Aula 14 – Físicas e Colisões na Unity

Bem-vindo à Aula 14 do nosso Curso de Desenvolvimento de Jogos 3D! Hoje, vamos mergulhar em um dos pilares mais fascinantes e essenciais da criação de jogos: a física e a detecção de colisões. Se você já se perguntou como os objetos em um jogo caem, batem uns nos outros ou interagem de forma realista, esta aula é para você. Entender esses conceitos não é apenas uma questão técnica, mas a chave para dar vida e credibilidade aos seus mundos virtuais.

Imagine um jogo onde os personagens flutuam sem peso, as bolas atravessam paredes ou os carros não reagem a impactos. Seria uma experiência frustrante e irreal, certo? É exatamente por isso que dominar as físicas e colisões na Unity é crucial. Ao final desta aula, você será capaz de aplicar o componente Rigidbody para simular comportamentos físicos, utilizar diferentes tipos de Colliders para definir as formas de interação, distinguir entre Triggers e Collisions, e implementar eventos de colisão para criar lógicas de jogo dinâmicas e responsivas.

Nossa jornada começará com o coração da simulação física, o Rigidbody, explorando como massa, gravidade e forças moldam o movimento dos objetos. Em seguida, desvendaremos os Colliders, as "formas invisíveis" que permitem que os objetos interajam. Por fim, vamos diferenciar Triggers de Collisions e aprender a usar os eventos de colisão, como OnCollisionEnter e OnTriggerEnter, para que seus jogos reajam de forma inteligente e envolvente. Prepare-se para dar um salto quântico na interatividade dos seus projetos!

O Coração da Interatividade: O Componente Rigidbody

Em qualquer jogo 3D que se preze, a sensação de peso, movimento e impacto é o que nos conecta à realidade virtual. Sem ela, os objetos parecem flutuar ou se mover de forma robótica, quebrando a imersão. É aqui que entra o componente Rigidbody na Unity, a peça central que transforma um objeto estático em um elemento dinâmico, sujeito às leis da física. Ele é o motor que permite que seus objetos caiam, sejam empurrados, girem e reajam a forças externas, como a gravidade ou uma explosão.

Pense no Rigidbody como a "alma física" de um objeto no seu jogo. Assim como nós, seres vivos, temos massa e somos afetados pela gravidade, um GameObject com um Rigidbody ganha essas propriedades. Ele não apenas permite que o objeto seja movido por forças, mas também o torna capaz de colidir e interagir com outros objetos de forma realista, calculando a resposta a cada impacto. Sem um Rigidbody, um objeto é apenas uma representação visual; com ele, ele se torna um participante ativo no mundo do jogo.

> 💡 **Ponto-Chave**
>
> **Rigidbody = Alma Física**
>
> Transforma objetos estáticos em elementos dinâmicos que respondem à gravidade, forças e colisões.

Ao adicionar um Rigidbody a um GameObject, você está essencialmente dizendo à Unity: "Este objeto deve ser controlado pelo motor de física". Isso significa que ele passará a ter propriedades como massa, arrasto (drag), arrasto angular (angular drag) e, crucialmente, será afetado pela gravidade. É a partir dessas propriedades que podemos simular desde uma simples queda de uma maçã até a complexa dinâmica de um veículo em alta velocidade, abrindo um leque de possibilidades para a interatividade e o realismo do seu jogo.

Rigidbody em Detalhes: Massa, Gravidade e Forças

Massa

Determina o "peso" do objeto e como ele reage a forças. Maior massa = mais difícil de mover.

Gravidade

Força invisível que puxa tudo para baixo. Pode ser desativada ou personalizada para efeitos especiais.

Forças

AddForce() e AddTorque() aplicam impulsos lineares ou rotacionais para mover e girar objetos.

A massa é uma das propriedades mais fundamentais de um Rigidbody, determinando o quão "pesado" um objeto é e como ele reage a forças. Um objeto com maior massa será mais difícil de mover ou acelerar, e terá um impacto maior sobre objetos mais leves. Imagine empurrar uma caixa de papelão vazia versus uma caixa cheia de pedras; a diferença na massa é o que dita a resistência e a inércia. Na Unity, você pode ajustar a massa para simular diferentes materiais e pesos, influenciando diretamente a jogabilidade.

