

Relatório Trabalho Prático II

Christoffer de Paula Oliveira, Paulo Henrique Tobias, Millas Násser

UFSJ- Universidade Federal de São João Del Rei

Table of Contents

Sumário	1
Introduction.....	1
1 Classe Jogo	2
1.1 Mudanças e Justificativas.....	2
2 Classe Sala	2
2.1 Mudanças e Justificativas.....	3
3 Classe Jogador	3
3.1 Mudanças e Justificativas.....	3
4 Classe Localizável	3
4.1 Mudanças e Justificativas.....	3
5 Classes Chave, Machado, Pocaio e Ouro	3
5.1 Mudanças e Justificativas.....	4
6 Classe Porta.....	4
6.1 Mudanças e Justificativas.....	4
7 Classe Troll	4
7.1 Mudanças e Justificativas.....	4
8 Classe TrollNome	4
8.1 Mudanças e Justificativas.....	4
9 Pontos positivos	5
10 Mudanças gerais	5
10.1 Exceções	5
Mudanças e Justificativas.....	5
11 Evolução.....	5
11.1 Salas	6
11.2 Machados	6
11.3 Jogador	6
12 Conclusão.....	7

Introdução

Este relatório tem como objetivo documentar as mudanças feitas no Trabalho Prático I para que este se adeque aos conceitos de Programação Orientada a Objetos (POO). Além disso, o relatório também relata o que ajudou e o que atrapalhou na implementação das novas funcionalidades requeridas no Trabalho Prático 2.

1 Classe Jogo

A classe Jogo estava com um alto nível de acoplamento. Várias funções – que, segundo a ideia de coesão, pertencem à outras classes – estavam implementadas aqui. Além disso, havia muito código repetido, que mais tarde foi transformado em métodos. Estes fatores deixavam o código muito grande e quase ilegível.

Praticamente todos os objetos do jogo eram declarados aqui. Além da lista de salas e trolls, também existiam listas para cada tipo de item. Estes itens, durante a inicialização, seriam espalhados aleatoriamente pelas salas. Para isso, existia uma classe *Localizavel* que guardava a localização de todos os objetos do jogo (Incluindo Jogador e Trolls).

A verificação e execução dos comandos do jogador também era feita nesta classe. Ações como pegar um item, por exemplo, que pertencem à classe Jogador, eram feitas na classe Jogo.

1.1 Mudanças e Justificativas

Para diminuir o acoplamento, foi criada uma nova classe chamada *Console*, que faz o processamento da entrada do usuário e chama as funções corretas, cada uma implementada na sua devida classe. Portanto, se o usuário digitou o comando "pickup gold", o console agora apenas identifica que é o comando "pegar" e chama a função correspondente da classe Jogador.

Além disso, uma nova classe *Mapa* foi criada, que guarda a lista de salas e o objeto Jogador. A lista de trolls, assim como as de itens, foram transformadas em listas individuais para cada sala. Assim, cada sala passou a ter suas próprias listas de trolls e de itens.

Ao final das modificações, a classe jogo apenas chama a função *console* até que o jogo termine.

2 Classe Sala

O principal problema das salas, além do já explicado acima, é que o método que inicializava o conjunto das salas estava implementado ali (veremos mais adiante que este foi um problema em quase todo o projeto). Podemos achar que esse tipo de método pertença à classe Sala, porém quem tem a função de inicializar as salas, ou seja, criar o mapa, é o próprio mapa.

2.1 Mudanças e Justificativas

A inicialização do conjunto de salas foi passada para a classe Mapa. Como dito anteriormente, as listas de trolls e itens foram movidas para a Sala. Por consequência disso, foram criados métodos para adicionar, remover, retornar e imprimir os itens e trolls das salas.

3 Classe Jogador

Pouco coesa. Os métodos que deveriam estar aqui – pegar, largar, mover, sair, atacar, etc – estavam em outros módulos.

Para armazenar o objeto (porta ou item) que estava perto do jogador eram usados dois atributos. Um do tipo *Porta* – para armazenar a porta – e outro do tipo *String* para armazenar os demais itens.

