TP do Jogo

Prof. Daniel Conrado

1 Introdução

Para este TP, você deve criar um jogo usando como estrutura básica o projeto do World of Zuul (já com as melhorias de design feitas em sala de aula). Seu jogo será avaliado em termos de qualidade de design do código-fonte, mas também de criatividade e jogabilidade. Embora não seja obrigatório, recomendo usar uma ferramenta de versionamento de código, como o **git**.

Você deve submeter pelo Classroom:

- uma breve descrição do jogo em PDF;
- os arquivos *.java soltos, sem compactar, OU um link para um repositório online (ex. Github) contendo os arquivos.

2 Descrição

Projete o seu próprio cenário de jogo. É importante que você faça isso longe do computador. Não pense em implementação, classes, ou qualquer coisa relacionada a programação. Apenas pense em inventar um jogo interessante.

O jogo pode ser qualquer coisa que tenha como estrutura base um jogador se movendo ao longo de diferentes locais. Eis alguns exemplos:

- você é um glóbulo branco andando pelo corpo em busca de virus para atacar...
- você está perdido em um shopping e precisa encontrar a saída...
- você é uma toupeira dentro da toca e não consegue se lembrar onde guardou as suas reservas de comida para o inverno...
- você é um aventureiro em busca de algo dentro de um calabouço cheio de monstros e outros caracteres...
- você faz parte do esquadrão antibomba e precisa encontrar e desarmar uma bomba antes que ela exploda...

Certifique-se de que seu jogo tenha uma meta/objetivo, de tal forma que há um final e que o jogador possa "ganhar/vencer". Tente pensar em várias coisas para tornar o jogo interessante (alçapões, itens mágicos, personagens que te ajudam se você der algo a eles, limites de tempo ou de movimentações... enfim, o que você quiser). Deixe a sua imaginação alçar voo. Novamente, não se preocupe em como implementar essas coisas.

Você deve desenhar um mapa do seu jogo e incluí-lo no PDF.

3 Tarefas

Vou listar aqui uma série de tarefas que você deve realizar em seu jogo. Várias delas estão relacionadas aos conceitos de coesão, acoplamento e encapsulamento. A ideia é que cada tarefa envolva uma melhoria no design do código-fonte. Primeiro, faça uma cópia do projeto **zuul-bad** e renomeie-a para o nome do seu jogo. Você deve realizar as tarefas nesse seu projeto.

Tarefa 1. Mude o método **createRooms** da classe **Game** para criar as salas e saídas que você inventou para o seu jogo.

Tarefa 2. Remova o código duplicado encontrado na primeira versão dos métodos **printWelcome** e **goRoom** da classe **Game**, e crie um método separado **printLocationInfo**.

Tarefa 3. Nesta versão, as informações sobre as possíveis saídas de uma localização (**Room**) são codificadas como campos públicos. Isso é um problema de falta de encapsulamento adequado, o que levou a um alto acoplamento entre as classes **Room** e **Game**. Use encapsulamento para reduzir o acoplamento: torne os campos privados e introduza o método **getExit(String)** à classe **Room**, conforme explicado em sala.

Tarefa 4. Faça uma alteração similar ao método **printLocationInfo** de **Game**, para que os detalhes das saídas sejam preparados pela classe **Room** ao invés de **Game**[1]. Defina um método na classe **Room** com o seguinte cabeçalho:

```
/**
 * Retorna uma descrição das saídas deste Room,
 * por exemplo, "Exits: north west".
 * @return Uma descrição das saídas disponíveis.
 */
public String getExitString()
```

Tarefa 5. Agora já é possível alterar a estrutura interna de **Room** de maneira *localizada*, ou seja, sem que essas alterações propagem-se para outras classes. No entanto, há ainda um método que precisa ser modificado para tornar a classe **Room** mais flexível. Troque o método **setExits(Room north, Room east, Room south, Room west)** por um método **setExit(String direction, Room neighbor)**, de tal forma que, ao invés de escrevermos, por exemplo:

```
lab.setExits(outside, office, null, null);
```

Vamos escrever:

```
lab.setExit("north", outside);
lab.setExit("east", office);
```

[1] Afinal, quem sabe das saídas é a própria classe **Room**, então, é natural que ela saiba dizer quais são as saídas possíveis.

Depois, troque a forma como as saídas são armazenadas. Ao invés de um campo por saída, use um **HashMap** para armazenar as saídas.

Tarefa 6. Provavelmente o seu método **getExitString** da classe **Room** esteja verificando apenas as saídas norte, leste, sul e oeste. Modifique esse método para que ele deixe de ter a responsabilidade de saber as saídas existentes e delegue isso para o objeto **HashMap** que você criou na tarefa anterior. Você vai precisar saber como funciona o método **keySet** do **HashMap** e usá-lo juntamente com um *for-each loop* para produzir a string com as saídas.

Tarefa 7. Atualmente, nós codificamos em **Game** que a informação que precisamos de um objeto **Room** é a sua descrição e a string de saída, como é possível ver no trecho de códgio abaixo:

```
System.out.println("You are " + currentRoom.getDescription());
System.out.println(currentRoom.getExitString());
```

Mas e se quisermos adicionar itens à sala? Ou monstros? Ou outros jogadores?

Teríamos não só de alterarmos a classe **Room** para adicionar essas coisas, como também alterar o trecho de código acima para imprimir também essas outras informações!