A gravidade, por sua vez, é a força invisível que puxa tudo para baixo. Por padrão, a Unity aplica uma força gravitacional a todos os Rigidbodies, simulando a gravidade terrestre. No entanto, você tem controle sobre isso: pode desativar a gravidade para objetos que flutuam ou criar efeitos de gravidade personalizados para ambientes espaciais ou mundos de fantasia. Essa flexibilidade permite criar desde jogos de plataforma realistas até experiências onde a gravidade é um elemento de quebra-cabeça.

Para mover ou influenciar um Rigidbody, utilizamos forças. Métodos como AddForce() e AddTorque() permitem aplicar impulsos lineares ou rotacionais, respectivamente. AddForce() pode ser usado para empurrar um objeto, fazê-lo pular ou até mesmo simular a propulsão de um foguete. Já AddTorque() é ideal para fazer objetos girarem, como uma roda ou um projétil com efeito. Entender como e quando aplicar essas forças é essencial para criar interações dinâmicas e responsivas no seu jogo.

Moldando a Interação: Compreendendo os Colliders

O que são Colliders?

Se o Rigidbody é a alma física de um objeto, os Colliders são a sua "forma física invisível". Eles são os componentes que definem os limites de um objeto para fins de detecção de colisão. Sem um Collider, mesmo que um objeto tenha um Rigidbody, ele simplesmente passaria por outros objetos como um fantasma, pois o motor de física não teria uma "superfície" para calcular o impacto. Eles são essenciais para que seus personagens não atravessem paredes e para que os projéteis atinjam seus alvos.

Pense nos Colliders como campos de força invisíveis que envolvem seus GameObjects. Quando dois desses campos de força se encontram, o motor de física da Unity detecta essa interação. A beleza dos Colliders é que eles não precisam ser idênticos à forma visual do seu objeto. Na verdade, para otimização de performance, é comum usar formas geométricas mais simples (como caixas ou esferas) para representar objetos complexos, desde que a aproximação seja boa o suficiente para a jogabilidade.

> ⚡ **Otimização**
>
> **Simplicidade = Performance**
>
> Use formas geométricas simples sempre que possível. Colliders complexos exigem mais recursos computacionais.

A escolha do Collider certo é um equilíbrio entre precisão e performance. Colliders mais simples, como Box ou Sphere, são computacionalmente mais baratos e rápidos de processar. Colliders mais complexos, como o Mesh Collider, oferecem maior fidelidade à forma visual, mas exigem mais recursos. Entender as características de cada tipo de Collider e quando usá-los é fundamental para criar jogos que não apenas pareçam bons, mas que também rodem de forma fluida e eficiente em diversas plataformas.

Formas Comuns de Collider: Box, Sphere e Capsule

Os Colliders mais básicos e eficientes na Unity são o Box Collider, o Sphere Collider e o Capsule Collider. Cada um tem sua aplicação ideal, dependendo da forma do objeto que você deseja representar e da precisão necessária para a interação. Dominar o uso desses três tipos é o primeiro passo para criar colisões eficazes e otimizadas.

1

Box Collider

Perfeito para objetos retangulares ou quadrados, como paredes, pisos, caixas, mesas ou edifícios. Sua simplicidade o torna extremamente eficiente para cálculos de colisão.

- Ideal para: Paredes, pisos, caixas
- Performance: Excelente
- Precisão: Boa para formas retas

2

Sphere Collider

Escolha natural para objetos redondos, como bolas, moedas, ou até mesmo para representar a área de efeito de uma explosão. Sua forma uniforme simplifica os cálculos de colisão.

- Ideal para: Bolas, moedas, efeitos circulares
- Performance: Excelente
- Precisão: Perfeita para formas redondas

3

Capsule Collider

Uma cápsula, forma cilíndrica com extremidades arredondadas. Amplamente utilizado para personagens, pois se adapta bem à forma humana, permitindo movimento suave por cantos e escadas.

- Ideal para: Personagens, pilares
- Performance: Muito boa
- Precisão: Ótima para formas humanoides

O Collider Avançado: Mesh Collider

Nem todos os objetos em um jogo têm formas simples que podem ser representadas por caixas, esferas ou cápsulas. Pense em uma árvore com galhos complexos, uma estátua detalhada ou um terreno irregular. Para esses casos, a Unity oferece o **Mesh Collider**, uma ferramenta poderosa que permite que a forma de colisão de um objeto seja exatamente a mesma de sua malha 3D (o modelo visual).

