Relatório Exercício Programa III - Batalha de Robôs

Fellipe Souto Sampaio *
Gervásio Protásio dos Santos Neto †
Vinícius Jorge Vendramini ‡

17 de novembro de 2013

MAC 0242 Laboratório de Programação II
Prof. Marco Dimas Gubitoso

Instituto de Matemática e Estatística - IME USP
Rua do Matão 1010
05311-970 Cidade Universitária, São Paulo - SP

*Número USP: 7990422 e-mail: fellipe.sampaio@usp.com
†Número USP: 7990996 e-mail: gervasio.neto@usp.br
‡Número USP: 7991103 e-mail: vinicius.vendramini@usp.br

1 Introdução

Esse relatório se destina a explicar a implementação do Terceiro Exercicío-Programa da disciplina de Laborátorio de Programação II.

Nesta etapa do projeto houve um refinamento das etapas anteriores através da implementação de uma interface gráfica e do processamento das chamadas ao sistema.

No tocante gráfico utilizou-se das bibliotecas padrões do java Swing e Jframe como arcabouço para criação de uma arena virtual e para representação das demais entidades que estão presente no jogo.

Na maquina virtual houve uma otimização do seu funcionamento através de certas mudanças feitas no Parser

Por fim diversas chamadas ao sistema foram implementadas e estão operantes e visualizaveis através da interface gráfica.

2 Interface Gráfica

2.1 Aspectos Estruturais

falar sobre como é dividido

2.2 Aspectos Funcionais

Explicar sucintamente como funciona elencar a estrutura de dependência das classes e falar um pouco sobre os principais métodos

3 Robôs

3.1 Battlerobot

3.2 Maquina Virtual

3.3 Syscall

3.4 Parser

Uma das principais mudanças em relação à fase anterior reside na classe Parser. Anteriormente cada time possuía um código fonte próprio, restringindo a diversidade de funcionamento dos robôs em campo. Nesta fase Parser foi alterado de forma que agora seja possível que cada robo tenha um source code próprio, aumentando substancialmente a gama de possibilidades do jogo. Para isso o sistema procura um arquivo chamado *sourceCode-x*, onde x varia de 0 até n-1 quantidade de robôs em jogo. Caso o código fonte não seja encontrado o jogo carrega no robô o código *defaultSource*, que possui um conjunto de instruções básicas definidas pelo usuário, essa decisão é tomada com intuito de possibilitar o funcionamento do jogo mesmo que nenhum código tenha sido escrito para um robô em específico. Por fim utilizou-se algumas técnicas do Java para navegação através de diretórios para centralizar os códigos fonte utilizados ao longo do jogo. O diretório padrão de leitura dos códigos é */sourceCodes*, subdiretório da pasta principal do Battlefield.

4 Battlefield

4.1 Funcionamento do jogo

O laço principal de funcionamento do jogo acontece na função *runTheGame*. Esta função pode ser dividida em três etapas:

- Execução de um ciclo de instruções da máquina.
- Processamento das requisições ao sistema feitas pelos robôs.
- Verificação se a condição de funcionamento do laço ainda é verdadeira.

No primeiro item a arena navega linearmente através do vetor de robôs deixando com que cada um execute um número ψ fixo de instruções de máquina.

A arena espera que o robô faça uma requisição ao sistema durante essas instruções, todavia se o robô executar ψ instruções sem nenhuma chamada ao sistema a arena causa uma interrupção no processamento da maquina virtual e força que esta faça uma chamada vazia, para assim permitir que outros robôs executem suas instruções evitando que o sistema fique preso em um looping infinito. As chamadas ao sistema são feitas através do método *systemCall*, no qual cada robô insere na lista requesList um pedido, modelado através do objeto *SystemRequest*.

Na segunda etapa o sistema, que ja terminou de processar todas os robos, começa a processar as requisições que foram feitas. Embora o vetor de robôs sempre seja percorrido linearmente o sistema aleatoriza o vetor de requisições, evitado com que um robô tenha vantagem sobre outro. Após a aleatorização da lista um iterador percorre-a executando cada chamada através do método *executeCall*. A arena é atualizadas graficamente ao final desta etapa.

E no fim o sistema verifica se a condição que o mantem dentro do laço do jogo ainda é verdadeira. Ao navegar linearmente através do vetor de robôs o sistema conta quantos estão inativos[1], caso o número de robôs inativos seja o mesmo de robôs em jogo o laço é quebrado e a execução do jogo termina.

4.2 Processamento das syscalls

Cada chamada ao sistema é um solicitação muito bem definida para que não exista ambiguidade. Em cada uma dessas chamadas uma ação diferente acontece, como é explicitado a seguir:

- WLK - Anda pela arena utilizando uma das seis direções possíveis (leste,oeste, nordeste,sudeste,sudoeste e noroeste).
- FIRE - Atira na direção solicitada.
- BOMB - Planta uma bomba na direção solicitada e a cada nova rodada o timer é decrementado.
- LOOK - Olha para uma direção solicitada e pergunta o que existe ai, podendo ser cristal, bomba, inimigo, amigo ou nada.
- ASK - Faz perguntas ao sistema sobre a quantidade de energia que um robô tem, sua quantidade de cristais, a distância em linha reta até a base inimiga e o mesmo calculo de distância até a sua base.

4.3 Atualização da arena

Explicar como funciona a atualização da arena

5 Compilação e funcionamento

Para compilar utilize o makefile que está dentro da pasta com o comando *make* via terminal. A classe principal é Battlefield.java e para executar o programa é necessário escrever os códigos fonte para os robôs e guardar na pasta *sourceCodes*

[1] entende-se *inativo* como um robô que chegou a o fim da sua execução (comando END) ou que foi destruído durante a batalha.

com o nome apropriado, caso contrário pode-se utilizar o *defaultCode* como código default para os robôs. A parte final dos aspectos do jogo, como carregar o cristal até a base inimiga e poder visualizar o dano causado em um robô por um tiro ou explosão, ainda serão implementados na fase seguinte, listamos a seguir as mudanças programadas para a próxima fase

5.1 Tarefas futuras

Implementação das ultimas instruções do sistema.

- Melhora em certas animações.
- Limpeza do código quanto a disposição das classes.
- Redução de redundâncias e pleonasmos no código.
- Separação dos arquivos quanto ao seu tipo através de pastas.
- Compactação em um arquivo jar.

->Resto das chamadas ->Limpeza do codigo ->Melhoridas na parte gráfica –>Background –>Tamanho fixo da janela –>Contorno dos hexagonos –>Melhorar os sprites