3.1 Mudanças e Justificativas

Largar e pegar itens, lançar machado, sair da sala e perder ouro, que são métodos responsáveis por realizar ações do jogador foram implementados aqui deixando o código mais coeso.

Foi criada uma classe *Aproximavel* que indica que o jogador pode se mover para perto deste objeto. Portanto, apenas um atributo é necessário para guardar esta informação. Com a remoção da classe *Localizavel*, foi criado um atributo do tipo *Sala* para armazenar a sala atual do jogador.

4 Classe Localizável

Praticamente todo o código era dependente dessa classe.

4.1 Mudanças e Justificativas

Essa classe foi excluída ocasionando em um bom desnível de acoplamento no código, com isso todo o restante do código que necessitava dessa classe foi adaptado para não depender mais.

5 Classes Chave, Machado, Poca e Ouro

Todas possuíam o mesmo problema que a classe sala: possuíam métodos que aparentavam pertencer à classe mas que na verdade deveriam estar em outra.

Índice de coesão extremamente baixo. Praticamente todos os métodos foram movidos para outros módulos.

5.1 Mudanças e Justificativas

Muitos métodos foram removidos e outros foram realocados. Os métodos de inicialização (presentes em todas as classes desta seção) foram transformados em um único método implementado na classe *mapa*. Estes métodos eram responsáveis por espalhar os itens pelas salas do jogo, e por isso deixavam a classe menos coesa. O efeito de cada item passou a ser implementado dentro de cada classe, ou seja, se o jogador tenta usar uma chave, por exemplo, então a função *usar* da classe *Chave* é chamada e a porta é destrancada.

No código original, o usuário podia entrar com vários nomes diferentes para cada item. Para a poção, por exemplo, eram reconhecidos "poção", "pocao" e "potion". O problema é que isso era feito manualmente para cada item e isto acabava gerando muito código repetido. Para manter a mesma funcionalidade e ao mesmo tempo usar os conceitos aprendidos na disciplina de POO (usando as ideias de Herança de Tipo e Implementação) foi criado o método *comparar* na super-classe *Aproximavel* (portas também gozam deste método). Este permite comparar uma palavra com todas os "nomes" do item, permitindo que ele possa ser acessado de várias maneiras diferentes.

A função que efetivamente compara o "nome" de um item à sua lista de nomes foi criada na classe *Util*, pois não se adequava a nenhum módulo existente.

6 Classe Porta

Poucas modificações foram feitas nesta classe.

6.1 Mudanças e Justificativas

Duas modificações foram feitas. A primeira delas foi retirar o método *getPortaByIdentificado*, que foi renomeado para *getPorta* e transferido para o módulo *Sala*, aumentando a coesão da classe. A segunda foi alterar o tipo do atributo *salaSaida* de *String* para *Sala*.

7 Classe Troll

Acoplada por depender de *sala* e também com baixa coesão, métodos de outras classes, como inicializa troll, estavam sendo implementados nessa classe.

7.1 Mudanças e Justificativas

Métodos de outras classes foram movidos para os seus devidos lugares, e método de mover trolls modificado para se adaptar ao novo código.

8 Classe TrollNome

Uma função de troll que poderia estar implementada na classe *Troll*.

8.1 Mudanças e Justificativas

Caso precise ser mais detalhado

9 Pontos positivos

Houve também pontos em que o código estava seguindo a Programação Orientada a Objeto. O encapsulamento foi feito de maneira correta, As classes Mochila e Pegavel para o antigo código estava sem problemas,isto é, nenhum erro de conexão ou acoplamento

10 Mudanças gerais

A partir do momento que erros de Programação Orientada a Objeto foi corrigidos, houve mudanças para a realização da expansão do código. O jogo passou a ter outras características tal como especificado na documentação da tarefa.

10.1 Exceções

O código inicial não se provia de nenhum meio para a detecção de exceções lançadas pelo jogo. Portanto os poucos meios de se evitar algum erro eram feitos a partir de retornos de funções que por sua vez tornavam o código um pouco difícil de se ler já que não havia nenhuma especificação sobre.

