04

Colisão

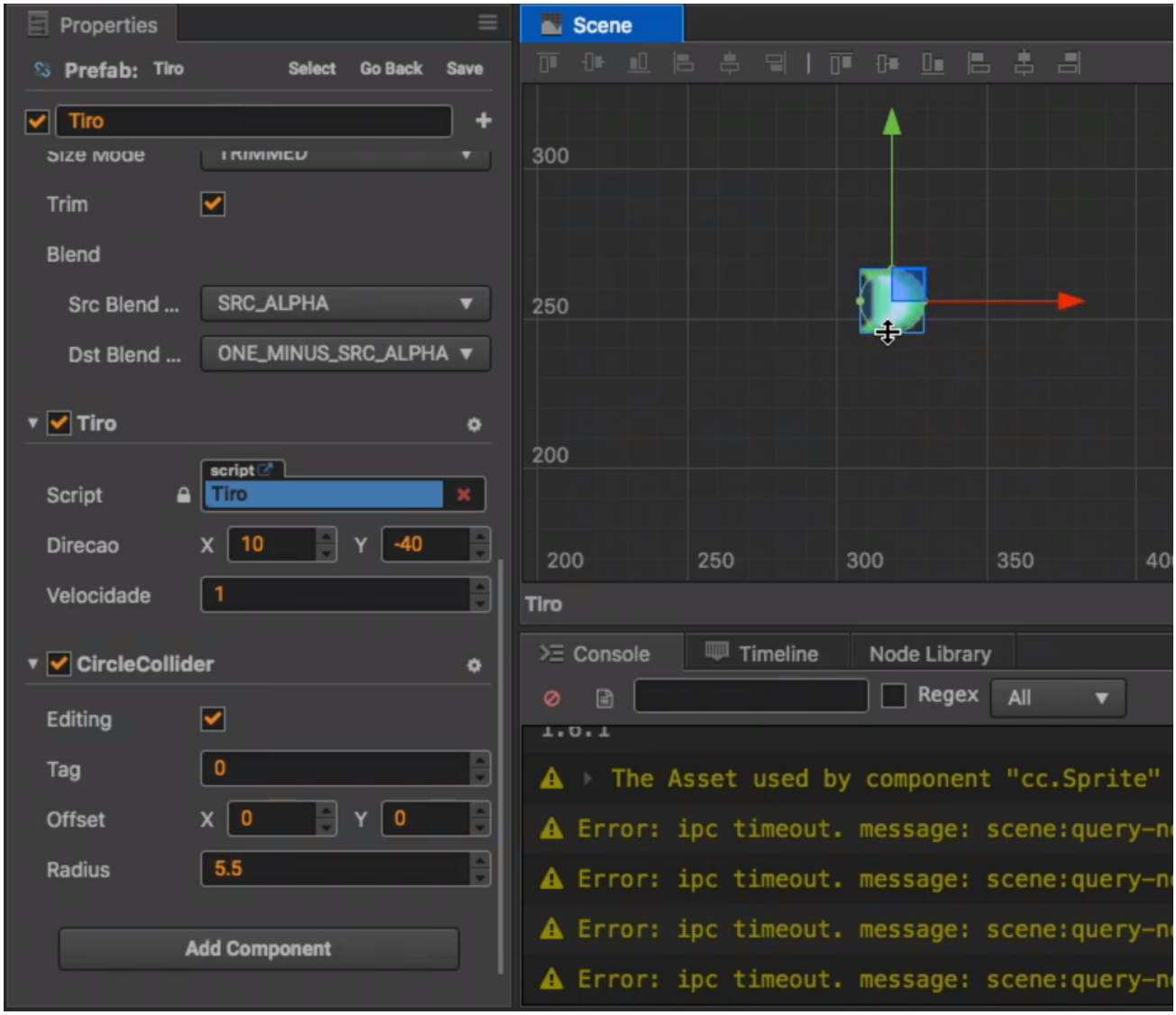
Transcrição

Talvez você já tenha notado, mas os tiros apesar de serem disparados como esperado, não colidem como deveriam. Se atirarmos na nave inimiga que está em nosso *canvas*, nada acontece.

O problema é que não adicionamos nenhum componente de colisão no tiro e nem mesmo na nave inimiga. Sabemos que um objeto da nossa cena é composto por vários componentes e agora veremos como adicionar um componente de colisão ao nosso objeto `Tiro`. Com este objeto selecionado, na aba `Properties` da Cocos, vamos descer toda a barra de rolagem da aba até encontrar o botão `Add Component`.

Precisamos selecionar a categoria do componente. Neste caso será de `Collider` com a opção `Add Collider Component` e por último o tipo de `Collider` que será usado. O tipo se refere a um formato, sendo assim, o `Box Collider` é uma área de colisão quadrada, um pouco estranho para um tiro. Escolheremos o `Circle Collider` que é circular.

Note que o objeto `Tiro` agora aparece com um círculo ao seu redor. O círculo parece ser um pouco maior do que o tiro em si, para ajustar isso, na aba `Properties` do objeto, no componente `CircleCollider` marcaremos a opção `Editing`, isso fará com que controles de ajuste apareçam no círculo de colisão no *canvas*, podemos aumentar ou diminuir a área de colisão. O ideal é que seja bem próximo dos limites da imagem do tiro.



