05

Inputs de teclado

Transcrição

Nossa nave já se movimenta, mas este movimento ainda é bem limitado. A nave segue infinitamente para a direita e não é isso que queremos. A ideia é que no jogo real, a nave se movimente apenas quando pressionada a tecla `A` do teclado. E que siga a direção do mouse. Primeiro faremos o movimento considerar o teclado, depois trabalharemos a questão do mouse.

Precisamos informar para a Cocos que usaremos o teclado. Este "aviso", não deve ser feito no método `update` por que estaríamos avisando em todo frame que o teclado será utilizado. Imagine por exemplo, uma pessoa cutucando você a cada 1 segundo te perguntando as horas, seria bem chato, certo? *não queremos ser chatos com a Cocos*.

Precisamos informar apenas uma vez para a Cocos que usaremos o teclado. Por isso utilizaremos o método `onLoad` que, diferente do método `update` , é executado apenas uma vez, logo após a Cocos carregar todos os objetos da cena, ou seja, carrega todos os objetos e depois executa o método `onLoad` de cada um deles.

No método `onLoad` então, adicionaremos o seguinte código:

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

O método `on` da propriedade `SystemEvent` da Cocos (`cc`) é o responsável por capturar eventos e fazer o *bind* com os métodos de *callback* que deverão responder a estes eventos. Neste caso, estamos dizendo que quando o evento de `KEY_DOWN` (tecla pressionada) acontecer, um método chamado `teclaPressionada` deve ser executado. O `this` final serve para que a Cocos saiba de qual objeto o método deve ser executado, caso não, teremos problemas com a dinamicidade do `this` no JavaScript.

Resumindo, o método `on` precisa de 3 argumentos: O tipo de evento a ser capturado, o *callback* que será executado e qual o *contexto* de execução do *callback*. Como o contexto é o próprio objeto (`this`), criaremos o método `teclaPressionada` na mesma classe do script `Jogador.js` logo abaixo do método `onLoad` .

```
teclaPressionada: function(event){

},
```

Como o método deve responder a um evento, recebemos o mesmo como parâmetro. Com base neste evento, precisamos saber se a tecla pressionada foi a da letra `a` . Se imprimirmos o objeto `event` na *console*, teríamos algumas informações relevantes, como por exemplo o número da tecla pressionada, também conhecida como `keyCode` . Caso já tenha feito algum código JavaScript que lidasse com eventos de teclado, saberá do que se trata.

O mais legal é que não precisamos comparar a propriedade `keyCode` do objeto `event` com um *número mágico* sem muito significado em nosso código. A Cocos fornece uma série de constantes que ajudam bastante nestes casos. Vejamos o código a seguir:

```
teclaPressionada: function(event){

    if(event.keyCode == cc.KEY.a){
        console.log("a");
    }

},
```

A propriedade `KEY` disponibilizada pela Cocos nos permite acessar o código da tecla de forma mais clara. Assim, podemos iniciar o jogo novamente e abrir o console do navegador. Ao pressionar a tecla `a` teremos a letra sendo impressa na console, caso outra tecla seja pressionada, nada acontece.

Acelerando e parando a nave

Agora que conseguimos capturar a tecla pressionada e verificar que é a tecla `a`, faremos com que a nave se movimente apenas enquanto a tecla estiver pressionada. Para isso, inicialmente criaremos uma variável *"privada"* dentro da chave `properties` do nosso componente. Esta variável se chamará `acelerando` e será do tipo *booleano*, inicialmente configurado como `false`.

```
properties: {
    // foo: {
    //    default: null,      // The default value will be used only when the component attachi
    //                           to a node for the first time
    //    url: cc.Texture2D,  // optional, default is typeof default
    //    serializable: true, // optional, default is true
    //    visible: true,      // optional, default is true
    //    displayName: 'Foo', // optional
    //    readonly: false,    // optional, default is false
    // },
    // ...
    _acelerando: false,
},
```

A propriedade `acelerando` com seu valor `false` indica que a nave deve estar parada. Para fazer com que a nave ande, alteraremos este valor para `true` quando o evento de tecla pressionada for disparado e a letra da tecla seja a letra `a`. Já temos essa verificação pronta no método `teclaPressionada` então precisamos apenas trocar o valor de `acelerando`.

```
teclaPressionada: function(event){

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

},
```

Para que tudo funcione precisamos alterar apenas mais um trecho do nosso código. O método `update` continua movendo nossa nave para a direita assim que o jogo é iniciado. Adicionaremos uma verificação antes do movimento de fato acontecer. Precisamos verificar se `acelerando` é `true`, assim movimentamos a nave de acordo com os controles que estamos criando. O código que estava assim:

```
update: function (dt) {
    this.node.x += 1;
},
```

Passa a ficar assim:

```
update: function (dt) {
    if(this._acelerando){
        this.node.x += 1;
    }
},
```

Se `acelerando` for `true`, movemos a nave. Agora tudo funciona, podemos testar! Só há um problema: em momento algum indicamos uma forma da nave parar, ou seja, depois de pressionar a tecla `a` e mantê-la pressionada, a nave anda, mas ao soltar a tecla, a nave continua se movendo.

Vamos resolver esse problema fazendo exatamente o código inverso do que já fizemos até o momento. Primeiro, registraremos a captura do evento de `KEY_UP` (*tecla solta*) informando que ao soltar a tecla, o método `teclaSolta` deverá ser executado. Importante lembrar de fazer o registro de captura dos eventos apenas no `onLoad`.

```
// 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);
},
```

Com o evento sendo capturado, criaremos o método `teclaSolta` que fará a mesma verificação do método `teclaPressionada` invertendo apenas o valor da propriedade `acelerando`.

```
teclaSolta: function(event){

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

},
```

Note que não é apenas receber o evento no caso da tecla solta. Precisamos verificar também qual tecla foi solta. Com este passo, temos o movimento da nave pronto. Agora ela anda ao pressionar a tecla `a` e para assim que a tecla for solta.

O que aprendemos aqui

Até aqui vimos como capturar os eventos de teclado e como a Cocos ajuda no trabalho de verificar quais teclas foram pressionadas. Aprendemos também que a captura dos eventos, por boa prática, devem ser registrados no método `onLoad` do componente.

Por último vimos como associar os eventos ao componente em si, caso tal tecla seja pressionada, algo acontece, caso seja solta, agora acontece. O código completo da classe no *script* `Jogador.js` se encontra assim:

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

    properties: {
        // foo: {
        //    default: null,      // The default value will be used only when the component atta
        //                           to a node for the first time
        //    url: cc.Texture2D,  // optional, default is typeof default
        //    serializable: true, // optional, default is true
        //    visible: true,      // optional, default is true
        //    displayName: 'Foo', // optional
        //    readonly: false,    // optional, default is false
        // },
        // ...
        _acelerando: false,
    },

    // 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);
    },

    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.x += 1;
        }
    },
});
```