MAC-212 — Laboratório de Computação

Exercício Programa 2: A Aventura Continua!

Prazo: 4 de junho

1 Introdução

Nosso objetivo agora é melhorar o sistema de execução de jogos tipo *adventure* desenvolvido no primeiro exercício programa. Esse sistema pode ser melhorado em várias direções. Três linhas de trabalho distintas são:

- separar do programa a definição do mundo virtual e permitir que o estado do jogo seja salvo num arquivo;
- fazer o jogo rodar num ambiente de janelas;
- permitir múltiplos jogadores interagindo uns com os outros via rede (jogo multiusuário).

Neste exercício programa você deve escolher **uma** dessas propostas e trabalhar nela. As três seções seguintes deste enunciado descrevem cada uma das linhas de trabalho sugeridas.

Como no primeiro exercício programa, vocês trabalharão em equipes de uma ou duas pessoas. Quem fizer o projeto em dupla, lembre que todos os detalhes da solução devem ser discutidos e entendidos pelos dois integrantes da equipe.

2 Carregar Mundo Virtual de Arquivo e Salvar Estado

Nesta linha de trabalho você fará duas coisas:

1. Retirar do código Java a definição do mundo virtual. O programa deverá ler essa definição de um arquivo de texto, que pode ter sido escrito por um "criador de jogos" sem nenhum conhecimento de programação.

2. Permitir que o estado do jogo seja salvo num arquivo e restaurado numa futura execução do jogo.

O ítem 2 acima é simples, graças aos recursos de serialização de objetos de Java. Basta garantir que o estado do jogo seja dado por um objeto (ou conjunto de objetos) serializável (isto é, que implemente a interface `java.io.Serializable`) e usar adequadamente os métodos `writeObject()` (da classe `java.io.ObjectOutputStream`) e `readObject()` (da classe `java.io.ObjectInputStream`).

Já o ítem 1 requer maior planejamento. Em vez de inventar um formato especial para o arquivo de texto com a definição do mundo virtual, usaremos XML (*Extensible Markup Language*), uma "linguagem de marcação" padronizada, que nos oferece as seguintes vantagens:

- É adaptável às necessidades específicas de uma aplicação ou classe de aplicações.
- Existem bibliotecas de processamento de XML disponíveis para Java e para outras linguagens de programação.
- Como XML está sendo cada vez mais usada para quase tudo (Internet, bancos de dados, aplicações como transferência de fundos, comércio eletrônico, etc), vale a pena aprender um pouco dela.

Eis um trecho de uma definição de um mundo virtual em XML:

```
<place>
  <place-name>clareira</place-name>
  <place-description>
  Você está numa pequena clareira na floresta. A vegetação a
  seu redor é fechada e impenetrável. Ao leste se vê o início
  de uma trilha muito estreita. Ao sul há algo que parece ser a
  entrada de uma caverna.
  </place-description>
  <exit>
    <exit-name>sul</exit-name>
    <exit-destination>caverna</exit-destination>
  </exit>
  <exit>
    <exit-name>leste</exit-name>
    <exit-destination>trilha</exit-destination>
  </exit>
</place>
```

Cada elemento do arquivo XML é demarcado por *tags* de início (com a forma `<algum-tag>`) e final (com a forma `</algum-tag>`). Os *tags* são dependentes da área de aplicação (você pode escolhê-los), assim como as regras de aninhamento de elementos. Essas regras dizem quais os elementos que precisam obrigatoriamente aparecer dentro de um certo elemento, quais os elementos que podem opcionalmente aparecer dentro de um certo elemento, etc. No caso de um elemento "lugar", é razoável que ele contenha um elemento "nome" e um elemento "descrição" (ambos obrigatórios), um conjunto (possivelmente vazio) de elementos do tipo "saída", e o que mais for conveniente (elementos para coisas, personagens, etc). Esse tipo de regra é especificada numa *Document Type Declaration* (DTD) específica para uma área de aplicação.