O Mesh Collider é a opção mais precisa, pois ele usa a geometria exata do modelo 3D para detectar colisões. Isso significa que, se você tem um objeto com muitos detalhes e precisa que as colisões sejam extremamente fiéis à sua forma visual, o Mesh Collider é a escolha certa. No entanto, essa precisão vem com um custo: ele é significativamente mais intensivo em termos de processamento do que os Colliders primitivos. Usar muitos Mesh Colliders complexos pode impactar negativamente a performance do seu jogo.

> ⚠️ **Atenção**
>
> **Propriedade "Convex"**
>
> Quando marcado como Convex, o Mesh Collider se torna mais eficiente e pode interagir com outros Mesh Colliders. Use para objetos dinâmicos.

Para mitigar o impacto na performance, é crucial entender a propriedade "Convex" do Mesh Collider. Quando marcado como Convex, o Mesh Collider se comporta como uma "casca" convexa do objeto, o que o torna mais eficiente e permite que ele interaja com outros Mesh Colliders. Sem essa propriedade, um Mesh Collider só pode ser usado para colisões estáticas (objetos que não se movem) ou para detectar colisões com Colliders primitivos. Use-o com sabedoria, priorizando-o para objetos estáticos e complexos, e sempre que possível, simplifique a malha de colisão.

A Dança da Interação: Triggers vs. Collisions

Quando dois objetos se encontram no mundo do jogo, a Unity pode interpretar essa interação de duas maneiras principais: como uma **Colisão** ou como um **Trigger**. Embora ambos detectem o contato entre Colliders, a forma como eles reagem e o propósito de cada um são fundamentalmente diferentes. Compreender essa distinção é crucial para criar lógicas de jogo que vão além do simples "bater e quicar".

Colisão

Uma **Colisão** é o que você esperaria de uma interação física no mundo real. Quando dois Colliders colidem, eles se impedem mutuamente de atravessar um ao outro, e o motor de física calcula a resposta, como ricochetear, empurrar ou parar. Para que uma colisão física ocorra, pelo menos um dos objetos envolvidos deve ter um componente Rigidbody, e ambos devem ter Colliders. É o cenário ideal para simular impactos, empurrões e qualquer interação que envolva uma barreira física.

Trigger

Um **Trigger**, por outro lado, é uma forma de detecção de contato que não envolve uma resposta física. Quando um Collider é configurado como Trigger (marcando a opção "Is Trigger" no Inspector), ele ainda detecta quando outro Collider entra em seu espaço, mas não impede o movimento do outro objeto. Em vez disso, ele "dispara" um evento, permitindo que você execute uma lógica de jogo específica. Pense em um Trigger como um sensor invisível que detecta a presença de algo, sem bloqueá-lo.

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
Colisão	Interação física, bloqueio de movimento	Requer Rigidbody em pelo menos um objeto	Carro batendo em uma parede, bola quicando no chão
Trigger	Detecção de contato sem bloqueio físico	Collider com "Is Trigger" ativado	Personagem coletando uma moeda, entrando em uma área de efeito

Triggers em Ação: Detectando Interações sem Bloqueio

Os Triggers são ferramentas incrivelmente versáteis para adicionar interatividade aos seus jogos sem a complexidade dos cálculos físicos de colisão. Eles permitem que você crie áreas sensíveis no seu ambiente que reagem quando um jogador ou outro objeto as atravessa, sem impedir seu movimento. Essa funcionalidade é a base para muitos elementos de jogabilidade que vemos em praticamente todos os jogos.

🔧 Como Configurar

Para configurar um Collider como Trigger, basta selecionar o GameObject no Inspector da Unity e marcar a caixa **"Is Trigger"** no componente Collider. Lembre-se que pelo menos um dos objetos envolvidos deve ter um Rigidbody.

Aplicações Práticas dos Triggers

Zonas de Coleta

O jogador pega itens como moedas ou power-ups simplesmente passando por eles.

Áreas de Ativação

Portas que se abrem automaticamente ou cutscenes que iniciam quando o jogador entra em uma região.

