Detectores de Colisões

Flávio Roberto Dias Silva

Encontro 5
Desenvolvimento de Jogos Eletrônicos com Unity 3D.

Universidade Estadual do Oeste do Paraná

Colisões

Uma importante e indispensável ferramenta no desenvolvimento de jogos é a detecção de colisão, em muitas situações de jogo queremos que um evento especifico aconteça exatamente no momento de uma colisão e com esse intuito a Unity disponibiliza várias funções para o tratamento de colisões.

As funções de colisão são disponibilizadas pela classe `MonoBehaviour`. Uma dessas funções é a `OnCollisionEnter`.

```
void OnCollisionEnter(Collision colisao)
{
// seu código aqui
}
```

Essa função é acionada de forma semelhante com **void** `Start()` e **void** `Update()`, isso é, ela é chamada automaticamente quando o jogo está rodando.

A função `OnCollisionEnter` é chamada quando ocorre uma colisão entre um objeto que tem `Rigidbody` e um outro objeto que tem um *colisor*, ou seja, para que ela funcione corretamente é necessários que esses componentes estejam corretamente adicionados aos objetos.

A definição da função deve estar em um script adicionado em um dos dois objetos envolvidos na colisão, o objeto com `Rigidbody` ou o objeto com *colisor*.

Assim, por exemplo, podemos ter um projetil com um script de dano que aplicará o dano quando o projetil atingir um alvo que pode tomar dano.

A aplicação do dano deve ocorrer quando o ocorrer a colisão.

Quando ocorre o dano o projetil deve desaparecer. Logo, na função de colisão devemos destruir o projetil que causou o dano.

O código a ser anexado ao projetil teria um corpo semelhante a:

```
void OnCollisionEnter(Collision colisao)
{
  // o código de causar dano aqui
  Destroy(gameObject);
}
```

O parâmetro da função que é uma variável do tipo `Collision` funciona de forma semelhante a variável do tipo `RaycatHit` que vimos na função `RayCast(...)`, essa variável do tipo `Collision` guarda informações sobre a colisão.

Entre as propriedades acessíveis pela variável do tipo `Collision` estão o número de pontos de colisão, as coordenadas desses pontos no mundo 3D, os vetores normais aos pontos de colisão, o nome dos gameObject's atingidos pela colisão, entre muitas outras.

Retornando ao nosso exemplo do projetil que causará dano. No código interior a função `OnCollisionEnter` devemos ter um **if** perguntando se o objeto atingido era um personagem que pode tomar dano ou não, para isso devemos acessar o nome, tag ou outro identificador do objeto através da variável do tipo `Collision` constante na definição da função.

Temos algumas variações da função de detecção de colisão entre rigidbody e collider, como por exemplo:

```
void OnCollisionExit(Collision colisao){}
void OnCollisionStay(Collision colisao){}
```

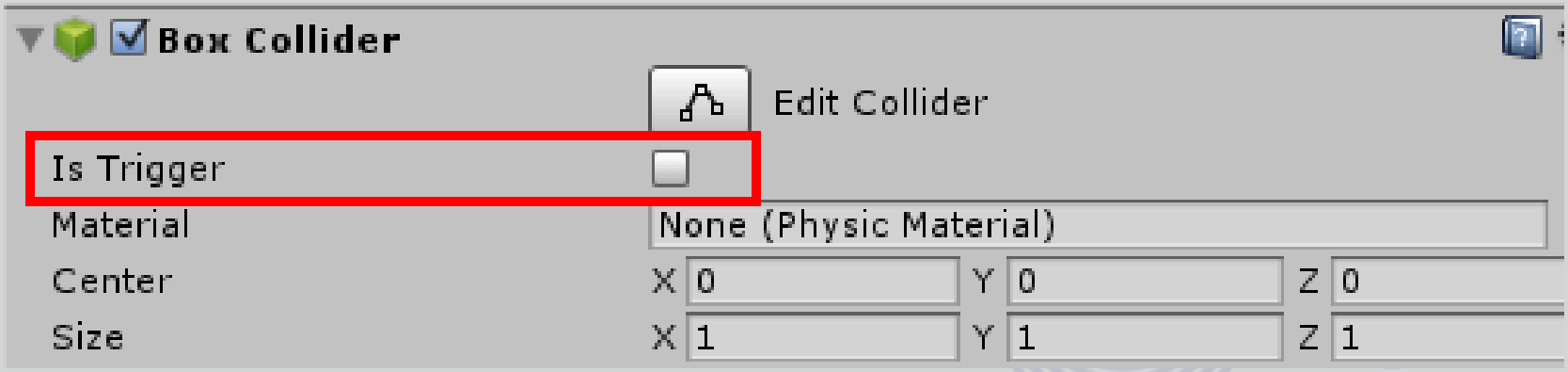
A primeira é chamada quando a colisão termina e a segunda é chamada enquanto a colisão está se mantendo.

