Relatório - EP Fase 3
Laboratório de Programação 2

2 de dezembro de 2013

1 Integrantes

- Victor Sanches Portella - Nº USP: 7991152
- Lucas Dário - Nº USP: 7990940
- Ruan Costa - Nº USP: 7990929

2 Introdução

Nesta fase foram implementados a parte visual do jogo, algumas instruções adicionais para melhor desenvolvimento dos robos através do nosso *Assembly*, e foram impelmentados alguns melhoramentos de algumas instruções. Este relatório tem como objetivo dar uma visão geral de todas essas mudanças.

3 Interface gráfica

Toda a parte da interface gráfica está implementada no arquivo **MapaVisual.java**. Neste arquivo há diversas classes e estruturas que controlam a parte gráfica do jogo. Uma breve explicação sobre algumas dessas estruturas:

- **classe Célula**: Representa um hexagono do do campo. Seus atributos são o centro (x, y), o raio, a imagem dessa célula e o hexagono(polígono). O construtor constrói o hexagono de maneira similar à disponibilizada no paca. A função draw desenha uma célula
- **classe Campo**: Extensão de JPanel. Tem como atributos uma matriz de células, o mapa, um boolean que diz se o jogo acabou e as

imagens usadas no jogo. O construtor seta o mapa e cria um campo com hexagonos de lado L.

A função **criaCampo()** carrega as imagens e varre a matriz de células configurando suas respectivas coordenadas de centro e a imagem de fundo de cada célula, de acordo com o Terreno que aquela célula representa.

A função **paintComponent(Graphics g)** é um override da função de mesmo nome implementada em JPanel. Percorre o campo desenhando os hexagonos com suas respectivas imagens. Caso o jogo já tenha acabado, também exibe uma imagem de game over.

A função **desenhaElemento(...)** desenha o elemento passado em cima da imagem de fundo da célula de coordenadas passadas. É usada para desenhar robos e bases.

A função **getScaledImage(...)** converte a imagem passada por argumento para uma outra com as dimensões passadas também por argumento.

- **classe Tela**: Extensão de JFrame. O construtor dá o título da janela, seta seu tamanho, adiciona um listener e adiciona o JPanel campo.

- **classe MapaVisual**: Classe que interfaceia a comunicação entre a estrutura do jogo e a parte gráfica. Possui como atributos o mapa, os valores estáticos de largura, altura (da janela) e lado do hexagono (em pixels), um objeto Tela e um objeot Campo. O construtor apenas seta os atributos.

  A função **atualiza()** repinta a tela(que é uma janela).

  A função **abreJanela()** seta os atributos campo e tela e cria um Runnable, tal que o método sobrecarregado run() apenas faz com que a tela(janela), seja visível.

  A função **gameOver()** seta para true o boolean gameOver do atributo campo e assim encerra o jogo.

4 Arquivos de mapas

Nessa fase, para uma edição e criação facilitada de mapas, fizemos com que os mapas fossem gerados a partir de um aruqivo de entrada de texto. Exemplo:

```
6 6
1 1 1 1 1 1
1 -1 0 0 0 1
1 0 1 1 0 1
1 0 1 1 0 1
1 0 0 0 0 1
1 2 2 2 2 1
```

Na primeira linha deve ter um par de números representando o número de **linhas** e de **colunas**, respectivamente. Em seguida, cada linha de números representa uma linha do mapa, onde cada número representa um terreno. Legenda dos números:

Número	Terreno
-N	Base do time de Id N
0	Liso
1	Rugoso
2	Água
3	Depósito de cristais
Default	Liso

5 Novas instruções

- **MOD**: Desempilha dois números da pilha, e empilha o resto da divisão do segundo pelo primeiro.
- **GETPOS**: Desempilha um Terreno da pilha, e empilha a posição do mesmo (obejto **Posicao**) do mesmo como resposta.
- **GETX**: Desempilha um objeto Posicao da pilha, e empilha sua coordenada x como resposta.
- **GETY**: Desempilha um objeto Posicao da pilha, e empilha sua coordenada y como resposta.
- **TIMEID**: Desempilha um time da pilha de dados, e empilha o número que representa aquela time (o ID do time).
- **GETBASE**: Desempilha um time da pilha de dados, e empilha o Terreno do tipo Base daquele time.

- **ISA**: Desempilha um número e um empilhável. Esse comando irá empilhar 1 caso o objeto desempilhado seja do tipo representado por aquele número (Vide tabela abaixo), e 0 c.c.

Número	Tipo
0	Numero
1	Terreno
2	Liso
3	Robo
4	Deposito
5	Rugoso

- **GETROBO**: Desempilha um número, representando uma direção, e empilha o Robo que está naquele terreno, e empilha 0 caso não haja Robo.
- **GETTIME**: Desempilha um número, e é empilhado como resposta o time correspondente a aquele número. Empilha 0 caso não existe tal time.
- **MYTIME**: Empilha o time do próprio robo.
- **LOOK**: Desempilha um número representando uma direção, e empilha o terreno daquela posição. Empilha 0 caso não exista nada naquela direção.

Além disso, os comandos de chamada ao sistema: **WALK**, **ATK**, **COLLECT** e **DROP** usam como argumento de direção um número no topo da pilha da máquina virtual. Outra modificação foi que, para evitar que o robô se "teletransportasse", já que a cada iteração cada robo executa 5 comandos de máquina, fizemos com que o robô acabe sua rodada ao tentar se mover. Caso isso não acontecesse, seria possível que ele conseguisse andar duas vezes em uma só iteração, dando a impressão que o robô pudesse "voar".

6 Tradutor em Perl

Nessa fase, o nosso tradutor em Perl recebe o mapa, diversos robos como argumento e é possível separá-los em diversos times. Apesar do número de times ser indefinido, nossa interface gráfica suporta somente 2 times. O mapa deve ser estar na pasta **data/map**. Os robos na pasta **test** e com extensão .asm. Sintaxe:

```
$>perl bin/montador.pl mapa3.txt 1 robo1_time1 robo2_time1 2 robo1_time2
  robo2_time2  ... n robo1_timen
```

Com isso, será gerado um novo arquivo Main.java na pasta src.

7 Modo de compilação

Para compilar, execute o seguinte comando:

```
$> ant
```

Com isso, o jogo será compilado, e será gerado um arquivo .jar localizado na pasta dist/. Assim, parara executar o jogo, a sintaxe é:

```
$> java -jar dist/EP3.jar
```