Zonas de Perigo

Detectar se um personagem caiu de um penhasco (zona de "game over") ou entrou em área de lava.

Efeitos Contínuos

Aplicar dano gradual em uma área de lava ou cura em uma zona de recuperação.

Collisions em Ação: Interações Físicas e Respostas

Quando usar Collisions?

- Objetos que devem bloquear movimento
- Simulação de impactos realistas
- Cálculo de forças e respostas físicas
- Interações que envolvem massa e inércia

Enquanto os Triggers detectam a presença, as Collisions são sobre o impacto e a resposta física. Elas são a espinha dorsal de qualquer jogo que simule um mundo com objetos sólidos que interagem de forma realista. Quando você quer que um objeto bata em outro, empurre-o, ou quebre-o, você está lidando com colisões.

Condições para uma Colisão Física

01

Ambos os GameObjects devem ter Colliders

Sem Colliders, não há superfície para detectar o contato.

02

Pelo menos um deve ter Rigidbody

O Rigidbody permite que o motor de física calcule a interação.

03

Colliders não podem estar marcados como Trigger

Se "Is Trigger" estiver ativado, será um Trigger, não uma Collision.

As colisões são fundamentais para a credibilidade do seu jogo. Elas permitem que você simule a física de um carro batendo em uma parede, uma bola quicando no chão, ou um personagem empurrando uma caixa. Além de impedir a passagem, as colisões geram informações detalhadas sobre o impacto, como a força da colisão, os pontos de contato e a velocidade relativa dos objetos. Essas informações são cruciais para criar respostas de jogo complexas, como aplicar dano, tocar sons de impacto ou gerar partículas.

Lidando com Eventos: OnCollisionEnter e OnTriggerEnter

Detectar que uma colisão ou um trigger ocorreu é apenas metade da batalha. A outra metade é fazer com que seu jogo reaja a esses eventos. É aqui que entram os métodos de callback da Unity: **OnCollisionEnter** e **OnTriggerEnter**. Eles são como "alarmes" que disparam quando uma interação acontece, permitindo que você execute um código específico em resposta.

Cenário 1: Coleta de Moeda

Quando o personagem toca uma moeda, você quer que a moeda desapareça e a pontuação do jogador aumente.

Solução: Use OnTriggerEnter

Cenário 2: Projétil vs. Inimigo

Quando um projétil atinge um inimigo, você quer que o inimigo sofra dano e o projétil seja destruído.

Solução: Use OnCollisionEnter

Cenário 3: Zona de Perigo

Quando o personagem entra em uma área de lava, você quer aplicar dano contínuo.

Solução: Use OnTriggerEnter + OnTriggerStay

Esses métodos são chamados automaticamente pela Unity quando as condições de colisão ou trigger são satisfeitas. Para utilizá-los, você precisa criar um script (um componente C#) e anexá-lo a um dos GameObjects envolvidos na interação. Dentro desse script, você define as funções OnCollisionEnter ou OnTriggerEnter (e suas variações Stay e Exit), e a Unity se encarregará de chamá-las no momento certo, passando informações sobre o outro objeto envolvido na interação.

OnCollisionEnter na Prática: Reagindo a Impactos

O método **OnCollisionEnter** é invocado no primeiro frame em que dois Colliders se tocam e pelo menos um deles possui um Rigidbody (não marcado como Trigger). Ele é o seu ponto de partida para lidar com impactos físicos e suas consequências. Quando essa função é chamada, a Unity passa um objeto do tipo Collision como parâmetro, que contém uma riqueza de informações sobre o evento.

Informações Disponíveis no Collision Object

gameObject

O objeto com o qual você colidiu

contacts

Lista de pontos de contato da colisão

relativeVelocity

Velocidade relativa dos objetos no momento do impacto

impulse

Força do impacto calculada

Por exemplo, se você tem um jogo de tiro, ao detectar uma colisão com OnCollisionEnter, você pode verificar a tag do objeto colidido. Se for um inimigo, você pode aplicar dano a ele e tocar um som de impacto. Se for uma parede, você pode gerar partículas de faísca e tocar um som de ricochete. A capacidade de acessar esses detalhes permite que você crie uma lógica de jogo sofisticada e imersiva.

