Exercício-programa II
Laboratório de Programação II (MAC 242)

Nomes

Danilo Novais (NUSP:7990759)
Fernando Camara Bizzotto (NUSP: 7991211)
Ricardo Mikio Morita (NUSP: 5412562)

1.Introdução

"Part of the inhumanity of the computer is that,
once it is competently programmed
and working smoothly,
it is completely honest."
Isaac Asimov

Nesta segunda parte do exercício-programa estaremos desenvolvendo a parte em Java do projeto de criação de uma Arena habitada por exércitos formados por robôs virtuais. Foi transcrita a máquina virtual que foi construída no primeiro EP em Perl para uma nova máquina virtual em Java e com Princípios de Orientação a Objetos (POO). As seções a seguir visam dar mais informações sobre o que foi feito no projeto.

1.1 Arquivos inclusos com suas funções

Esta primeira parte serve como uma lista daquilo que cada arquivo contém. Talvez seja mais interessante considerar esta seção como um índice remissivo do projeto para seus eventuais

- **Arena.java**
  - *public* Arena ( )
  - *public* void insereExercito(instrucao[ ] programa)
  - *public* void atualiza( )
  - *public* boolean listaRobosVazia( )
  - *public* int sistema(Operacao op)
- **Base.java**
- **Config.java**
- **Empilhavel.java**
- **Instrucao.java**
  - *public* String getLabel( )
  - *public* String getInstrucao( )
  - *public* Empilhavel getOperando( )
  - *public* void setLabel(String label)
  - *public* void setInstrucao(String instrucao)
  - *public* void setOperando(Object operando)
- **Maquinavirtual.java**
  - *public* MaquinaVirtual(Instrucao[ ] programa)
  - *public* Boolean execucaoFinalizada( )
  - *public* void executaInstrucao( )
  - private Instrucao retornaInstrucao( )
- **Numero.java**
  - *public* numero( )
  - *public* numero (int num)
  - *public* int getNum( )
  - *public* void setNum(int num)
- **Operacao.java**
  - *public* void setY(int y)
  - *public* int getY()
  - *public* void setX(int x)
  - *public* int getX()
  - *public* void setAcao(String acao)
  - *public* String getAcao()
- **Pilha.java**
  - *public* Pilha( )
  - *public* void Empilha(Empilhavel obj)
  - *public* Empilhavel Desempilha( )
  - *public* void Dup( )
  - *public* void Descarta( )
  - *public* void Inverte( )

    - *public* Empilhavel Consulta( )
- **Plano.java**
- **Programa.java**
    - *public* Instrucao[ ] getPrograma(Arena mapa)

- **RepositorioCristais.java**
- **Robo.java**
    - *public* void setPrograma(Instrucao[ ] programa)
    - *public* MaquinaVirtual getMaq( )
    - *public* String getNome( )
    - *public* void setNome(String nome)
- **Rugoso.java**
- **Terreno.java**
    - *public* Boolean terrenoOcupado( )
    - *public* void ocupaTerreno()
- **Teste.java**
    - *public* static void main(String[ ] args)
- **Texto.java**
    - *public* Texto(String str)
    - *public* String getString()
    - *public* void setString(String str)
- **Montador.pl**
- **README.pdf**

1.2. Explanação sobre os arquivos

- **Arena.java**
    - Contém dados sobre a arena, a qual foi implementada como um objeto do tipo *Empilhável*.

- **Base.java**
    - Descreve um objeto do tipo *Terreno*.

- **Config.java**
    - Contém o nome dos robôs que podem ser usados na arena, retornando um array de strings com o nome deles.

- **Empilhavel.java**
    - Seguindo os desígnios do professor, esta é a Interface que é usada para se colocar itens na pilha. Todos os itens da pilha são, portanto, “Empilháveis”.

- **Instrucao.java**
    - Usado para lidar com a manipulação das instruções em *Assembly*, lida com labels, instruções e operandos, armazenando-os e retornando-os conforme necessidade.

- **MaquinaVirtual.java**
    - Lê o vetor de instruções em *Assembly* de cada robô, executando-as conforme as chamadas da Arena e parando quando for encontrada a instrução de parada END.

- **Numero.java**
    - Lida com objetos do tipo *Numero*, o qual é um tipo de Empilhável.

- **Operacao.java**
    - Usado quando se faz uma chamada de sistema da Máquina Virtual ao sistema. Também é um Empilhável.

- **Pilha.java**
    - Implementa as funções de manipulação da pilha. Além das funções básicas de empilhar, desempilhar, consultar o topo e descartar o topo, também pode duplicar o topo e inverter a posição dos dois objetos no topo da pilha.
    - Apenas objetos do tipo Empilhável podem entrar na pilha.

- **Plano.java**
    - Descreve um objeto do tipo *Terreno*.

- **Programa.java**
    - Nome padrão criada pelo montador em Perl (Ver mais abaixo Montador.pl), contém instruções que podem ser usadas pelos robôs.

- **RepositorioCristais.java**
    - Caracteriza um terreno do tipo que está armazenando um Cristal.

- **Robo.java**
    - Caracteriza cada robô do ambiente virtual. Cada robô possui um nome e uma máquina virtual individual.

- **Rugoso.java**
    - Caracteriza um terreno do tipo rugoso.

- **Terreno.java**
    - Descreve características básicas para um tipo de terreno.

- **Teste.java**
    - Arquivo de teste com a função main(), exemplifica uma forma de se criar uma arena.

- **Texto.java**
    - Lida com objetos do tipo *Texto*, o qual é um tipo de Empilhável.

- **Montador.pl**

    - Transcreve um arquivo escrito em Assembly para uma arquivo “Programa.java”, o qual é usado pela máquina virtual do projeto. Este arquivo visa facilitar a alimentação dos robôs com outros códigos em linguagem de máquina.

- **README.pdf**
    - Este documento, o qual inclui informações sobre o projeto em si.

2. Ideias implementadas

Seguindo a orientação dada durante as aulas e no enunciado do EP, temos várias entidades que compõem o projeto. De modo geral, temos o objeto *Arena*, responsável pelo mapa e gerenciamento mundo. Ele é atualizado conforme as entidades *Robôs* interagem com a *Arena*. Cada *Robô* age conforme um programa inserido nele, o qual é interpretado por uma máquina virtual inerente à cada *Robô.*

O programa que o Robô interpreta é uma adaptação do Assembly, com uma chamada adicional denominada **SYS**, a qual é usada para a interação do robô com a arena com instruções como **mover, atacar, coletarCristal** e eventuais outras que podem ser definidas adiantes. Importante notar que o robô, ao usar esta função, recebe uma confirmação da *Arena* para saber se esta ação foi bem-sucedida ou não. Assim, a *Arena* é responsável pela resolução de conflitos nas ordens dos robôs e também por impedir eventuais contradições nas ordens dos robôs.

3. Conclusão

O projeto ainda está em desenvolvimento, com muitas idéias possíveis de serem aplicadas. Várias partes poderão ser reescritas e, portanto, uma decisão definitiva ainda não é alcançável. Entretanto, neste trabalho pudemos ter contato com uma linguagem orientada à objetos, além de desenvolver o arcabouço do Mundo Virtual. Nas próximas etapas uma soldificação do projeto virá à tona, ou assim se espera.