2.1 Um Empurrão com essa História de XML

Uma "solução incompleta" para a parte de XML está disponível na página de MAC–212. Essa solução inclui os arquivos `world.xml` (uma descrição de um mundo virtual muito simples) e `world.dtd` (uma pequena DTD com as regras de descrição do mundo), que você pode usar como modelos para construir um mundo virtual mais elaborado. Ela também inclui novas versões das classes fornecidas como empurrão inicial para o primeiro exercício programa (`br.usp.ime.mac212.adventure.Place`, `br.usp.ime.mac212.adventure.Main` e `br.usp.ime.mac212.util.Console`), bem como a classe `br.usp.ime.mac212.util.XML`, que é uma classe auxiliar útil para fazer leitura de arquivos XML. Essa "solução incompleta" é muito parecida com a que foi fornecida para o primeiro exercício programa. Há, no entanto, uma importante diferença: o programa principal agora lê o mundo virtual do arquivo `world.xml`, que pode ter sido escrito por uma pessoa sem conhecimentos de programação.

Sua primeira tarefa é estudar e rodar a "solução incompleta". Entenda bem a DTD fornecida, pois você precisará alterar as regras de descrição de mundo para permitir que lugares tenham coisas, personagens, etc. Experimente mexer no arquivo `world.xml` e tornar inválida a descrição do mundo. Adicione um *tag* não previsto nas regras de descrição, ou "esqueça" um *tag* de fechamento de elemento, ou omita um elemento obrigatório, como o nome que precisa aparecer dentro de um lugar. Veja como o programa reclamará se um criador de jogos distraído fizer alguma besteira.

Note que as classes fornecidas como "empurrão inicial" não fazem o salvamento do estado do jogo num arquivo nem a restauração do estado do jogo de um arquivo. A versão do programa que você entregará deve ler um argumento da linha de comando ou interagir com o usuário de algum modo para saber se o usuário quer um jogo novo ou quer continuar um jogo que ele salvou antes. Se for jogo novo, o programa lê o arquivo XML com a descrição do mundo e cria um mundo virtual novo. Se for continuação de jogo salvo, o programa lê o estado do jogo de um arquivo especificado pelo usuário. Neste caso o arquivo XML nem será lido, pois todo o mundo virtual será restaurado a partir do arquivo com o estado do jogo.

3 Jogo Num Ambiente de Janelas

Nesta linha de trabalho você adicionará a seu programa uma interface gráfica com o usuário (GUI) e fará o jogo rodar num ambiente de janelas. A proposta **não** é fazer um jogo gráfico com imagens animadas: isso demandaria uma quantidade de conhecimentos e de trabalho incompatível com esta disciplina. O jogo continuará a ser textual, mas o texto que descreve o lugar atual será apresentado num componente textual de uma janela. Veja exemplos em `http://www.linux.ime.usp.br/java/tutorial/uiswing/components/simpletext.html`.

Sempre que possível, o usuário deve interagir com o jogo através de botões ou de menus. Alguns exemplos:

- Faça os caminhos que saem de um lugar aparecerem num conjunto de botões cujos rótulos e cuja quantidade variam de um lugar para outro.

- Cada uma das coisas existentes no lugar pode ser apresentada como uma *combo box* com as ações que tem significado para essa coisa.

- Ponha no topo da janela principal do jogo uma *menu bar* através da qual o usuário pode sair do jogo, salvar o jogo, restaurar um jogo salvo, etc.

Faça o usuário digitar numa caixa de entrada de texto só quando não houver outro jeito.

Java tem dois pacotes de janelas, o AWT (presente desde a primeira versão da linguagem) e o Swing (mais recente, apareceu na versão 1.2). Não use o AWT, pois o Swing é muito melhor e de utilização bem mais fácil.

Se você tiver à sua disposição um programa tipo *GUI builder* e quiser utilizá-lo, pode fazer isso. Note, entretanto, que um *GUI builder* não é muito útil quando os componentes visuais na janela não forem conhecidos de antemão. Pense no caso dos botões que representam as saídas de um lugar. A menos que você construa uma janela específica para cada lugar[1], um *GUI builder* não o ajudará a dispor esses botões na janela.