Exemplo de Código

```
void OnCollisionEnter(Collision collision) {
 // Verifica se o objeto colidido tem a tag "Inimigo"
 if (collision.gameObject.CompareTag("Inimigo")) {
 Debug.Log("Inimigo atingido!");
 // Exemplo: Aplicar dano ao inimigo
 // collision.gameObject.GetComponent().ReceberDano(10);
 }
 else if (collision.gameObject.CompareTag("Parede")) {
 Debug.Log("Colisão com a parede!");
 // Exemplo: Tocar som de impacto ou gerar partículas
 }
}
```

OnTriggerEnter na Prática: Ativando Eventos

Similar ao OnCollisionEnter, o método **OnTriggerEnter** é chamado quando um Collider (marcado como Trigger) detecta que outro Collider entrou em seu espaço. A principal diferença é que, como um Trigger não causa uma colisão física, o parâmetro passado para a função é um objeto do tipo Collider (representando o Collider do outro objeto), e não um Collision object.

Essa simplicidade no parâmetro reflete a natureza do Trigger: ele se preocupa mais com "quem entrou" do que com os detalhes físicos do impacto. Com o objeto Collider recebido, você pode acessar o gameObject do outro objeto, sua tag, name, e outros componentes anexados a ele. Isso é tudo o que você precisa para a maioria dos casos de uso de Triggers.

> 💡 **Dica**
>
> **Triggers são perfeitos para:**
>
> - Sistemas de coleta
> - Zonas de ativação
> - Detecção de proximidade
> - Áreas de efeito

Aplicações Práticas

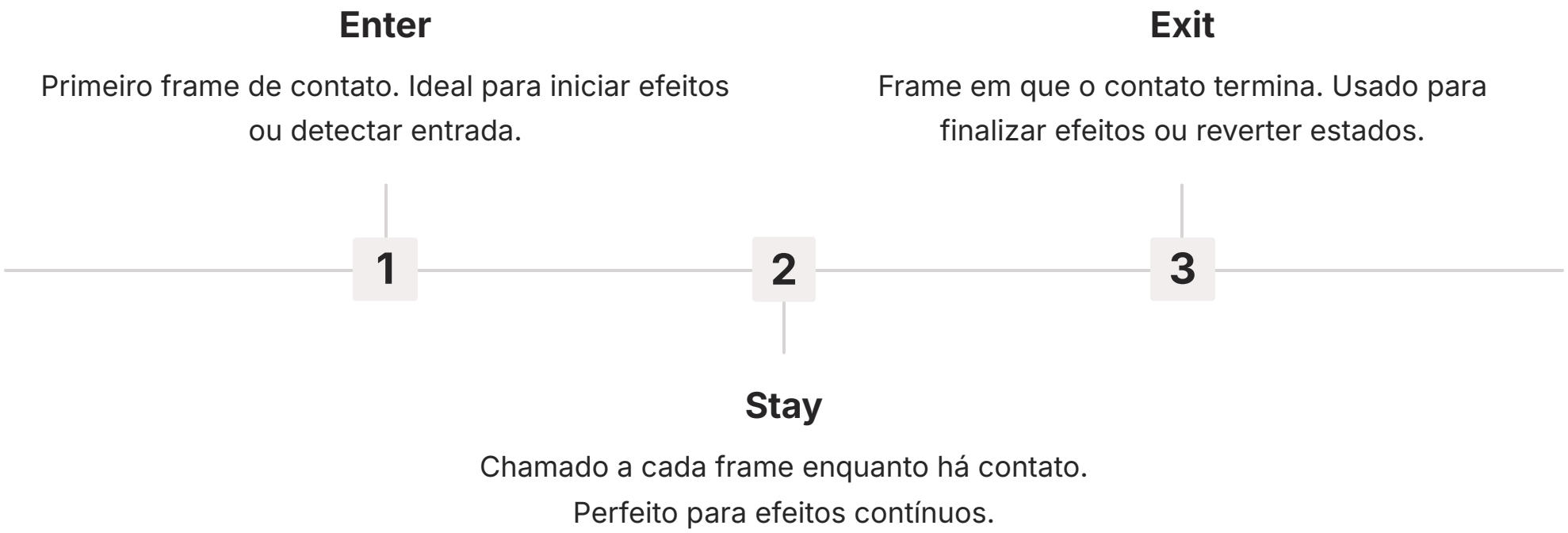
As aplicações de OnTriggerEnter são vastas e fundamentais para a interatividade do jogo. Imagine um jogo de coleta: quando o jogador entra no Trigger de uma moeda, você pode desativar a moeda (fazê-la "desaparecer"), adicionar pontos à pontuação do jogador e tocar um som de coleta. Em um jogo de aventura, entrar em um Trigger pode iniciar um diálogo com um NPC ou ativar uma missão. A flexibilidade de OnTriggerEnter o torna uma ferramenta poderosa para criar eventos baseados em proximidade.

