03

Comportamento dos inimigos

Transcrição

A cada passo nosso jogo fica ainda mais interessante. Já podemos nos mover, atirar e temos inimigos aparecendo aleatoriamente dentro do universo do jogo, mas estes inimigos ainda estão estáticos. Eles não se movem, aparecem e ficam ali parados.

A ideia é que , além de se moverem sozinhos, eles também atirem contra nós. De certa forma já codificamos estes dois comportamentos antes para o jogador. Ele se move e também atira.

Vamos iniciar essa nova etapa já criando o *script* `Inimigo.js` na aba `Assets` e definindo algumas de suas propriedades. Uma delas é o alvo, ou seja, quem a nave inimiga irá perseguir. A direção e a velocidade.

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

    properties: {
      _alvo: cc.Node,
      _direcao: cc.Vec2,
      velocidade: 50,
    },

    // use this for initialization
    onLoad: function () {
    },

    // called every frame, uncomment this function to activate update callback
    update: function (dt) {
    },
});
```

Já sabemos que o alvo a ser seguido é a própria nave do jogador. E também já sabemos como encontrar objetos na cena com o método `find` . Considerando isso, no método `onLoad` já faremos essa relação para que depois possamos calcular a direção em que a nave inimiga deve se mover.

```
onLoad: function () {
    this._alvo = cc.find("hero");
},
```

Um detalhe importante que precisamos considerar agora é relacionado ao cálculo da direção da nave inimiga. Ela não considera eventos de mouse nem nada do tipo. Tudo que precisamos fazer é subtrair da posição do alvo, a nossa posição, assim já sabemos para que direção a nave deve seguir. Lembrando que precisamos normalizar esse vetor para que a velocidade seja constante e não dependa das distâncias. Assim criaremos o método `mudarDirecao` .

```
mudarDirecao: function(){
    let direcao = this._alvo.position.sub(this.node.position);
    direcao = direcao.normalize();
```

```
    this._direcao = direcao;
},
```

Agora só precisamos fazer com que no método `update` tenhamos a mudança de direção sendo executada junto ao cálculo de deslocamento e reposicionamento da nave considerando o tempo e não o *frame rate*.

```
update: function (dt) {
    this.mudarDirecao();
    let deslocamento = this._direcao.mul(this.velocidade * dt);
    this.node.position = this.node.position.add(deslocamento);
},
```

Observação: Lembre-se de associar o script `Inimigo.js` ao objeto inimigo da cena e atualizar o *prefab* deste objeto.

Fazendo os inimigos atirarem

Já temos o comportamento de locomoção dos inimigos em nossa direção funcionando. Porém, eles ainda não atiram. Cuidaremos dessa parte agora. Já trabalhamos com tiros antes e por isso temos o *prefab* do mesmo pronto. Precisamos dele como uma propriedade do objeto `enemy`. Outra coisa que precisamos é de um intervalo de disparo. A nave inimiga não terá controles, por isso agendaremos os disparos a cada segundo com o método `schedule`. Primeiro criamos as propriedades.

```
properties: {
    _alvo: cc.Node,
    _direcao: cc.Vec2,
    velocidade: 50,

    tiroPrefab: cc.Prefab,
    tempoAtaque: 1,
},
```

O método que fará os disparos da nave inimiga na nossa nave se chamará `atirar`, então no método `onLoad`, já deixaremos o agendamento dos disparos codificado.

```
onLoad: function () {
    this._alvo = cc.find("hero");

    this.schedule(this.atirar, this.tempoAtaque);
},
```

Feito isso, criaremos o método `atirar` que fará as seguintes tarefas: Criar uma instância do tiro, posicionar o tiro na mesma posição da nave inimiga na cena e indicar quem é o pai deste novo objeto e por fim, indicará ao objeto `Tiro` dentro dessa instância a direção para onde ele deve seguir. Neste caso a direção será a mesma direção da nave.

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

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

Observação: Antes de testar, lembre-se de associar o *prefab* de tiro ao objeto da nave inimiga e salvar o *prefab* da mesma.

Agora teremos um comportamento muito estranho. A cada dois segundos a nave aparece, se move, e um segundo depois ela some. O que está acontecendo? Veremos a seguir.