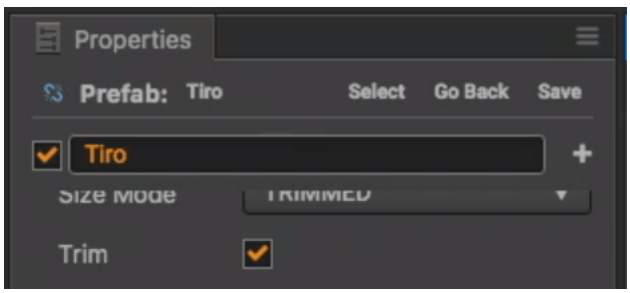
Caso tenha dificuldade em ver o tamanho da área de colisão ou do tiro, é possível aumentar o zoom do *canvas* usando os controles de rolagem do mouse.

Atualizando um Prefab

Nós adicionamos o componente de colisão ao tiro, mas sendo mais específico, adicionamos ao objeto que já está na cena. Isso quer dizer que o *Prefab* que cria os demais tiros não possuem esse componente em seu conjunto de instruções.

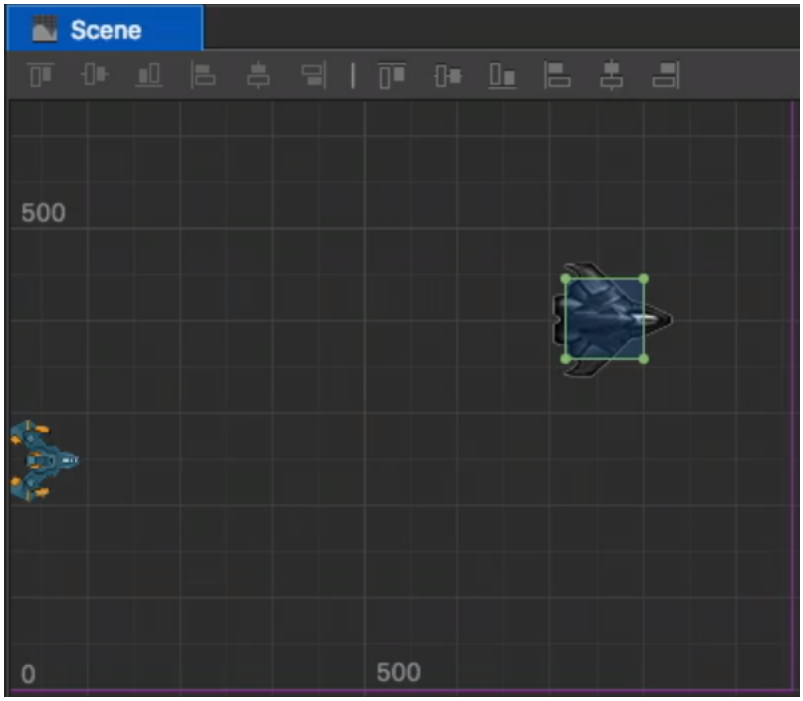
Para atualizar esse conjunto de instruções, na aba *properties* do objeto `Tiro` apertamos o botão `Save`.

Você verá que há também um botão chamado `Go Back` vizinho ao botão `Save`. Este botão serve para atualizar um objeto que já esteja na cena, mas ainda não está com as últimas atualizações do seu *Prefab*.



Como nosso código já instancia o tiro quando clicamos no *canvas* e o *prefab* do mesmo já encontra-se atualizado, podemos selecionar o objeto `Tiro` na aba `Node Tree` e com o clique direito, selecionar a opção `delete` para removê-lo da cena. Ele já é criado dinamicamente.

Também precisamos adicionar um componente de colisão para a nave inimiga. Dessa fez será do tipo `Box Collider` já que a nave não é exatamente um círculo. Também podemos editar a área de colisão para considerar apenas o centro da nave. Não tem problema se o bico e as asas da nave ficarem um pouco de fora.



Habilitando as colisões

Até podemos testar, mas não veremos nenhum resultado real no que fizemos até aqui. O problema agora é que por padrão o sistema de colisões da Cocos vem desabilitado. Isso por que este tipo de ação consome um pouco mais de recursos processamento da máquina.

Por enquanto, nosso tiro é quem precisa colidir e quem atira é a nossa nave. Por este motivo, vamos ativar o sistema de colisões da Cocos na classe do `Jogador.js` . No método `onLoad` adicionamos:

```
cc.director.getCollisionManager().enabled = true;
```

Com o trecho acima estamos acessando o objeto `director` da Cocos que gerencia vários recursos e dele estamos acessando o gerenciador de colisões onde alteramos a propriedade `enabled` para `true` . Assim teremos o sistema de colisões ativo.

Tratando as colisões

Com o sistema de colisões ativo, precisamos executar mais um passo para termos algum *feedback* de que realmente tudo funciona como esperado: Codificar o que acontece quando houver uma colisão.

Quando o objeto colide, a Cocos aciona um evento dentro do objeto. O método se chama `onCollisionEnter` que recebe dois parâmetros. O primeiro deles é o outro objeto com qual se colidiu e o segundo é uma referência ao próprio objeto. Para fiz de teste, podemos adicionar o este na classe do *script* `Tiro.js` da seguinte forma:

```
onCollisionEnter: function(outro, eu){
    console.log("colidiu");
}
```

Teste com a console do navegador aberta. Ao atingir a nave inimiga com o tiro, a mensagem `colidiu` deve ser impressa.

O próximo passo é fazer com que os dois objetos sumam. Isso por que o tiro atingiu seu alvo e a nave foi destruída. O componente `Node` de todos os objetos da cena além de posicionarem o objeto, podem remover o objeto da cena, para isso usamos seu método `destroy`. Como o método `onCollisionEnter` recebe referências aos dois objetos que colidiram, precisamos apenas chamar este método em cada um deles.

```
onCollisionEnter: function(outro, eu){
    console.log("colidiu");
    outro.node.destroy();
    eu.node.destroy();
}
```

Agora ao atingir o inimigo, tanto o tiro quanto o inimigo devem sumir da cena.