Isso "viola" as regras do projeto dirigido por responsabilidades. Como é a classe **Room** que armazena as informações sobre uma localização, ela também deve ser responsável por produzir uma descrição dessa localização. Para fazermos isso, vamos adicionar um método **getLongDescription** à classe **Room**, para retornar uma String contendo a descrição do quarto mais a string das saídas; de tal forma que possamos substituir as duas linhas de código para por apenas uma:

```
System.out.println(currentRoom.getLongDescription());
```

A "descrição longa" da localização agora inclui a descrição e as informações sobre as saídas, e pode no futuro incluir qualquer outra coisa que se pode dizer sobre a localização (presença de monstros, itens etc.). Qualquer extensão futura nesse aspecto vai requerer mudanças apenas na classe **Room**.

Tarefa 8. Adicione o comando **look** no seu jogo.

Tarefa 9. Adicione um outro comando no seu jogo. Para começar, você pode começar com algo simples, como um comando **eat** que, quando executado, apenas imprima na tela "Você já comeu e não está com fome mais". Depois, você pode melhorar isso para que o jogador vá ficando com fome ao longo do tempo e precise encontrar comida.

Tarefa 10. Conserte o *acoplamento implícito* existente entre a classe **Game** (método printHelp) e a classe **CommandWords**. Note que **printHelp** atualmente assume que os comandos disponíveis são **go, quit** e **help**. Eles não incluem o comando **look** ou qualquer outro comando que você adicionou no jogo.

Tarefa 11. Estenda o seu projeto para que uma localização possa conter um item. Os itens tem uma descrição e um peso. Ao criar localizações (rooms) e configurar suas saídas, crie também os itens. Quando um jogador entrar em uma localização, as informações sobre o item nela devem ser impressas.

Para pensar: como a informação deve ser produzida sobre o item presente na localização? Qual classe deve produzir a string que descreve o item? E qual classe deve imprimí-la? Por quê?

Tarefa 12. Modifique o projeto para que uma localização possa armazenar qualquer quantidade de itens. Use uma coleção para isso. Certifique-se de que a localização possua um método para adicionar um item na localização. Certifique-se também de que todos os itens da localização são apresentados ao jogador quando ele entra nela.

Tarefa 13. Implemente o comando **back**. Este comando não tem uma segunda palavra. Ao entrar com **back**, o comando devolve o jogador à localização anterior onde ele estava.

Teste seu comando. Ele funciona como o esperado? Teste também casos em que o comando é usado incorretamente. Por exemplo, o que o seu programa faz se um jogador escreve uma segunda palavra depois do comando **back**? O comportamento faz sentido?

O que o seu programa faz quando você escreve "back" duas vezes? Faz sentido?

Implemente o comando **back** para que, se usado repetidamente, vai retornando o jogador para as suas localizações anteriores, podendo até chegar no início do jogo. Você vai precisar de uma *pilha* para fazer isso. Será que a biblioteca padrão de Java possui uma classe de pilhas?[2]

Tarefa 14. Refatore o seu projeto para introduzir uma classe separada representando um jogador (classe **Player**). Um objeto **Player** deve armazenar pelo menos a sua localização atual, mas você pode também querer armazenar o nome do jogador ou qualquer outra informação.

Tarefa 15. Implemente uma extensão que permita a um jogador pegar um item. Isso inclui implementar dois comandos: **take** e **drop**.

Depois, estenda a sua implementação para permití-lo carregar qualquer quantidade de itens.

Tarefa 16. Adicione uma restrição que permita ao jogador carregar apenas uma quantidade máxima de peso. O peso máximo é um atributo do jogador.

Tarefa 17. Implemente o comando **items**, que imprime todos os itens que o jogador está atualmente carregando, junto com o peso total.

[2]Pilha é um tipo de coleção de objetos, tal que os objetos adicionados vão empilhando-se, formando uma pilha (p. ex. uma pilha de livros). A principal característica da pilha é que o primeiro elemento que você tira dela é sempre o último elemento que foi adicionado.

Tarefa 18. Adicione como item um *cogumelo mágico* a uma localização. Adicione um comando do tipo **comer cogumelo**, por exemplo. Se um jogador encontra isso e come, isso aumenta a sua capacidade de carregamento de peso. (Naturalmente, você pode modificar essa tarefa para combinar melhor com o cenário do seu próprio jogo.)

Tarefa 19. Adicione a classe **GameMain** ao seu projeto. Defina apenas o método **main** e, nele, apenas crie um objeto da classe **Game** e chame seu método **play**.

4 Sugestões

As próximas tarefas são apenas *sugestões* que você pode seguir ou não para incrementar o seu jogo. Como você pode perceber, este jogo apresenta várias possibilidades de extensão. Eu te incentivo a inventar as suas próprias. Você pode fazer mais que isso, ou fazer extensões diferentes das aqui apresentadas. Eis algumas sugestões:

1. Adicione uma espécie de tempo limite ao seu jogo. Se uma determinada tarefa não é completa em um dado tempo, o jogador perde. Um tempo limite pode ser facilmente implementado contando o número de movimentos ou o número de comandos digitados. Não é preciso usar um tempo real.

2. Implemente um alçapão em algum lugar, ou alguma forma de porta que você só pode através em uma direção.

3. Adicione um equipamento de teletransporte no seu jogo. Seria um equipamento que você aciona e dispara. Quando você o aciona, ele memoriza a sua localização atual. E, depois, quando você o dispara, ele te teletransporta para o lugar onde ele foi acionado.

4. Adicione portas trancadas ao jogo. O jogador deve procurar uma chave para abrir a porta.

5. Adicione uma porta mágica que aleatoriamente te transporta para alguma outra localização dentro do jogo.

6. Adicione personagens ao jogo. Eles são semelhantes aos itens, mas eles podem falar. Eles podem te dar alguma ajuda se você der algum determinado item para eles.

7. Adicione personagens que movem de localização. Cada vez que um jogador digita um comando, esses personagens movem-se para alguma localização adjacente.