

# Projeto de Laboratório de Programação II - Fase 4

Karina Suemi, Vinícius Silva, Renato Cordeiro

## ..... MANUAL DO USUÁRIO .....

## JOGO

O jogo consiste em programar uma série de robôs para batalharem, num estilo de RTS 2x2. Para tanto, os robôs devem ter suas ações programadas. Eles irão executá-las até que o jogo acabe ou sejam destruídos.

Nesta fase do desenvolvimento, a programação deve ser feita em uma linguagem desenvolvida especificamente para esse jogo.

Os programas devem ser criados com extensão \*.pos\*. Exemplos estão disponíveis no diretório 'test/' junto ao código-fonte.

## INSTALAÇÃO

### – 1º PASSO –

Antes de mais nada, verifique se seu computador possui o Ant instalado, caso não possua, faça o seguinte:

1. Abra o terminal
2. Digite: `sudo apt-get install ant`

### – 2º PASSO –

Instalação do ivy:

1. Abra o terminal
2. Entre na pasta (MAC0242-PROJECT)
3. Digite: `sudo bash install_ivy.bash` para uma instalação de sistema ou `bash install_ivy.bash` para a instalação de usuário.

### – 3º PASSO –

Para compilar o jogo:

1. Ainda com o terminal aberto
2. Entre na pasta onde está localizado o arquivo do jogo (MAC0242-PROJECT)
3. Digite `ant`

### – 4º PASSO –

E para iniciar o jogo:

1. Continuar no terminal
2. Digite: `java -jar dist/MAC0242-Project.jar programa1.pos programa2.pos programa3.pos`

Observação: Na pasta "behaviors" você pode encontrar exemplos de programas de robô para testar. Eles são "Carrier.pos" e "Protector.pos".

## Usando a barra de Menu

Os botões da lateral direita da tela têm as seguintes funções:

- **MiniMap** : Habilita/Desabilita a janela do mini mapa.
- **Add Robot** : Ao clicar nesse botão um robô é criado.  
Primeiramente ele pergunta o nome do robô ser criado.  
Depois, caso o editor de texto esteja vazio ou desabilitado, ele pede um nome de programa já criado, para que ele possa comandar o robô.  
Caso o contrário, ele irá carregar o programa que foi digitado no editor de texto e salvá-lo na pasta user/ .
- **Editor** : Habilita/Desabilita a janela do editor de texto.
- **Clear** : Limpa o editor de texto.
- **Exit** : Sair do jogo.
- **Exit** : Clicando nos robôs, abre-se a janela do editor de texto com o programa do respectivo robô clicado.

## Opções

Você pode escolher várias formas de jogar A Batalha de Robôs, sendo que cada uma delas envolve um tipo de cenário diferente. Entre eles temos:

- Artical
- Tropical
- Desertic
- Continental

Para jogar cada um deles, basta iniciar o programa do seguinte modo:

- Artical ->

```
java -jar dist/MAC0242-Project.jar programa1 programa2 programa3 --artical
```

- Continental ->

```
java -jar dist/MAC0242-Project.jar programa1 programa2 programa3 --continental
```

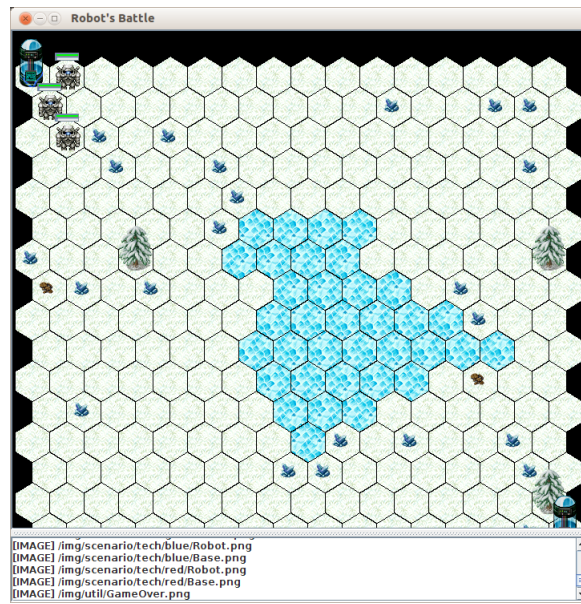
- Desertic ->

```
java -jar dist/MAC0242-Project.jar programa1 programa2 programa3 --desertic
```

- Tropical ->

```
java -jar dist/MAC0242-Project.jar programa1 programa2 programa3 --tropical
```

