UNIVERSIDADE FEDERAL DE MATO GROSSO

CURSO DE ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA DE

PROGRAMAÇÃO ORIENTADA A OBJETOS

TRABALHO PRÁTICO

BRUNO FERNANDES DE LIMA SANTOS

THAIS CRISTINA COUTO HURTADO

CUIABÁ, MT

2016

BRUNO FERNANDES DE LIMA SANTOS

THAIS CRISTINA COUTO HURTADO

TRABALHO PRÁTICO DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Documentação referente ao trabalho prático da disciplina, solicitado pelo Prof. Fabrício Carvalho, docente da disciplina de Programação Orientada a Objetos.

CUIABÁ, MT

2016

1 OBJETIVO

O presente documento possui como objetivo aplicar os conceitos de programação orientada a objetos, aprendidos em sala de aula, no desenvolvimento de uma aplicação de jogo. Toda a documentação necessária para o entendimento do trabalho será referenciada ou mesmo "linkada" neste documento: desde os enunciados do trabalho até o código-fonte do programa, escrito em linguagem Java.

2 INTRODUÇÃO

O software disponível neste pacote desenvolve o jogo **Mecha Wars**. Este é um jogo realizado via linhas de comando, isto é, sem interface gráfica, entre dois jogadores.

O jogo consiste em uma arena de batalha e dois jogadores, cada um manipulando um robô. Cada jogador pode realizar uma jogada por turno, sequencialmente, um após o outro. As opções de ação de um jogador (robô) são: **mover** (no espaço da arena) ou **atacar**. Na ação de atacar, um robô dispara um ataque no robô adversário, decrementando pontos de vida dele. Na ação de mover-se o robô se desloca para uma outra posição na arena. Nesta arena também há, em posições aleatórias, itens especiais de três tipos: bomba, arma ou vírus. Tais itens especiais afetam o robô de formas diferentes: no caso de uma bomba, esta bomba explode no robô e pontos de vida são descontados do robô afetado; no caso de um vírus, este vírus infectará o robô por uma quantidade t de turnos, descontando uma quantia de pontos de vida durante estes turnos; caso seja uma arma, uma opção é oferecida ao jogador se ele quer trocar a arma equipada do seu robô pela arma encontrada na arena.

Para mais detalhes acerca da descrição do jogo, como as equações que descrevem as variáveis utilizadas neste jogo, coeficientes de dano, de armas, bem como os tipos de armas e robôs utilizados neste aplicativo, consulte os documentos de descrição 1 e 2.

- Clique aqui para acessar o documento 1 da descrição do trabalho.
- Clique aqui para acessar o documento 2 da descrição do trabalho.

3 DESENVOLVIMENTO

Aqui será descrita os passos para a elaboração desta aplicação. Detalhes internos acerca de métodos de cada objeto, tais como tipo de retorno, parâmetros, exceções lançadas etc, constam no **Javadoc** presente no **código-fonte** deste projeto.

3.1 Modelagem dos objetos

Em desenvolvimento de software utilizando-se de programação orientada a objetos um dos primeiros passos do programador é pensar em quais serão os **objetos** (entidades) que deverão ser projetados e quais serão as **relações** entre estes objetos.

A princípio é fácil ver que **robô** aqui deverá ser um objeto, que terá os seus métodos de atacar e mover. A **arena** também é modelada aqui como um objeto, que deve ter atributos das suas três dimensões x, y e z. Um robô ocupa uma posição P(x, y, z) em uma arena, com isto, um objeto que representasse uma coordenada cartesiana em três dimensões foi pensada. Foi criado então um objeto **ponto 3d**, que possui os atributos x, y e z, dados em números inteiros. Números inteiros foram escolhidos, tanto para as coordenadas de ponto quanto para as dimensões da arena, pois cada posição na arena deve ser discretizada: assim, uma arena cujas dimensões são 10 x 10 x 10 possui exatamente 1000 posições diferentes. Estas posições serão ocupadas pelos dois robôs e os itens especiais. **Itens especiais** também é uma classe de objetos, que deverá ser herdada pelos objetos **bomba**, **arma** e **vírus**. Cada item especial, tal como o robô, deverá conter um objeto de ponto 3d como atributo. Cada um dos três objetos possui um atributo numérico que corresponde ao coeficiente de dano que este causa a um robô. Este coeficiente é utilizado nos cálculos de dano nas fórmulas descritas nos arquivos de descrição 1 e 2.

