Relatório - EP Fase 2
Laboratório de Programação 2

29 de outubro de 2013

1 Integrantes

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

2 Introdução

Neste relatório temos como objetivo dar uma explicação concisa sobre a organização das classes no funcionamento do programa atualmente. Para isso, explicaremos o funcionamento do programa inteiro, e iremos nos aprofundar nos assuntos que forem necessários.

3 Funcionamento

Nessa fase o EP ainda não têm o funcionamento de um jogo em si, mas já é possível ver diversos robôs agindo em paralelo. Para isso, o script em Perl[1] traduz diversos programas para um arquivo chamado **Main.java**.

Todos os robôs traduzidos serão colocados no mesmo time. Ao rodar o programa, os programas de cada robô serão executados até o fim. Quando todos os robos tiverem encerrado sua execução, o programa será finalizado.

Cada robô imprime as instruções que executou para que assim possamos acompanhá-las.

[1] Explicação do mesmo feita mais a frente

4 Classes

4.1 Classe Arena

Essa classe representa a Arena em si. Nela, temos os seguintes atributos:

- **Mapa**: Objeto do tipo **Mapa** que representa o mapa em si. Nesse objeto existe uma matriz de Terrenos, cada um contendo informações como robôs naquele terreno, assim como cristais, bases, etc. Melhores explicações sobre esse objeto serão dadas mais a frente.
- **List<Robos>**: Objeto do tipo **List** usado como uma lista de objetos do tipo **Robo**. Essa lista contêm todos os robôs em jogo.
- **Tempo**: Uma variável inteira representando o número de rodadas já passadas. O valor inicial dessa variável é 0 e, a cada rodada, é incrementada em 1.

Principais métodos dessa classe:

- **Arena(Mapa)**: Construtor da classe. Recebe uma mapa, e retorna uma arena vazia, ou seja, com nenhum robo, o tempo é iniciado com 0, e o mapa passado é definido como o mapa da arena.
- **void atualiza()**: Função que faz a Arena executar uma rodada. Atualmente, essa função percorre a lista de robôs de forma aleatória, executando em cada robô uma instrução de máquina. Em caso de conflito de instruções, tem preferência o robô que for executado primeiro naquela iteração.
- **void sistema(Operacao)**: Função chamada pelos robôs para fazerem chamadas ao sistema. Toda vez que os robôs tentam executar uma ação que influencia seu exterior, ele só pode fazê-la chamando esse método. O objeto do tipo **Operacao** recebido descreve qual operação está sendo executada, e qual objeto Programável está tentando executar aquela ação. Dependendo da operação, esse método mexe na pilha do robo, e executa as ações pedidas, caso possível.
- **void insereExercito(Programa[ ],Posicao[ ],int)**: Função para inserir um exército de robôs na Arena. Para isso, recebe três argumentos:
  - **Programa[ ]**: Um vetor de objetos do tipo Programa. Para cada programa desse vetor será criado um robô a ser inserido na Arena.

+ **Posicao[ ]**: Um vetor com posições x,y indicando onde cada robô criado deve ser posicionado, seguindo a ordem do vetor de programas.
+ **int time**: Um inteiro que indica a qual time esse exército pertencerá.

4.2 Classe Maquina

Um objeto da classe Máquina possui os seguintes atributos:

- **dados** Objeto do tipo Pilha. É a pilha de dados da máquina.
- **mem** Vetor de Empilháveis usado para o robô armazenar constantes.
- **prog** Objeto do tipo Programa. É o vetor de comandos que aquela máquina executará.
- **vars** Objeto do tipo variáveis. Guarda o estado atual das variáveis de determinada máquina.
- **arena** Referência para a Arena para a realização de chamadas ao sistema.
- **index** Inteiro que guarda o índice do próximo comando do programa a ser executado.
- **obj** Objeto do tipo Programável. A princípio é apenas uma referência para o robô que contém aquela máquina.

Principais métodos:

- **Construtor**: Recebe o sistema (arena) com o qual a máquina vai se comunicar e o Programável que a possuirá. Mais uma vez, nesse ep, Programáveis são sempre robôs. Inicializa mem com tamanho 10 e index setado em 0(zero).
- **int executaProx()**: executa o comando cmd da posição index do programa. Para isso chama a função executaCmd(cmd), caso o programa não tenha chegado ao fim.
- **int executaCmd(Comando)**: Executa o comando passado como argumento. Verifica qual o código do comando e realiza a devida ação. Retorna o índice do próximo comando a ser lido. O sistema é chamado por meio da função sistema(), caso o comando passado precise de permissão da arena(i.e, se for ATK, COLLECT, DROP ou WALK).

