01

Fantasmas

Transcrição

[00:00] É hora de colocar os fantasmas para correr. Se vamos trabalhar com movimento, vamos trabalhar tanto com o nosso quanto dos fantasmas. Primeiro, vamos olhar o nosso. Ele está baseado no calcula nova posição, quando recebemos o herói e a posição. Nós verificamos quatro condições diferentes com o código cheio de ifs. A lógica de ter quatro condições diferentes não tem escapatória. A complexidade é intrínseca, natural dessa nossa lógica. Tem quatro coisas diferentes que posso fazer. Tenho que de alguma maneira poder fazer quatro coisas. A questão é como mostrar ao programa que tenho essas quatro coisas para fazer.

[00:55] Nesse instante o que eu estou fazendo é mudando a variável zero, a variável um, está bem despadronizado. Cada vez faço uma coisa totalmente diferente. Nós poderíamos padronizar mais esse código e ver o que podemos refatorar.

[01:20] A primeira coisa é reparar que sempre faço algo no zero e no um. Nesse caso, do menos igual a um, seria a mesma coisa que falar que na posição um eu não vou subir nada. É um código desnecessário, mas quero padronizar para ver o que pode surgir à medida em que todos eles fazem a mesma coisa. Todos eles somam ou subtraem algum valor do zero ou do um. Se eu tivesse mais experiência, poderia traduzir logo que eles fazem a mesma coisa e extrair para o código final logo de cara, mas aqui estou indo passo a passo.

[02:14] Se eu mudar o menor igual para maior igual a menos um, eles vão mesmo fazer a mesma coisa, que é somar um valor no zero e somar um valor no um. Padronizamos. Se logo de cara eu tivesse feito essa refatoração diversas vezes na minha vida, eu saberia dizer rapidamente. Depois de muitas vezes pegamos essa prática.

[02:56] O que quero fazer agora é olhar esse código e perceber que em um ando na linha e no outro na coluna. Posso resumir com herói zero anda na linha e herói um anda na coluna. Com muita paciência, vamos trocando os nomes.

[03:38] Lembra que as variáveis locais em Ruby são locais à minha função? Então o anda linha da coluna está disponível para ser acessado aqui.

[03:50] Isso tudo segue um padrão ainda mais claro. Um padrão de sempre se movimentar em dois valores possíveis, linha e coluna. Se for W, ande menos um e zero. Se for S, ande mais um e zero. Se for A, ande zero e menos um. E se for D, ande zero e mais um. É o padrão que temos.

[04:20] Como eu conecto a letra ao array? Com uma seta. Colocando a letra como uma string. Quero criar um mapa, alguém que associe as letras aos números. Quero criar um dicionário, alguém que seja capaz de traduzir um valor em outro valor. Procurar um valor e encontrar o valor que está mapeado a ele. Bem comum usar os termos dicionário e mapa. Apesar do termo mais utilizado em Ruby ser uma array associativa. Ele possui quatro elementos, só que ele associa o elemento da esquerda com o elemento da direita. Poderia ser uma string, uma array na direita, qualquer coisa associada a qualquer coisa.

[05:59] Movimentos é esse meu dicionário, que para ser definido uso chaves. Colchetes usamos para definir um array. Chaves para definir um array associativo. Agora, vou dizer que o movimento que quero executar é o array correspondente à letra. Ao invés de usar zero, um, dois, três para acessar o array posicional, vou usar através da associação, da string. A string está na variável direção. Toda vez que eu usar o número, vou usar a letra. Ou seja, a direção.

[07:02] Dada a direção, pego do array associativo o que está associado a ele. Tudo funcionando. Criamos uma array associativa, um dicionário, um mapa. Eu dou para ele um valor à esquerda e ele me devolve o da direita. Troquei aquele

case por quatro condições de uma array associativa.

[07:57] Cuidado. A lógica ainda tem a complexidade de quatro condições diferentes. Disso não temos como fugir. Temos que ter quatro possibilidades, senão você não pode andar para os quatro lados.

[08:12] Outro ponto importante é notar que usamos um recurso de estrutura de dado, um dicionário, um mapa, para implementar uma lógica que de outra maneira é implementada através de vários ifs. Qual dos dois é melhor? São perguntas válidas. Se você achar que é melhor deixar o case, você pode deixar. As estruturas que vamos criando para trabalhar com a lógica do nosso negócio podem melhorar ou piorar a legibilidade e a manutenção a longo prazo. Eu acredito que nessa situação conseguimos deixar concisos os movimentos possíveis, extraímos o movimento e efetuamos, então vou deixar assim.

[09:00] Não acredito que a array associativa como uma maneira de selecionar elementos seja a grande solução mágica para todos os momentos em que tenho condicionais do meu código. Não é. Mas nesse caso ficou mais simples. Claro que eu teria que tratar uma letra inválida, assim como antes eu teria que tratar as letras inválidas. Como em qualquer situação.