01

# O código do Jogo

**Transcrição**

*Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/carreira-js/stages/05-gallows.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.*

Apesar de a carreira na qual este projeto faz parte não abordar ES2015 (ES6) podemos modificar nosso código para que use alguns dos recursos desta nova versão. Alura possui cursos de ES6 que podem agregar ainda mais no seu cabedal de conhecimento. Aqui, faremos um preview do que você pode aprender conosco.

## let e const

A partir do ECMASCRIT 2015 (ES6) foi introduzida duas novas sintaxes para substituir declaração de variáveis com `var`, são elas **let** e **const**. Todavia, quais problemas elas vêm resolver?

Vejamos um simples exemplo:

```
for(var i = 1; i <= 10; i++) {
    console.log(i);
}
console.log(i); // qual será o valor de i?
```

No exemplo anterior, qual será o valor da última instrução? Se você respondeu 11 acertou. Em linguagens como C ou Java `i` não seria acessível fora do bloco do `for`, diferente em JavaScript, pois variáveis declaradas com `var` possuem escopo de função e não de bloco, mesmo quando declaradas dentro de um bloco.

Outro ponto a destacar de `var` é que variáveis declaradas com elas podem ser redeclaradas sem que o interpretador reclame (a não ser que o `"use strict"`; seja adicionado no script ativando um modo mais criterioso). Vejamos:

```
var nome = 'cangaceiro';
nome = 'Flávio';
var nome = 'Almeida'; // é aceito!
```

Podemos resolver esses dois problemas que acabamos de ver declarando as variáveis com `let`:

```
for(let i = 1; i <= 10; i++) {
    console.log(i);
}
console.log(i); // erro, não é acessível fora do bloco fort
```

```
let nome = 'cangaceiro';
nome = 'Flávio';
let nome = 'Almeida'; // erro, não pode declarar novamente uma variável que já foi declarada
```

Excelente, e onde entra o `const` nessa história? Quando usado, ele não permite que seja mais utilizado o perador `=` para atriburi um novo valor a variável. Vejamos:

```
const ID = 10;
ID = 100; // erro, não pode atribuir um novo valor
```

Como não podemos usar o operador `=` mais de uma vez, somos obrigados a atribuir o valor da variável, ou seja, somos obrigado a declará-la e inicializá-la ou teremos um erro.

Então, dentro do que acabamos de ver, podemos usar `let` e `const` em diversos pontos do nosso código. Vejamos:

```
const criaJogo = function(sprite) {

    let etapa = 1;
    let lacunas = [];
    let palavraSecreta = '';

    const criaLacunas =  function() {

        for (let i = 0; i < palavraSecreta.length; i++) {
            lacunas.push('');
        }
    };

    const proximaEtapa = function() {
        etapa = 2;
    };

    const setPalavraSecreta = function (palavra) {

        palavraSecreta = palavra;
        criaLacunas();
        proximaEtapa();
    };

    const getLacunas = function () {

        return lacunas;
    };

    const getEtapa = function () {

        return etapa;
    }

    const processaChute = function(chute) {
        // atenção, precisou separar const e var, porque resultado não é inicializada
        const exp = new RegExp(chute, 'gi');
        var resultado
            ,acertou = false;

        while (resultado = exp.exec(palavraSecreta))
            acertou = lacunas[resultado.index] = chute;
```

```
        if (!acertou) sprite.nextFrame();
    };

    const ganhou = function() {

        return !lacunas.some(function(lacuna) {
            return lacuna == ''
        });
    };

    const perdeu = function () {

        return sprite.isFinished();
    };

    const ganhouOuPerdeu = function () {

        return ganhou() || perdeu();
    };

    const reinicia = function () {

        etapa = 1;
        palavraSecreta = '';
        lacunas = [];
        sprite.reset();
    };

    return {
        setPalavraSecreta: setPalavraSecreta,
        getLacunas: getLacunas,
        getEtapa: getEtapa,
        processaChute: processaChute,
        ganhou: ganhou,
        perdeu: perdeu,
        ganhouOuPerdeu: ganhouOuPerdeu,
        reinicia: reinicia
    };
};
```

Faz sentido nossas declarações de funções serem declaradas com `const` porque não queremos que alguém acidentalmente atribua um novo valor para elas. Já a variável `current` que precisa ser incrementada precisa ser declarada com `let` , isto porque tanto o pós e pré incrementos são um atalho para o uso do operador `=` .