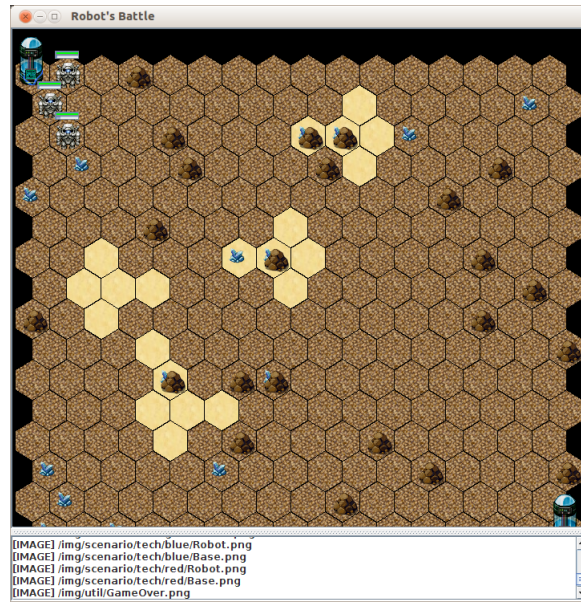
exemplos:



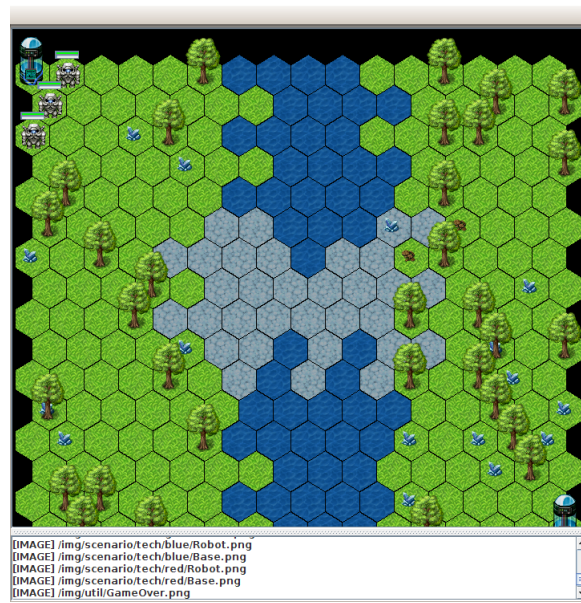
**Figura 1:** *Artical*



**Figura 2:** *Continental*



**Figura 3:** *Desertic*



**Figura 4:** *Tropical*

## Utilitários

- Se quiser modificar o tamanho da arena, coloque `-large` ao final do comando que inicia o programa:

```
$ java -jar dist/MAC0242-Project.jar programa1 programa2 programa3 -artical -large
```

- E caso queira executar o programa em modo Debugger, digite:

```
$ java -jar dist/MAC0242-Project.jar programa_jogador_1 programa_jogador_2 -v
```

# Guia de Linguagem

Se não quiser utilizar os programas já feitos que estão sendo disponibilizados na pasta "test", você pode programar seus próprios robôs utilizando uma linguagem um tanto quanto simples que é baseada no estilo das linguagens de programação mais populares.

Para isso, aprenda como essa linguagem funciona:

- **Final de um comando :**

No fim de um comando, deve-se colocar ; .

Ex.: `variavel = 2 ;`

- **Comentários :**

Uma linha de comentário é seguida de `//`.

Ex.:

`// Esse é o comentário!`

- **Declaração de variáveis :**

Para uma variável ser inicializada, deve-se sempre colocar um `my`, para variáveis locais ou `our`, para variáveis globais, antes de seu nome.

Ex.: `my variavel;`

Ex.: `our variavel = 2;`

- **Atribuição :**

Quando queremos atribuir algum valor a determinada variável, basta fazer usar `=`. Também pode-se atribuir um valor a uma variável em sua declaração.

Ex.: `variável = 42;`

- **Imprimir - PRINT :**

Para imprimir, é necessário usar o comando `print` e colocar o que se deseja imprimir entre parênteses. Você pode imprimir uma "String" ou o valor de uma variável.

Ex.: `print(variavel) ;`

Ex.: `print("Hello World!") ;`

- **Imprimir - SAY :**

Para imprimir, é necessário usar o comando `say` e colocar o que se deseja imprimir entre parênteses. Você pode imprimir uma "String" ou o valor de uma variável. A diferença entre SAY e PRINT é que SAY pula uma linha no fim e PRINT não.

Ex.: `say(variavel) ;`

Ex.: `say("Hello World!") ;`



- **IF | ELSE | ELSIF :**

Para uma condição inicial, usamos `if`, para uma segunda condição, caso a primeira não ocorra, usamos `elsif` e quando nenhuma das ações ocorra usamos `else`.

Ex.:

```
if(a>b)
{
    say("a é maior");
}
elsif(a == b)
{
    say("a é igual a b");
}
else
{
    say("a é menor a b");
}
```

- **WHILE :**

O `while` é utilizado normalmente, quando se deseja ter um laço, o usamos com a condição entre parêntesis.