- **void sistema(Comando)**: Pede para o sistema executar o Comando cmd.

  As outras funções são, em geral, auxiliares e estão bem explicadas nos comentários.

4.3 Classe Mapa

Essa classe é usada para representar o mapa que, em linhas gerais, é uma matriz hexagonal de objetos do tipo **Terreno**. Nessa classe, temos 3 atributos: **altura**,**largura** e uma matriz de objetos do tipo **Terreno**.

Os métodos dessa classe estão bem explicados já no código.

Para transformar a matriz normal em uma matriz hexagonal, tratamos ela assim como mostrado na imagem a seguir. Assim, apesar da matriz ser a usual, com a altura e largura definidas no atributo, usamos ela como uma matriz hexagonal através dos métodos da classe.

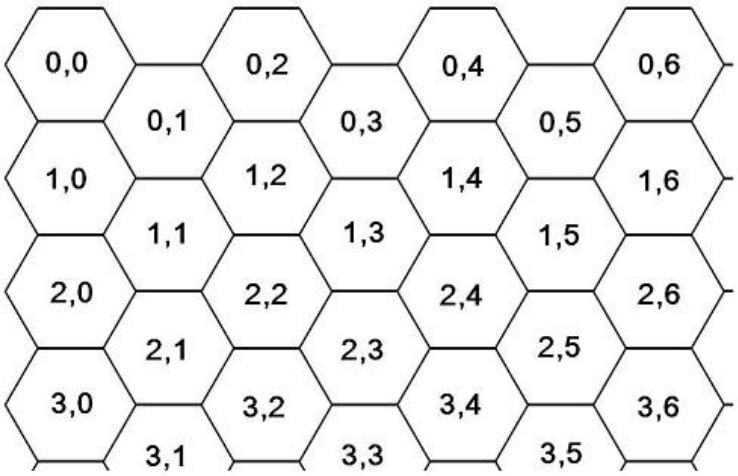

Figura 1: Imagem mostrando o modo de enumeração dos terrenos

5 Interfaces

Aqui daremos uma breve explicação sobre as interfaces presentes no projeto:

- **Programável**: Toda classe que implementa essa interface pode ser colocada dentro de um obejto do tipo **Maquina**. Ou seja, quem im-

plementa essa interface é todo aquele objeto que conterá uma máquina virtual. Por enquanto, só os robôs são programáveis, mas no futuro alguns terrenos poderão ser programáveis.

- **Empilhável**: Toda classe que implementa essa interface é tratada como um *Empilhável*, ou seja, objetos que a máquina virtual manipula em sua pilha de execução e em sua memória.

6 Novas instruções

Nessa fase, criamos novas instruções. Breve explicação sobre elas:

- **ALO**: Aloca uma variável local, e empilha um objeto do tipo **Endereço**.
- **GET**: Desempilha um **Endereço**, e empilha o valor contido nesse Endereço.
- **SET**: Desempilha dois elementos da pilha. Se o segundo for do tipo **Endereço**, é colocado o valor do primeiro argumento desempilhado na variavel localizada no endereço desempilhado.
- **Chamadas ao sistema**[2]: Atualmente, todas as chamadas ao sistema devem ser acompanhadas de um argumento indicando a direção. As direções podem ser: 1 - Cima, 2 - Direita superior, 3 - Direita inferior, 4 - Baixo, 5 - Esquerda inferior, 6 - Esquerda superior. As instruções implementadas são:
  - **WALK**: O robô tenta andar para a posição indicada
  - **COLLECT**: O robô tenta coletar cristais da posição indicada
  - **DROP**: O robô tenta deixar o cristal na posição desejada. Se a posição não for uma base, o cristal é retornado para seu ponto inicial.
  - **ATK**: O robô tenta atacar outro robo localizado na posição indicada.

[2]Atualmente, só o WALK chegou a ser testado e de fato implementado. Os outros comandos, apesar de implementados, não são possíveis de serem testados, já que o mapa é todod de terreno liso sem cristais e todos os robôs são do mesmo time.

7 Tradutor em Perl

Para efeito de teste, adaptamos o montador em perl para gerar um arquivo Java que, ao ser executado, irá transformar os programas em linguagem de máquina para robôs e irá executar iterações da Arena até que não exista mais nenhum outro robô para ser executado.

Assim, para gerar esse arquivo em Java, chamado **Main.java**, execute a seguinte linha de comando:

```
$>./montador.pl arquivo1 arquivo2 ... arquivoN
```

8 Modo de compilação

Para compilar, execute o seguinte comando:

```
$> javac *.java
```

Caso você tenha gerado o arquivo Main.java com o script em perl, para executar o programa, execute:

```
$> java Main
```