Agora, podemos ver outro recurso do ES2015 (ES6) que pode nos ajudar a escrever um código mais enxuto.

## Arrow function

Excelente, mas podemos lançar mão das arrow functions que inicialmente é uma forma mais sucinta de declarar funções, mas nos cursos avançados de JavaScript aqui da Alura que abordam esse recurso você verá que há algumas diferenças. Todavia, para nosso programa, entender que são uma forma sucinta já é suficiente.

Veja como uma de nossas funções declaradas com arrow function (se você não declarou as funções como expressão, não poderá usar arrow function, ou seja, não há function declaration com arrow function, apenas function expression):

```
const perdeu = () => {

    return sprite.isFinished();
};
```

Agora, a `processaChute` :

```
const processaChute = (chute) => {

    const exp = new RegExp(chute, 'gi');
    var resultado
        ,acertou = false;

    while (resultado = exp.exec(palavraSecreta))
        acertou = lacunas[resultado.index] = chute;

    if (!acertou) sprite.nextFrame();
};
```

Quando uma arrow function possui apenas um parâmetro, podemos omitir os parânteses:

```
const processaChute = chute => {

    const exp = new RegExp(chute, 'gi');
    var resultado
        ,acertou = false;

    while (resultado = exp.exec(palavraSecreta))
        acertou = lacunas[resultado.index] = chute;

    if (!acertou) sprite.nextFrame();
};
```

Uma outra característica é quando o bloco de uma arrow function possui apenas uma instrução, podemos remover seu bloco e colocar a instrução diretamente sem a necessidade de utilizar return que já é considerado implicitamente nesta forma abreviada de declaração. Vejamos como perdeu() ficará:

```
const perdeu = () => sprite.isFinished();
```

Nosso código com arrow function fica assim:

```
const criaJogo = sprite => {

    let etapa = 1;
    let lacunas = [];
    let palavraSecreta = '';

    const criaLacunas = () => {

        for (let i = 0; i < palavraSecreta.length; i++) {
```

```
            lacunas.push('');
        }
    };

    const proximaEtapa = () => etapa = 2;

    const setPalavraSecreta = palavra => {

        palavraSecreta = palavra;
        criaLacunas();
        proximaEtapa();
    };

    const getLacunas = () => lacunas;

    const getEtapa = () => etapa;

    const processaChute = chute => {

        const exp = new RegExp(chute, 'gi');
        var resultado
            ,acertou = false;

        while (resultado = exp.exec(palavraSecreta))
            acertou = lacunas[resultado.index] = chute;

        if (!acertou) sprite.nextFrame();
    };

    const ganhou = () => !lacunas.some(lacuna => lacuna == '');

    const perdeu = () => sprite.isFinished();

    const ganhouOuPerdeu = () => ganhou() || perdeu();

    const reinicia = () => {

        etapa = 1;
        palavraSecreta = '';
        lacunas = [];
        sprite.reset();
    };

    return {

        setPalavraSecreta: setPalavraSecreta,
        getLacunas: getLacunas,
        getEtapa: getEtapa,
        processaChute: processaChute,
        ganhou: ganhou,
        perdeu: perdeu,
        ganhouOuPerdeu: ganhouOuPerdeu,
        reinicia: reinicia
    };
};
```

## Atalho para declaração de objetos literais

Vamos lançar nosso olhar para a última instrução de `criaJogo()`, aquela que retorna um objeto com algumas propriedades:

```
return {
    setPalavraSecreta: setPalavraSecreta,
    getLacunas: getLacunas,
    getEtapa: getEtapa,
    processaChute: processaChute,
    ganhou: ganhou,
    perdeu: perdeu,
    ganhouOuPerdeu: ganhouOuPerdeu,
    reinicia: reinicia
};
```

Quando declaramos um objeto com a sintaxe {} estamos declarando-o na forma literal. ES2015 (ES6) criou um atalho que pode ser usado toda vez que o nome da propriedade é igual ao nome da variável que será atribuída a propriedade. Nesse caso, basta declararmos a propriedade no objeto:

```
return {
    setPalavraSecreta,
    getLacunas,
    getEtapa,
    processaChute,
    ganhou,
    perdeu,
    ganhouOuPerdeu,
    reinicia
};
```

Muito mais enxuto não?