Mudanças e Justificativas Para tal foi criado três conjuntos de exceções

- Personagem
- Item
- Aproximavel

Exceções do tipo personagem, são responsáveis por lançar mensagens para trolls, que são por sua vez indicam quais movimentos ele poder fazer e sobre o que ele pode interagir. Já para o jogador, estas fazem parte de todas as ações do mesmo, partindo desde interação de algum objeto presente, movimentação, e se o mesmo escreveu algum comando errado.

Exceções geradas por algum item geralmente é que os mesmos não puderam realizar suas devidas ações, como exemplo abrir portas ou usar poção em algum lugar inválido.

Por fim a última exceção a ser gerada pelo programa acontece pela interação das duas anteriores, estas geram um erro quando o personagem tenta se aproximar de algum item em específico e não foi possível realizar a ação desejada como chegar perto de portas que não existem na sala e entre outros.

11 Evolução

Para essa nova abordagem, o jogo apresenta novas funcionalidades.

11.1 Salas

Salas agora possuem tamanhos diferentes, com isso as salas serão divididos em metros quadrados que por sua vez visam limitar a quantidade de ouro disponível, onde no máximo podem caber dez peças de ouro por metro quadrado. Portanto salas de tamanhos diferentes armazenam quantidades diferentes.

Com essa modificação no tamanho, chegamos a outro ponto, elas por sua vez não podem possuir a mesma configuração de tamanho.

Para isso foi adicionado um novo campo na classe Sala, que é tamanho de cada uma para saber identificar qual a quantidade certa de ouro. E a criação das mesmas agora são feitos a partir de um arquivo json, diferentemente de antes que era feita a partir da sua própria classe. Deixando o código mais limpo.

11.2 Machados

Para essa nova abordagem, o jogo apresenta novas configurações de machados.

- Machado de ouro
- Machado de bronze
- Machado de ferro

Para isso a classe Machado anteriormente se tornou abstrata e recebeu dois novos atributos.

O primeiro deles foi a durabilidade, essa indica qual a quantidade de vezes o machado será lançado e o outro campo é o material, fornecendo a durabilidade individual de cada machado.

Machados de ferro possuem durabilidade um, ou seja, podem ser lançados apenas uma vez. Os de bronze podem ser lançados em até duas vezes e por fim machados de ouro são lançados 5 vezes.

A abordagem foi criar mais 3 novas classes, uma para cada material acima descrito. Todas elas possuem funcionamento em comum, portanto todas herdaram da classe machado.

Em seu lançamento cada machado é responsável por saber quem é seu alvo de ataque e qual seu efeito sobre ele, o se lançar em um Troll este será diminuído em sua durabilidade. Mas se o ataque for ao jogador então será mandado um sinal para a classe mochila que ficará a cargo de remover os itens necessários, dependendo da durabilidade do machado.

11.3 Jogador

A principal mudança realizada foi expandir a mochila e limitar a quantidade de itens de cada tipo que o jogador pode carregar. Pode-se agora levar até quatro machados, independente de seu material, poções e chaves são possíveis carregar até três delas, não mais que isso.

Para simplificar o processo a mochila agora conta com três listas de cada item descrito acima. E ao fazer a verificação é necessário apenas ver o tamanho de cada uma.

12 Conclusão

Tomando um código bastante acoplado e pouco coeso, notou-se muita dificuldade em se evoluir o código apresentado, para isso foi necessário primeiramente colocá-lo no formato desejado, após realizar a alteração, em questão de pouco tempo as novas alterações foram implementadas. Portanto os conceitos desenvolvidos pela disciplina de POO foram muito importantes, observando que um código pouco acoplado é mais fácil de se dar manutenção e para se fazer testes.

Além de trechos onde não se podia ter a garantia de não haver erros, foram tratados também a partir de conceitos apresentados em sala

Logo para projetos onde se apresenta o dilema de programação em grupo, a programação modular se torna indispensável pois evita demasiado acesso a trechos críticos do projeto. Outro ponto a se enaltecer é a necessidade de deixar o código legível a outros programadores evitando perda desnecessária de tempo ao identificar o funcionamento dos trechos do programa.

(Example from Jensen K., Wirth N. (1991) Pascal user manual and report. Springer, New York)