09

Organizando nosso código

Quando terminamos de criar nosso controle de animação em aula acabamos deixando nosso código com muitos `ifs` encadeados e caminhos possíveis para nosso código. Da maneira que deixamos nosso código acabamos tendo dificuldade de entender o que o método `mudarAnimação` está fazendo.

Veja o código final e tente ver quanto tempo você demora pra decifrar o que está acontecendo.

```
mudaAnimacao: function (direcao, estado) {
    let angulo = Math.atan2(direcao.y, direcao.x);
    let cosseno = Math.cos(angulo);
    cosseno = Math.abs(cosseno);
    let seno = Math.sin(angulo);
    seno = Math.abs(seno);
    if (cosseno >= this._cossenoMinimo) {
        if (direcao.x > 0) {
            proximaAnimacao += "Direira";
        } else if (direcao.x < 0) {
            proximaAnimacao += "Esquerda";
        }
    }
    if (seno>= this._senoMinimo) {
        if (direcao.y > 0) {
            proximaAnimacao += "Cima";
        } else if (direcao.y < 0) {
            proximaAnimacao += "Baixo";
        }
    }

     if (!this._animacao.getAnimationState(proximaAnimacao).isPlaying) {
            this._animacao.play(proximaAnimacao);
        }
},
```

Claro que depois de ter visto as aulas você já passou por esse código e provavelmente entende com alguma clareza o que está acontecendo. Mas podemos melhorar esse código.

O primeiro ponto que podemos atacar aqui quanto a repetição de código. Veja que o código que temos para a verificação do eixo X e do eixo Y tem uma forma comum. Nesses dois pontos do código estamos vendo se o valor de uma variável é maior ou menor do que 0 e estamos concatenando uma string à nossa variável **proximaAnimação**

```
if (direcao.x > 0) {
    proximaAnimacao += "Direita";
} else if (direcao.x < 0) {
    proximaAnimacao += "Esquerda";
}

...

if (direcao.y > 0) {
    proximaAnimacao += "Cima";
```

```
} else if (direcao.y < 0) {
    proximaAnimacao += "Baixo";
}
```

Apesar de não estarmos repetindo o código exatamente nos dois pontos, temos uma estrutura de decisão idêntica e isso nós podemos abstrair para um outro método. Como o que estamos repetindo é a estrutura de decisão, podemos passar as informações que queremos via parâmetros para esse novo método que vamos chamar de `verificarEixo`

```
verificarEixo: function (valorEixo, sentidoPositivo, sentidoNegativo) {
    if (valorEixo > 0) {
        return sentidoPositivo;
    } else if (valorEixo < 0) {
        return sentidoNegativo;
    }
},
```

Esse método tem a estrutura de decisão que estamos utilizando nos dois pontos e ele apenas recebe os valores por parâmetro e retorna o valor correto para nós. Com isso podemos utilizar o retorno desse método para concatenar os valores corretos na nossa **proximaAnimação**.

Com essa pequena alteração nosso código já ficou mais claro e ainda evitamos a repetição dessa estrutura de decisão.

Outro ponto que precisamos arrumar é o final do nosso método afinal, o que a linha `this._animacao.getAnimationState(proximaAnimacao).isPlaying` quer dizer?

Nesse ponto estamos querendo ver se a animação escolhida para executarmos já está tocando. Só que estamos buscando isso através de um método do nosso componente *Animation* e depois verificando a propriedade `isPlaying`. Tudo isso é sim necessário mas, faz parte da Cocos, da ferramenta que estamos utilizando.

Para deixar nossa lógica mais fácil de ser entendida precisamos extrair a maneira de escrever esse código, do ferramental para o domínio do problema. Ou seja, vamos criar um método que internamente vai chamar essa linha, mas que no nome dele extrai o que realmente estamos tentando resolver.

```
animacaoEstaTocando: function (animacao) {
    return this._animacao.getAnimationState(animacao).isPlaying
},
```

Dessa forma temos um código que ao ser lido "fala" para outro programador o que estamos querendo resolver. A maneira como resolvemos fica "escondida" na camada de baixo do nosso código.

Podemos fazer a mesma coisa para a linha `let angulo = Math.atan2(direcao.y, direcao.x);`. O que nós interessa não é a maneira como achamos o ângulo e sim achar o ângulo.

Fazendo isso e reescrevendo nosso método `mudaAnimação` temos um código muito mais simples de ser lido e entendido.

```
    mudaAnimacao: function (direcao, estado) {
        let proximaAnimacao = estado;

        let angulo = this.calcularAngulo(direcao);
```

```
        let cosseno = Math.cos(angulo);
        cosseno = Math.abs(cosseno);

        let seno = Math.sin(angulo);
        seno = Math.abs(seno);

        if (cosseno >= this._cossenoMinimo) {
            proximaAnimacao += this.verificarEixo(direcao.x, "Direita", "Esquerda");
        }
        if (seno >= this._senoMinimo) {
            proximaAnimacao += this.verificarEixo(direcao.y, "Cima", "Baixo");
        }

        if (!this.animacaoEstaTocando(proximaAnimacao)) {
            this._animacao.play(proximaAnimacao);
        }

    },

    animacaoEstaTocando: function (animacao) {
        return this._animacao.getAnimationState(animacao).isPlaying
    },

    verificarEixo: function (valorEixo, sentidoPositivo, sentidoNegativo) {
        if (valorEixo > 0) {
            return sentidoPositivo;
        } else if (valorEixo < 0) {
            return sentidoNegativo;
        }
    },

    calcularAngulo: function (direcao) {
        let anguloEmRadianos = Math.atan2(direcao.y, direcao.x);
        return anguloEmRadianos;
    },
```