Ex.:

```
while(a<5)
{
    say("Hello!!");    a = a+1;
}
```

- **BREAK :**

`Break` é um auxiliar do `While` usado quando se deseja parar o loop.

Ex.:

```
while(k<5)
{
    if(k == 1)
    {
        break ;
    }
}
```

- **CONTINUE :**

Continue também é um auxiliar do while que é usado quando se deseja pular tudo o que vem depois do continue e continuar com o laço.

Ex.:

```
while(k<5)
{
    if(k == 1)
    {
        break ;
    }
}
```

- **FUNÇÕES :**

Nas linguagens de programação usuais, vemos a presença de funções. Não sendo diferente delas, essa linguagem também utiliza funções para facilitar o modo de programação. Porém possui as seguintes restrições:

1. Para declarar a função, usamos `def`, tipo de retorno, nome da função e (tipo de parâmetros)

Ex.:

```
def number square(number);
```

2. Para chamá-la usamos o nome da função e seus parâmetros entre parênteses, em ordem.

Ex.:

```
hSquared = square(catA);
```

3. Para contruí-las basta usar o tipo de retorno + o nome da função e entre parênteses os nomes dos parâmetros acompanhados de seus respectivos tipos. Tudo isso seguido da ação que ela irá realizar entre chaves .

Não esquecendo de dar `return` nas funções que retornam algum tipo de dado.

Ex.:

```
number square(number x)
{
    my j = x;
    return x*j;
}
```

- **toCOORD :**

Transforma dois números I e J em coordenadas

- **toNumberI :**

Pega uma coordenada e devolve o número relacionado a posição I

Ex.:

```
toNumberI(c);
```

sendo c uma coordenada.

- **toNumberJ :**

Pega uma coordenada e devolve o número relacionado a posição J

Ex.:

```
toNumberJ(c);
```

sendo c uma coordenada.

- **ASK**

Para saber onde determinado robô se localiza ele deve usar a função `ask` que retornará a sua posição na forma `Coordinate`.

Ex.:

```
my c = toCoord(ask("position"));
```

- **HIT**

Para atacar um outro robô é necessário usar o comando `hit`, ele recebe como parâmetro a direção em que se deseja atacar. Essa direção deve ser da forma `Direction`. Devolvendo 1 quando conseguiu atacar o alvo naquela direção e 0 caso o ataque não tenha sido bem sucedido.

Ex.:

```
hit(->NW)
```

- **MOVE**

Para mover um robô de lugar, usa-se o comando `move`, sendo que este recebe como parâmetro a direção em forma de `Direction` e retorna 1 caso tenha andado e 0 caso contrário.

Ex.:

```
move(->E)
```

- **FIRE**

A função `fire` funciona no mesmo estilo de `HIT`, porém essa é para os robôs que tem mais alcance. Ao executar essa função, você estará atacando um alvo em determinada coordenada, caso o seu alcance permita. Ele funciona como um tiro normal, caso haja algum obstaculo no meio, este será atingido.

Essa função recebe como parâmetro a coordenada em que se deseja atacar e devolve 1 caso o ataque tenha sido realizado com sucesso e 0 caso contrário, lembrando que se o alcance do robô não permitir tal ataque, a função retornará 0 e o ataque não será executado.

Ex.:

```
fire([12,15]);
```

- **LOOK**

Essa função procura por algum item (como por exemplo o cristal). Quando usamos `look`, precisamos de um parâmetro do tipo `Item` que se refere ao tipo de objeto que estejamos procurando. Em seu retorno, obtemos a coordenada do objeto encontrado.

Ex.:

```
x = look(
```

- **DRAG**

Para coletar algo do mapa, deve-se utilizar `drag`, essa função irá retornar 1 se o objeto foi coletado e 0 caso contrário. E como parâmetro é passada a direção em forma de `Direction`;

Ex.:

```
drag(->NW);
```

- **DROP**

Para soltar algo no mapa, deve-se utilizar a função `drop`, ela irá retornar 1 se o objeto for solto com e 0 caso isso não tenha dado certo. E deve receber como parâmetro uma direção em forma de `Direction`.

Ex.:

```
drop(->E);
```

# Tipos de Variáveis

Temos os tipos:

- **Item:** Itens como rocha e cristal

`%item`

**Ex.:** `%stone`

- **Number:** Números

– número normal –

**Ex.:** `4242`

- **String:** Variáveis texto

`"mensagem"`

**Ex.:** `"Hello"`

- **Direction:** Direções como leste, oeste, nordeste, entre outros.

`->direção`

**Ex.:** `->NE`

- **Coordinate:** Coordenadas da representação quadrada da matriz diagonal.

`[I,J]`

**Ex.:** `[4,2]`

## DOCUMENTAÇÃO

A documentação do código-fonte está disponível no formato Javadoc e no formato de relatório Latex para compreensão do código.