A arena possui itens aleatórios espalhados em sua dimensão. Pensando nisso, uma coleção `Map` foi utilizada para mapear os itens especiais. A chave deste mapa é justamente o ponto 3d agregado ao item especial, enquanto seus valores serão os itens especiais que possuem o ponto da chave. Assim, cada ponto na arena pode ser ocupado por 0 ou 1 item especial, mas não mais do que isto.

As ações do jogo são dados persistentes e devem ser serializadas, isto é, seus valores deverão ser gravados em arquivos binários para que possam ser lidos posteriormente por outra aplicação. Pensando nisso, um objeto **ação** foi elaborado, onde este recebe em seu construtor referências a objetos que participam de uma ação de jogador, como mover-se ou atacar. Este objeto ação posteriormente tem seu estado salvo em arquivos utilizando-se dos objetos `ObjectOutputStream`, presentes na API do Java.

A classe principal leva o nome do jogo como nome da classe, e é a classe que chama todos os objetos e realiza toda a interação com os usuários por meio dos fluxos de dado padrão `System.out` e `System.in`.

Conforme descrito nos arquivos de descrição do trabalho, exceções específicas devem ser criadas e devem ser disparadas quando o jogador realiza um movimento imprudente com o robô. Com isso, exceções `LimitMoveException` e `LimitArenaException` foram criadas para tal.

Além destas, foi criado uma classe de `Error`: `BadConfigError` será lançado sempre que a aplicação tentar, sem sucesso, ler dados de alguns dos arquivos de texto que carregam configurações críticas, como listas de robôs e armas, e configurações importantes. Tais dados são fundamentais para o funcionamento da aplicação e, sem eles, o programa não pode seguir: é disparado então um erro em vez de uma exceção.

3.2 Fluxos de dados e interação com o usuário

Além da entrada e saída padrões do sistema, também há o carregamento de dados referentes ao jogo por meio de arquivos de texto. São utilizados neste trabalho três arquivos.

1) Arquivo "config.txt": Contém dados referentes a algumas configurações do jogo, como dimensão mínima para arena e coeficiente de preenchimento (será explicado mais adiante).

2) Arquivo "robots.txt": Contém uma listagem de todos os robôs e respectivos atributos imutáveis a este robô, como nome, HP de base (HP são os pontos de vida do robô) e armadura de base.

3) Arquivo "weapons.txt": Contém uma listagem de todas as armas e respectivos coeficientes de dano associados a cada arma.

Os arquivos incluem ainda linhas para comentários: Para adicionar um comentário basta iniciar uma linha com o caractere '#'. O leitor de arquivos do aplicativo irá ler este arquivo e ignorar as linhas comentadas. Note que estes nomes podem ser alterados pelo usuário livremente, desde que alterações no código-fonte sejam realizadas para assegurar de que a aplicação chamará o caminho destes arquivos em seu construtor (ver seção 3.3.1).

A interação com o usuário ocorre durante todo o jogo por meio da entrada padrão e saída padrão do sistema. Através de comandos digitados pelo jogador, a aplicação recebe estes dados, processa-os e os exibe na tela via saída padrão: o prompt de comandos, caso

seu sistema operacional seja o Windows, ou o terminal, caso seu sistema operacional seja o Linux.

3.3 Desenvolvimento geral da aplicação

Inclui-se aqui uma descrição literal sobre o desenvolvimento do jogo como um todo.

Vale ressaltar mais uma vez que mais detalhes sobre os métodos utilizados, e como eles funcionam, podem ser consultados através da documentação que consta no código-fonte que acompanha este documento.

