MAC 216
Relatório da segunda parte do projeto

César Gasparini Fernandes (10297630)
Artur M. R. dos Santos (10297734)
Daniel Silva Nunes (10297612)

22/10/2017

1 Visão geral

Na segunda fase do projeto, implementamos o código em arena.c, juntamente com o arena.h, que representam a arena em que as batalhas entre os exércitos de robôs ocorrerão no jogo. Ela possui as funções de sistema, criação e destruição de um robô, atualização, coleção dos cristais, ataque, encerramento do jogo, movimentação e, claro, a de inicialização da arena. Portanto, na nossa implementação, a arena que gerencia todo o jogo, executando as principais funções para que o jogo possa ocorrer.

A movimentação foi feita usando o grid hexagonal, de forma que valores determinam para qual local o robô se movimenta (descrito no código de arena.c).

A função de ataque possui somente uma opção de ataque, mas futuramente iremos implementar ataques diferenciados para os robôs.

Para a segunda fase do projeto, o sistema está elaborado de modo a administrar as ações de uma máquina no jogo, possuindo apenas a estrutura geral de um escalonador, mas que ainda não lida com múltiplos robôs e exércitos.

A função sistema, a principal do programa, gerencia todo o cenário do jogo, sendo a única que manipula a pilha de forma direta. Ademais, é necessário uma requisição ao sistema para que os robôs possam executar seus pedidos, assim, evitamos possíveis ações indesejadas e trapaças (cheats).

Os arquivos da 1 a fase necessitaram algumas alterações:

- O montador foi adaptado de modo a compor o array **prog** no código gerado com o formato correto do instrução, que agora conta com um **OpCode** e um **Operando**. Operandos possuem um **tipo** e um valor. (Ver exemplo do montador abaixo)

- O maq.c, que possui as instruções, recebeu cases adicionais para tratar as interações com a arena. Por consequência, a struct **Máquina**, em maq.h, recebeu alterações em sua struct. Os valores que estão dentro das structs 'Célula' e 'Robô' foram acessados de maneira diferente da implementação anterior, pois agora acessamos valores destas.

- As instruções em **instr.h** foram ampliadas, como dito anteriormente, e sua estrutura foi levemente alterada. Foram especificados novos tipos, direções para movimentação na arena, e o operando (OPERANDO) recebeu novos valores que foram implementados por meio de uma struct e uma union. Uma observação: a union foi criada porque temos o intuito de criar um 'box' na próxima fase do projeto (na parte gráfica), e portanto, poderemos acessar os valores que desejamos apenas 'chamando' essa union. Maiores detalhes de implementação estão detalhados no código do programa.

2 Testes

2.1 Montador

O seguinte código em "linguagem de máquina" no arquivo **tprog** foi dado na entrada do montador pela entrada padrão e gerou o seguinte código em linguagem C no arquivo **tprog.c** como está mostrado a seguir:

Código em tprog:

```
PUSH 1
DUP
ADD
PRN          #imprimir 2
PUSH 5
PUSH 2
SUB
PRN         #imprimir 3
PUSH 4
DUP
MUL
PRN         #imprimir 16
PUSH 10
DUP
DIV
PRN        #imprimir 1
END
```

Código em tprog.c:

```
#include <stdio.h>
#include "maq.h"
INSTR prog[] = {
{PUSH, NUM, {1}},
{DUP, NONE, {0}},
{ADD, NONE, {0}},
{PRN, NONE, {0}},
{PUSH, NUM, {5}},
{PUSH, NUM, {2}},
```

```
{SUB, NONE, {0}},
{PRN, NONE, {0}},
{PUSH, NUM, {4}},
{DUP, NONE, {0}},
{MUL, NONE, {0}},
{PRN, NONE, {0}},
{PUSH, NUM, {10}},
{DUP, NONE, {0}},
{DIV, NONE, {0}},
{PRN, NONE, {0}},
{END, NONE, {0}},
};

int main(int ac, char **av) {
CriaArena(20, 2, 10, 5);
        Maquina *maq = cria_maquina(prog, 0, 0, 100, 0, 1);
        exec_maquina(maq, 1000);
        destroi_maquina(maq);
        return 0;
}
```

O código em tprog foi feito a fim de testar o funcionamento das novas funções que realizam operações aritméticas. O resultado obtido a partir da execução do arquivo "arena", executável do tprog.c, foi:

```
2
3
16
1
```

2.2 Chamadas ao sistema

O jogador faz chamadas ao sistema usando principalmente cinco das funções implementadas em maq.c. São elas:

- MOV *Dir*: Solicitação para mover-se na arena para uma nova célula, indica pelo argumento Dir, que pode assumir os valores NORTH, SOUTH, NORTHEAST, SOUTHEAST, NORTHWEST, SOUTHWEST ou CURRENT.
- ATK *Dir*: Ataca a célula vizinha ou a própria célula indicada pela direção do argumento Dir, com comportamento similar ao da função MOV.
- CLT *Dir*: Coleta cristais, se existirem, na célula vizinha indicada pelo argumento Dir, com comportamento similar ao das funções anteriores.
- INF: Solicita ao sistema informações sobre a célula em que ele está. O sistema então empilha um objeto do tipo **Celula** na pilha de dados da máquina.

- DEP *Dir*: Deposita cristais na célula indicada por Dir.

  Executando o código a seguir:

```
MOV NORTH
PRN
ATK NORTH
PRN
INF
ATR 1
PRN
CLT SOUTH
PRN
DEP SOUTHEAST
PRN
END
```

  Obteve-se a seguinte saída:

```
true
false
0
false
false
```

  Que significa, de acordo com cada linha, que:

  - O robô conseguiu se mover para a célula norte
  - Não conseguiu atacar a célula ao norte de sua posição (pois ela está vazia, só há um robô na arena por enquanto)
  - A informação pedida sobre célula atual, o atributo 1 (que é se está vazia ou não), retornou 0 (falso, mas **vazia** é int na structure Celula)
  - Tentou coletar cristais na célula sul mas não conseguiu, pois não havia cristais lá, então retornou falso.
  - Tentou depositar seu cristais na célula a sudeste de sua posição, mas não conseguiu porque não possui cristais.