02

instanciação

Transcrição

Já temos o comportamento do tiro definido, mas o comportamento de atirar ainda não. Este pertence à nave do jogador. A classe no `Jogador.js` atualmente se encontra assim:

```
cc.Class({
    extends: cc.Component,

    properties: {
        _acelerando: false,
        _direcao: cc.Vec2,
    },

       // use this for initialization
    onLoad: function () {
        cc.SystemEvent.on(cc.SystemEvent.EventType.KEY_DOWN, this.teclaPressionada, this);
        cc.SystemEvent.on(cc.SystemEvent.EventType.KEY_UP, this.teclaSolta, this);

        let canvas = cc.find("Canvas");
        canvas.on("mousemove", this.mudarDirecao, this);
    },

    mudarDirecao: function(event){
        let posicaoMouse = event.getLocation();
        posicaoMouse = new cc.Vec2(posicaoMouse.x, posicaoMouse.y);

        let direcao = posicaoMouse.sub(this.node.position);
        direcao = direcao.normalize();

        this._direcao = direcao;
    }

    teclaPressionada: function(event){
        if(event.keyCode == cc.KEY.a){
            this._acelerando = true;
        }
    },

    teclaSolta: function(event){
        if(event.keyCode == cc.KEY.a){
            this._acelerando = false;
        }
    },

    // called every frame, uncomment this function to activate update callback
       update: function (dt) {
        if(this._acelerando){
            this.node.position = this.node.position.add(this._direcao);
        }
    },
});
```

Como o comportamento de atirar depende do clique do mouse dentro do `Canvas`, registraremos mais uma captura de evento para o mouse sendo este para o evento `mousedown`.

```
// use this for initialization
onLoad: function () {
    cc.SystemEvent.on(cc.SystemEvent.EventType.KEY_DOWN, this.teclaPressionada, this);
    cc.SystemEvent.on(cc.SystemEvent.EventType.KEY_UP, this.teclaSolta, this);

    let canvas = cc.find("Canvas");
    canvas.on("mousemove", this.mudarDirecao, this);

    canvas.on("mousedown", this.atirar, this);
},
```

O método `atirar` será o responsável por realizar o comportamento. Vamos codificá-lo!

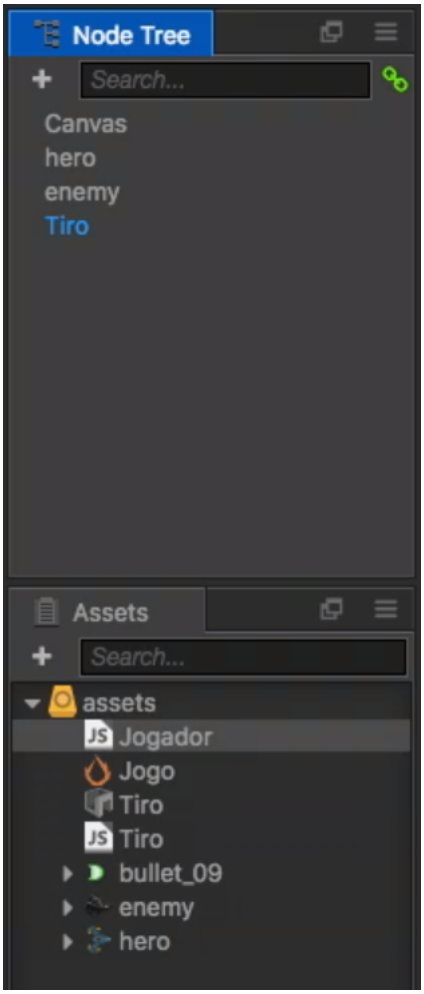
```
atirar: function(event){

},
```

Conhecendo os Prefabs

Nossa nave precisa criar várias instâncias de tiro para serem disparados conforme os cliques vão sendo feitos. Então precisamos de um conjunto de instruções que criam um tiro. Esse conjunto de instruções é chamado de **Prefab**. A criação de um *prefab* é muito fácil, basta clicar no objeto que desejamos criar o *prefab* na aba `Node Tree` e arrasta-lo para a aba `Assets`.