3.3.1 A classe principal: MechaWars

Como foi dito, a classe `MechaWars` é a classe que invoca o jogo propriamente, ou seja, chama todos os construtores de classe dos outros objetos componentes do software, e faz o interfaceamento entre usuário e aplicação.

A classe possui alguns atributos: uma `Queue` (fila) contendo objetos do tipo `Action` (ação), um objeto do tipo `Arena`, dois objetos do tipo `Player` (isto é, os jogadores participantes do jogo) e dois `Set` (conjunto), sendo um para objetos do tipo `Robot` (robô) e outro para objetos do tipo `Weapon` (arma). Estes conjuntos recebem os registros de todos os robôs que constam nos arquivos de configuração do jogo. Uma vez que os conjuntos são definidos, os arquivos não serão mais utilizados. O construtor da classe recebe três Strings, todas para indicar o caminho de arquivos de texto: a primeira deve indicar o caminho para o arquivo que contém a listagem dos robôs, o segundo para o arquivo de armas e o terceiro deve conter para o arquivo que contém as informações adicionais.

A classe possui ainda alguns métodos, sendo apenas um deles público: `runGame()` é o método que chama todos os outros métodos privados da classe, promovendo o encapsulamento dos atributos e funcionalidades do aplicativo. Assumindo que nenhum erro de configuração aconteça, este método irá encerrar-se de duas maneiras: Ou quando um dos jogadores vencer a batalha, ou quando um dos dois jogadores digitar o comando “exit” durante a aplicação.

3.3.2 A classe que manipula ações dos jogadores: Action

Esta classe executa todas as modificações de status do jogo. O status do jogo é o estado atual em que o jogo se encontra, como posição dos robôs na arena, posição dos itens especiais, pontos de vida de cada robô etc. A classe `Action` é a responsável por, cada vez que é chamada, alterar o estado atual do jogo de alguma forma.

A classe possui como atributos referências para jogadores (que, por sua vez, possuem um atributo de robô) e referência para uma arena. Apenas um método em `Action` não é privado: `void make(String command)` é o método que deve ser chamado pelo controlador do jogo. Seu parâmetro é uma string digitada pelo usuário, que deve corresponder a um comando, a saber: **attack**, **move dx dy dz** ou **exit**. Note que dx, dy e dz são os valores em deslocamento, ou seja, ao passarmos este comando para o construtor estamos dizendo: *"mova-me dx casas no eixo X, dy casas no eixo Y e dz casas no eixo Z"*. Estes valores de deslocamento podem ser tanto positivos como negativos, ou até mesmo nulo, caso o jogador não queira fazer nenhum movimento.

A partir do comando do usuário, o método `make()` invoca outros dois métodos: `makeAttack()` e `makeMove()`. Os dois realizam, respectivamente, ação de ataque e ação de movimento.

O método de ataque é mais simples: ele chama o método `attack()` do objeto robô atacante, passando como argumento o robô alvo, e então calcula o dano. Este dano é então registrado no log de eventos daquela ação.

O método de movimento é bastante mais complexo, pois antes de realizar o movimento o método precisar fazer algumas checagens:

1. O robô está se movimentando para fora dos limites da arena? Se sim, disparar a exceção correspondente à violação da restrição da arena;
2. O robô está se movimentando além do seu limite? Se sim, disparar a exceção correspondente à violação da restrição de seu próprio limite de movimento;
3. A nova posição possui um item especial? Se sim, produzir seu efeito através do método polimórfico `effect()`;
4. A nova posição já possui o robô adversário nela? Se sim, deslocar o robô ator do movimento uma posição, de forma aleatória, porém garantindo que este não caia para fora da arena.

4 CONCLUSÃO

A funcionalidade do jogo, embora básica, tem por trás todos os conceitos de programação orientada a objetos mais fundamentais, como herança, polimorfismo e encapsulamento. Configurações diversas podem ser alteradas, bem como novas armas e novos robôs podem ser adicionados, sem precisar mexer no código-fonte da aplicação: através das técnicas de manipulação de arquivos do Java pode-se customizar o jogo à maneira do usuário.

O programa é extensível e pode ser utilizado, ainda, em projetos futuros.