual estamos aplicando lateralmente nele não faça nem ele sair do lugar. Para resolver esse problema, podemos diminuir o atrito no chão com um material menos resistente! Afinal, se observarmos o nosso robozinho, nem perna ele tem, desliza pelo mundo afora e, por isso, merece uma superfície bem polida!

Para finalizar a nossa alteração no projeto, vamos criar um novo material físico 2D com um atrito bem baixo e adicioná-lo ao Rigidbody 2D do nosso robozinho! Também já vimos isso nesta aula, então vou deixar que vocês façam por conta própria, ok? Aposto que conseguem! Não esqueçam de criar a nova pasta para manter os Assets organizados!

Assim, finalizamos a nossa quarta aula! Foi uma aula e tanto, hein? Bastante assunto interessante e nem deu tempo de entrarmos em detalhes de todas as partes! Muita coisa boa ainda ficou como leitura complementar, então não deixem de conferir nas referências!

Com grande satisfação encerramos nossa aula de hoje, até a próxima!

# Leitura Complementar

Documentação do Motor de Física 2D: https://docs.unity3d.com/Manual/Physics2DReference.html.

Vídeos sobre o motor de Física 2D: https://unity3d.com/pt/learn/tutorials/topics/2d-game-creation/2d-physics-overview?playlist=17093.

# Resumo

Na aula de hoje, conhecemos a utilização de física nos jogos! Aprendemos que a física não existe no conjunto de pixels o qual forma a tela de um computador e vimos como podemos utilizar simulações e modelos para fazer parecer que tudo aquilo é igual ao mundo real! Conhecemos o motor de física do Unity e sua separação em 2D e 3D. Aprofundamos nossos conhecimentos em 2D e ainda nos elementos Rigidbody 2D, Colliders, Joints, Effectors e Constant Force 2D, além de aprendermos a criar materiais físicos 2D e configurar as suas propriedades!

Foi uma aula e tanto! Conseguimos avançar bastante o nosso projeto! Espero que todos tenham acompanhado sem problemas. O projeto completo está disponível neste link, e o fórum está aberto a todos que queiram contribuir com sugestões, dúvidas ou qualquer outro assunto.

# Autoavaliação

1. Como o Unity separa o seu motor de física em relação ao 2D e ao 3D?
2. Quais são os principais componentes do Motor 2D?
3. Quais os tipos diferentes que um Rigidbody 2D pode assumir?
4. Qual a utilidade de um material físico?

# Referências

Documentação oficial do Unity - Disponível em: https://docs.unity3d.com/Manual/index.html.

Tutoriais oficiais do Unity - Disponível em: https://unity3d.com/pt/learn/tutorials.

RABIN, Steve. Introdução ao Desenvolvimento de Games, Vol 2. CENGAGE.