Exemplo de Código

```
void OnTriggerEnter(Collider other) {
 // Verifica se o objeto que entrou no Trigger tem a tag "Moeda"
 if (other.CompareTag("Moeda")) {
 Debug.Log("Moeda coletada!");
 // Exemplo: Desativar a moeda e aumentar a pontuação
 other.gameObject.SetActive(false);
 // GameManager.Instance.AdicionarPontos(10);
 }
 else if (other.CompareTag("ZonaPerigo")) {
 Debug.Log("Entrou na zona de perigo!");
 // Exemplo: Aplicar dano contínuo ou iniciar um efeito
 }
}
```

Além do "Enter": Eventos de Permanência e Saída

As interações em um jogo nem sempre são instantâneas. Às vezes, um objeto pode permanecer em contato com outro por um período, ou sair de uma área de interação. Para cobrir esses cenários, a Unity oferece variações dos eventos Enter: **OnCollisionStay**, **OnCollisionExit**, **OnTriggerStay** e **OnTriggerExit**. Esses métodos permitem que você crie lógicas de jogo mais dinâmicas e responsivas a interações contínuas.



Eventos de Permanência (Stay)

OnCollisionStay e OnTriggerStay são chamados a cada frame enquanto os Colliders permanecem em contato (seja por colisão física ou por trigger). Isso é extremamente útil para situações onde você precisa de um efeito contínuo. Por exemplo, se um personagem está em uma área de lava (um Trigger), OnTriggerStay pode ser usado para aplicar dano gradualmente a cada segundo. Ou, se um objeto está sendo empurrado contra uma parede, OnCollisionStay pode ser usado para aplicar uma força constante.

Eventos de Saída (Exit)

Por outro lado, OnCollisionExit e OnTriggerExit são chamados no frame em que os Colliders param de se tocar ou saem da área do Trigger. Esses eventos são perfeitos para finalizar efeitos ou reverter estados. Por exemplo, quando um personagem sai da área de lava, OnTriggerExit pode parar o dano contínuo. Ou, quando um objeto para de colidir com uma superfície, OnCollisionExit pode parar um som de atrito. Juntos, esses eventos fornecem um controle completo sobre o ciclo de vida das interações no seu jogo.

Otimizando Físicas e Colisões: Performance é Chave

> ⚡ **Performance**
>
> **Lembre-se:** Um jogo otimizado não é apenas mais rápido, mas também mais divertido e acessível para um público maior.

No desenvolvimento de jogos, a performance é sempre uma preocupação. Físicas e colisões, embora essenciais, podem ser computacionalmente intensivas se não forem gerenciadas corretamente. Um jogo com muitas colisões complexas ou cálculos de física desnecessários pode rapidamente sofrer de quedas de frame rate, resultando em uma experiência de jogo ruim. Por isso, otimizar esses sistemas é uma prática fundamental para qualquer desenvolvedor.

Estratégias de Otimização

1 Use Colliders Primitivos

Prefira Box, Sphere e Capsule Colliders sempre que possível. Eles são muito mais leves que Mesh Colliders para o motor de física processar.

2 Mesh Colliders Apenas para Estáticos

Para objetos complexos que não se movem, um Mesh Collider estático é aceitável. Evite Mesh Colliders convexos em objetos dinâmicos, a menos que seja estritamente necessário.

3 Use Camadas de Colisão

Configure Collision Layers para controlar quais objetos podem colidir entre si, reduzindo cálculos desnecessários. Por exemplo, projéteis de inimigos não precisam colidir com outros inimigos.

4 Ajuste as Configurações de Física

Modificar a frequência de atualização da física no projeto pode ter um grande impacto na performance, especialmente em dispositivos móveis.