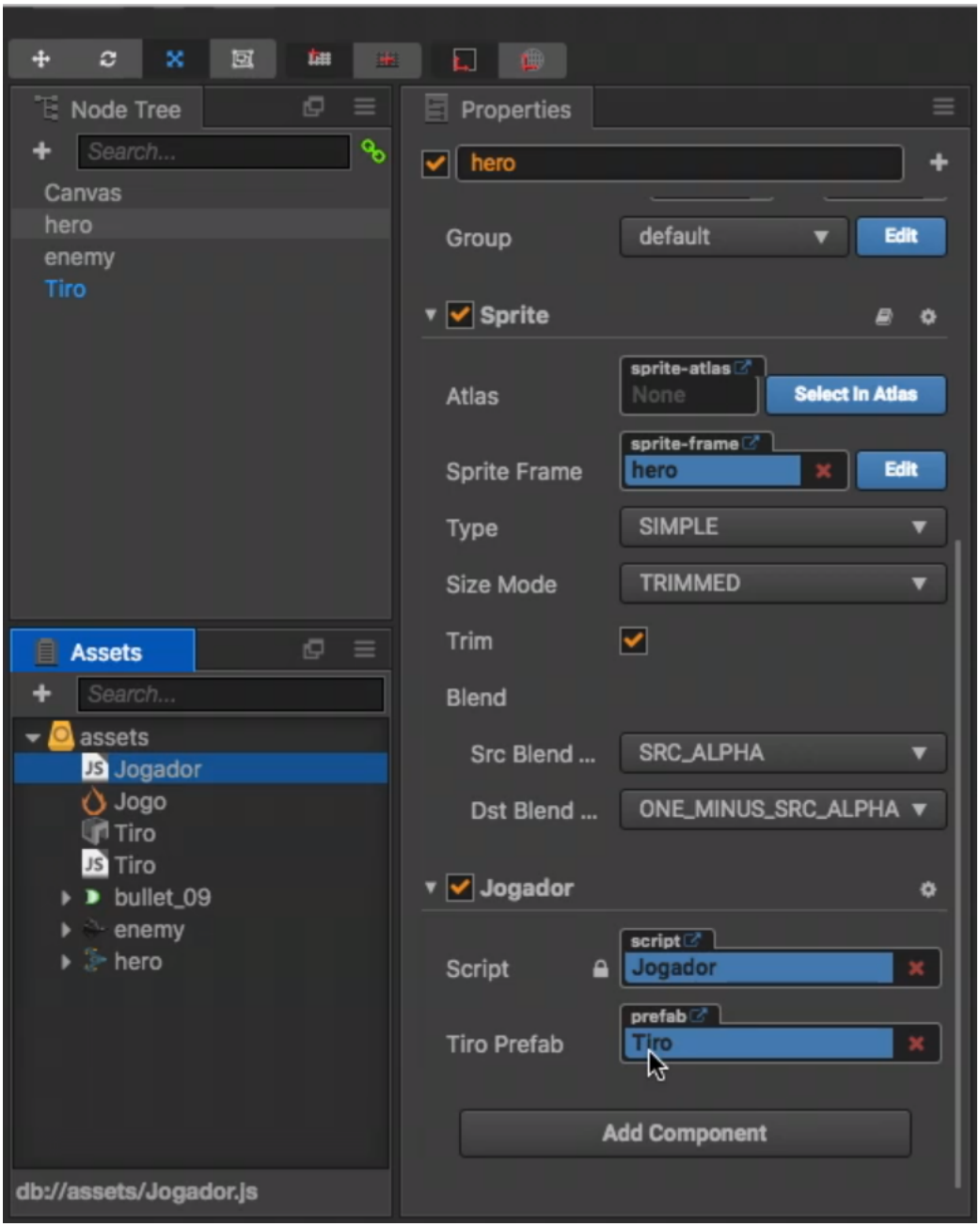
Ao criar o *prefab* do objeto tiro, veremos que o objeto `Tiro` na aba `Node Tree` da Cocos terá seu nome destacado em azul, e um novo objeto com um ícone de caixa cinza aparecerá na aba `Assets`, com o mesmo nome do objeto.



Também precisamos criar uma propriedade do tipo `Prefab` na classe do `Jogador.js`.

```
properties: {
    _acelerando: false,
    _direcao: cc.Vec2,
    tiroPrefab: cc.Prefab,
},
```

O último passo é associar o *asset* do *prefab* do tiro com a propriedade `tiroPrefab` da classe em `Jogador.js`. Faremos isso arrastando o *asset prefab* para a propriedade `Tiro Prefab` na aba `Properties` do objeto `hero`.



Lembre-se de selecionar o objeto `hero` antes de arrastar o *tiro prefab* para a propriedade correta.