4 Jogo Multiusuário Via Rede

Aqui a proposta é fazer seu jogo rodar num servidor que é empregado por vários jogadores ao mesmo tempo. Não se assuste, isso não é tão difícil quanto pode parecer à primeira vista. Caso você não conheça um jogo assim, rode (da linha de comando)

```
telnet 195.34.31.70 4000
```

ou (dá na mesma) aponte um *browser* para

```
telnet://195.34.31.70:4000
```

4.1 Um Empurrão com essa História de Servidor

Está disponível na página de MAC–212 uma versão multiusuário da "solução incompleta" do primeiro exercício programa. O código disponibilizado implementa um "servidor de jogo" bem simples, que recebe conexões de usuários que rodam `telnet`. Comece experimentando esse programa: compile as cinco arquivos fonte fornecidos e gere as classes `GameServer`, `Session`, `EndOfSessionException`, `Game` e `Place`, todas elas pertencentes ao pacote `br.usp.ime.mac212.adventure`. A seguir execute o servidor, com o comando

```
java br.usp.ime.mac212.adventure.GameServer
```

Aparecerá a mensagem

```
Servidor pronto para receber conexões
```

Para encerrar a execução do servidor basta digitar Ctrl-C. Não faça isso ainda. Com o servidor rodando, abra outra janela de comandos (*prompt* do MS-DOS ou *shell* do Linux) e diga

```
telnet localhost 4444
```

Depois de fornecer um nome para seu personagem, você entrará num jogo praticamente igual ao "solução incompleta" do primeiro exercício programa. A diferença é que podem haver vários jogadores ao mesmo tempo. Abra mais uma janela e dê o mesmo comando `telnet` outra vez. Forneça outro nome de personagem. Note como um personagem pode ver o outro.

Caso o servidor de jogos esteja rodando na internet ou mesmo numa intranet, os usuários poderão se conectar com ele a partir de outras máquinas da rede. Basta trocar o "`localhost`" pelo nome da máquina onde está rodando o servidor. Por exemplo: com o servidor estiver rodando na máquina `epicurus` da rede Linux, de qualquer máquina da rede Linux alguém pode dizer

```
telnet epicurus.linux.ime.usp.br 4444
```

e entrar no jogo.

[1]Essa solução é ruim. Além de inviabilizar a separação entre o programa e a descrição do mundo virtual, ela é inadequada para lugares que se modificam ao longo do tempo.

4.2 Seu Trabalho

O mini-servidor multiusuário disponibilizado na página da disciplina lida com algumas coisas que não foram estudadas em classe, como *threads* e comunicação via rede usando *sockets*. Se você quiser estudar e entender essas coisas, ótimo! Se não quiser, mesmo assim poderá utilizar as classes `GameServer`, `Session`, `EndOfSessionException` como estão, ou com pequenas alterações que você deverá discutir com o professor. Nesta linha de trabalho sua primeira tarefa é pegar o jogo que você escreveu no primeiro exercício programa e fazê-lo funcionar em modo multiusuário, utilizando as três classes citadas acima. Isso não deve ser difícil.

Sua tarefa seguinte é muito mais interessante: alterar o jogo para que ele explore o modo multiusuário, permitindo que os jogadores se comuniquem uns com os outros, lutem entre si, se ajudem contra um inimigo comum, etc.

5 Recomendações

As mesmas do primeiro exercício programa:

- Não saia escrevendo código sem pensar! Invista algum tempo planejando sua implementação antes de escrevê-la. Decida que classes você vai precisar, que informações cada classe precisa manter em seus campos, de que forma suas classes vão interagir umas com as outras, etc.

- Lembre sempre que todo programa deve ser organizado em funções ou métodos pequenos (30 linhas em média, uma página no máximo[2]) com objetivos muito bem definidos e que se adaptem naturalmente ao problema que se quer resolver. Todo método deve ser precedido de um comentário que diz sucintamente o que o método faz!

- Se um método ficou muito grande e obscuro, reorganize-o! Considere a possibilidade de transferir parte de suas tarefas para métodos auxiliares.

- Se uma classe parece grande e sobrecarregada de responsabilidades (métodos demais, por exemplo), considere a possibilidade de criar classes auxiliares que assumam parte das responsabilidades da classe.

Bom Trabalho!

[2]Esta é uma "regra de bolso", que não pode ser levada ao pé da letra em todas as situações. O exemplo típico de situação que escapa dessa regra é um método contendo um comando `switch` com muitos casos.