Pense no motor de física como uma equipe de engenheiros trabalhando para calcular todas as interações. Quanto mais complexas as formas e mais objetos para calcular, mais trabalho eles terão. Uma das principais estratégias de otimização é usar Colliders primitivos (Box, Sphere, Capsule) sempre que possível, em vez de Mesh Colliders. Eles são muito mais leves para o motor de física processar. Para objetos complexos que não se movem, um Mesh Collider estático é aceitável, mas evite Mesh Colliders convexos em objetos dinâmicos, a menos que seja estritamente necessário.

Consolidação e Próximos Passos

Chegamos ao fim de nossa jornada pelas físicas e colisões na Unity, um universo que, como vimos, é o alicerce para a interatividade e o realismo em qualquer jogo 3D. Exploramos o Rigidbody como o coração da simulação física, dando massa e movimento aos seus objetos. Desvendamos os Colliders, as "formas invisíveis" que definem os limites de interação, desde os eficientes Box, Sphere e Capsule até o preciso, mas custoso, Mesh Collider.

Compreendemos a distinção crucial entre Triggers e Collisions, aprendendo a usar cada um para diferentes propósitos: Triggers para detecção de eventos sem bloqueio físico e Collisions para interações com resposta física. E, finalmente, dominamos os eventos de callback como OnCollisionEnter e OnTriggerEnter, bem como suas variações Stay e Exit, para que seus jogos possam reagir de forma inteligente e dinâmica a cada interação. A capacidade de otimizar esses sistemas garante que seus jogos não apenas funcionem, mas rodem de forma fluida e eficiente.

🎯 Em Prática

Agora você tem as ferramentas para fazer objetos caírem, baterem, rolarem e interagirem de maneiras convincentes. Use Colliders simples para a maioria dos objetos, reserve Mesh Colliders para cenários específicos e utilize Triggers para coletáveis e zonas de efeito. Implemente os eventos de colisão para criar lógicas de jogo que respondam a cada toque e impacto, transformando seu mundo estático em um ambiente vibrante e interativo.

Autoavaliação

Teste seus conhecimentos

01

Qual componente é essencial para que um GameObject seja afetado pela gravidade e possa reagir a forças físicas na Unity?

a) Transform
b) Collider
c) Rigidbody
d) Mesh Renderer

02

Para qual tipo de objeto o Box Collider é mais adequado, visando otimização e precisão?

a) Uma bola de futebol
b) Um personagem humano
c) Uma parede retangular
d) Uma árvore com galhos complexos

03

Qual a principal diferença entre um Trigger e uma Colisão na Unity?

a) Triggers são para objetos estáticos, Collisions para objetos dinâmicos.
b) Triggers detectam contato sem impedir o movimento, Collisions impedem o movimento.
c) Collisions usam Colliders, Triggers não.
d) Triggers são mais lentos que Collisions.

04

Qual método de callback é invocado quando um Collider (marcado como Trigger) detecta que outro Collider entrou em seu espaço, sem causar uma interação física?

a) OnCollisionEnter
b) OnCollisionStay
c) OnTriggerEnter
d) OnTriggerExit

05

Explique a importância da propriedade "Convex" em um Mesh Collider e quando ela deve ser utilizada.

Gabarito

1. c) Rigidbody

2. c) Uma parede retangular

3. b) Triggers detectam contato sem impedir o movimento, Collisions impedem o movimento.

4. c) OnTriggerEnter

Próxima Aula

Na **Aula 15 – Trabalhando com Materiais e Iluminação**, exploraremos como dar vida visual aos seus modelos 3D, aplicando texturas e materiais que definem sua aparência, e como a iluminação pode transformar completamente a atmosfera e a percepção do seu jogo.

Recursos Adicionais

- **Documentação Oficial da Unity sobre Physics**: Para aprofundar nos detalhes técnicos e nas configurações avançadas do motor de física.
- **Tutoriais de Colisão no YouTube (canal Unity Learn)**: Para ver exemplos práticos e visuais da implementação de colisões e triggers.
- **Asset Store da Unity (Physics Assets)**: Para explorar soluções prontas e ferramentas que podem otimizar ou expandir as capacidades físicas do seu projeto.