Instanciando o tiro

Temos o *prefab* do tiro configurado, mas ainda não estamos criando instâncias desse objeto. Quem faz isso é o método `instantiate` da Cocos. No método `atirar` teremos:

```
atirar: function(event){
    let tiro = cc.instantiate(this.tiroPrefab);
},
```

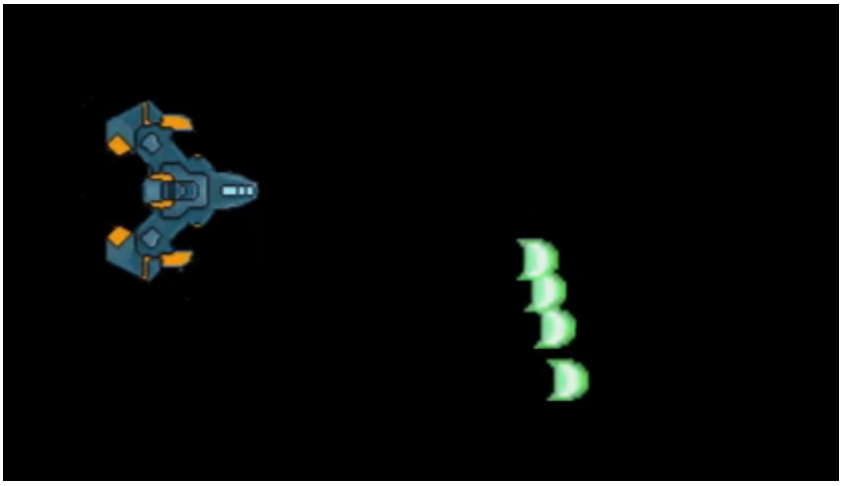
Dessa forma, criamos um objeto `Node` na variável `tiro` , ou seja, uma instância do nosso *prefab*, mas do jeito que está, a Cocos cria o objeto mas não sabe o que fazer com ele. Ele nem aparecerá na tela. Não estamos informando isso para a Cocos.

Para desenhar um objeto na tela, a Cocos precisa saber onde. E isso podemos informar configurando quem é o objeto pai deste objeto atribuindo um valor para a propriedade `parent` deste objeto. A Cocos, sabendo quem é o objeto pai,

desenhará o objeto "filho" na tela. Como os tiros serão disparados em relação à nave, configuraremos como pai do `tiro` o mesmo pai do objeto `Node` da nave. Dessa forma teremos:

```
atirar: function(event){
    let tiro = cc.instantiate(this.tiroPrefab);
    tiro.parent = this.node.parent;
},
```

Se testarmos agora, veremos que a cada clique, um novo tiro será desenhado na tela. E estes tiros seguirão as instruções de direção e velocidade que já codificamos anteriormente.



Reposicionando e direcionando o tiro

As instâncias dos tiros já funcionam, mas elas possuem um problema comum. Independente da posição da nave, eles surgem sempre no mesmo lugar. Isso por que não estamos indicando de forma alguma a posição inicial do objeto `tiro`. Como ele precisa ser igual ao posicionamento da nave, fica muito simples de resolver. Basta atribuirmos a propriedade `position` do objeto `Node` da nave ao `position` do objeto `tiro` que é um objeto `Node`.

```
atirar: function(event){
    let tiro = cc.instantiate(this.tiroPrefab);
    tiro.parent = this.node.parent;
    tiro.position = this.node.position;
},
```

Isso já corrige o posicionamento inicial do tipo, mas não a direção. Lembra que o objeto `Tiro` possui um componente chamado `Tiro`? (*neste caso o componente tiro referido é a classe dentro do script `Tiro.js` associada ao objeto*). Dentro deste componente há uma propriedade pública chamada `direcao`.

Considerando que a direção do tiro é em relação ao mouse e que o direcionamento da nave também segue esse mesmo padrão. Podemos atribuir a direção da nave diretamente para o tiro. O problema é que não podemos acessar o componente `Tiro` diretamente. Por isso usaremos o método `getComponent` no objeto `tiro` para buscar o componente `Tiro` e assim fazer a atribuição da direção corretamente.

```
atirar: function(event){
    let tiro = cc.instantiate(this.tiroPrefab);
    tiro.parent = this.node.parent;
    tiro.position = this.node.position;
```

```
        let componenteTiro = tiro.getComponent("Tiro");
        componenteTiro.direcao = this._direcao;
    },
```

Agora nossos tiros além de serem instanciados corretamente, seguem uma direção e têm uma velocidade com o posicionamento inicial que já considera a posição da nave.