Outra função dedicada a “detecção de colisões” é a função `OnTriggerEnter(Collider colisor)`.

Detecção de colisões está entre aspas no paragrafo anterior porque o trigger é um detector de colisões sem colisão, isso é, ele demarca uma caixa de colisão no espaço 3D a qual podemos atravessar. A função é disparada quando entramos na área demarcada pelo colisor.

Para que `OnTriggerEnter` funcione devemos ter um objeto com um colisor e outro objeto com um colisor marcado como trigger.

A opção `isTrigger` pode ser vista no Inspector de todos os colisores.



Observação: Uma das muitas utilidades do trigger é na criação de itens coletáveis. Nesses casos interessa detectar quando o personagem toca no item, mas não nos interessa que haja uma colisão entre o coletável e o personagem, não interessa que o coletável bloqueie o movimento do personagem.

Observação: A função `OnTriggerEnter` tem um funcionamento melhor quando tem um `Rigidbody` anexado ao objeto com o trigger.

A função `OnTrigerEnter` também tem suas variantes semelhantes a função `OnCollisionEnter`. Existem:

```
void OnTriggerExit(Collider colisor)
// disparada quando o objeto sai do trigger
void OnTriggerStay(Collider colisor)
// disparada enquanto o objeto está em contato com
//o trigger
```

Como vimos a função `OnCollisionEnter` e suas variantes funcionam quando um objeto tem rigidbody e outro tem Collider. Para o caso de uma movimentação com `CharacterController` a função

```
void OnControllerColliderHit(ControllerColliderHit
hit)
{
//seu código aqui
}
```

faz um papel semelhante.

A função `OnControllerColliderHit` é chamada quando há uma colisão entre um objeto que tem `CharacterController` e um outro objeto que tem `Collider`, no entando, só é chamada quando a função `Move()` do `CharacterController` estiver sendo chamada.

Isso causa alguns inconvenientes já que se o controlador de personagens estiver parado e um objeto com colisor tocar nele `OnControllerColliderHit` não será chamada. É preciso estar atento a isso e preparado pra contornar esses problemas caso apareçam no decorrer do programar do jogo.

Exercício: Retornar a cena do último encontro com a bala que acertava objetos e era destruída e destruir alvos.

Desafio: Criar comportamentos de inimigo para outros objetos 3D. Criar um script de vida do personagem. Fazer com que os inimigos “morram”, sejam destruidos ao serem atingidos pelos projeteis ou tomem dano e sejam destruidos quando os pontos de vida chegarem a zero.

[1] Barnes, D. J., Kölling, M.(2009), Programação Orientada a Objetos com Java. Uma introdução prática usando BLUEJ, 4ªed., Pearson Prentice Hall.

[2] Battaiola, A. L. (2000). Jogos por Computador ? Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação In: XIX Jornada de Atualização em Informática. Curitiba: SBC, Julho/2000, v. 2. pp. 83 - 122

[3] Battaiola, A. L.; Elias, N. C.; Domingues, R.G. et al (2002). Desenvolvimento de um Software Educacional com Base em Conceitos de Jogos de Computador In: XIII Simpósio Brasileiro de Informática na Educação. São Leopoldo: SBC, 2002, pp. 282-290.

[4] Crua, E. W. G.; Bittencourte, J. R.(2005) Desenvolvimento de Jogos 3D: Concepção, Design e Programação. Anais da XXIV Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, pp.1313-1356, São Leopoldo, Brasil, Julho de 2005. Deitel, H. M., Deitel, P. J.(2010), Java: Como programar, 8ªed., Pearson Prentice Hall. Rio de Janeiro IMPA.

[5] Freeman, E., Freeman, E.(2007), Use a Cabeça Padrões de Projetos, 2ªed., Rio de Janeiro Altabooks.

[6] Sintes, A.(2002), Aprenda Programação Orientada a Objetos em 21 dias, São Paulo Makron Books.

[7] Stellman, A.; Greene, J.(2011), Use a Cabeça! *C#*, Rio de Janeiro, AltaBooks.

[8] Unity Tecnologies (2016)(A). Unity 3D User Manual [online]. Disponivel em: [http://docs.unity3d.com/Manual/index.html][Acesso em 18/04/2016]

[9] Unity Tecnologies (2016)(B). Unity 3D Community Forum [online]. Disponivel em: [http://forum.unity3d.com/][Acesso em 18/04/2016]

[10] Unity Tecnologies (2016)(C). Unity 3D Online Tutorials [online]. Disponivel em: [https://unity3d.com/pt/learn/tutorials][Acesso em 18/04/2016]

[11] Unity Tecnologies (2016)(D). Unity 3D Community Wiki [online]. Disponivel em:

[http://wiki.unity3d.com/index.php/Main_Page][Acesso em 18/04/2016]