Jogo dos Robos de Prospecção

E

Linguagem Lua

Introdução ao Jogo dos Robôs

O jogo simula um ambiente de prospecção de petróleo, no qual atuam automatos programados pelo aluno.

A finalidade do jog é fazer o aluno ter um primeiro contato com noções de programação, de algorítmos de busca e de inteligência artificial.

Será promovido um compeonato entre os autômatos criados pelos alunos.

A idéia do jogo

- Um conjunto de robôs marinhos é lançado ao mar sobre uma região onde se acredita haver petróleo.

- Os robôs atingem o solo submarino em posições aleatórias.

- Duas ou mais equipes lançam robôs semelhantes para competir por poços na mesma região.

- Os robôs são equipados com diversos sensores que fornecem constantemente informações como a posição do robô, a pressão subterrânea no local, o gradiente da pressão no local etc. Cada robô pode perfurar o solo e iniciar a prospecção de petróleo em sua posição corrente.

- A produção de petróleo em um local será proporcional à pressão subterrânea no local. Por isso, os robôs devem encontrar os locais com maior potencial de produção e iniciar perfurações nesses locais.

- Cada robô é controlado por um computador que executa um programa escrito pelo aluno. O programa controla o robô por meio de um conjunto de comandos predefinidos. Baseado nas informações provenientes dos sensores, esse programa deve decidir para onde direcionar a robô, quando iniciar a perfuração de petróleo etc, efetivamente controlando todo o comportamento do robô.

- Vence o jogo a companhia cujos robôs conseguirem produzir mais petróleo durante um período predeterminado.

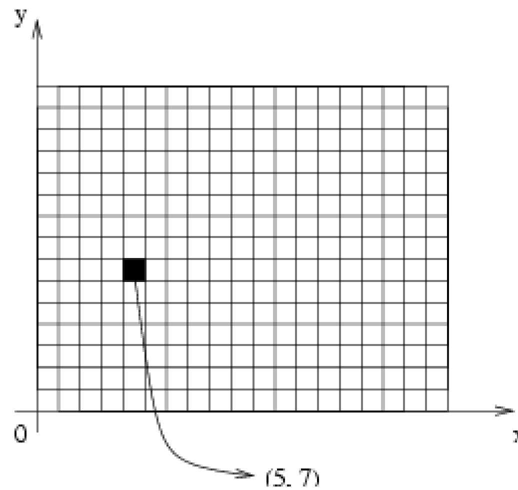
A arena

- Uma região quadrada, com uma distribuição de pressão, desconhecida dos robôs.

- Os limites extremos da arena funcionam como paredes: os robôs não conseguem sair da arena.

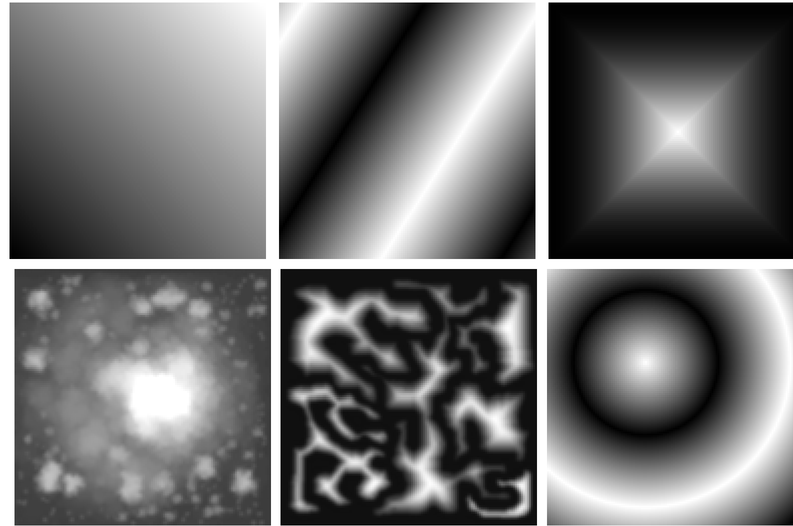
- A arena é dividida em células identificadas por suas coordenadas cartesiandas `(x, y)`, onde `x` e `y` são inteiros.

- Na figura abaixo, encontra-se ressaltada a célula `(5, 7)` da arena:



Cada célula têm um valor de pressão associado, variando dentro do intervalo `[0, 1]`.

Exemplos de Arenas



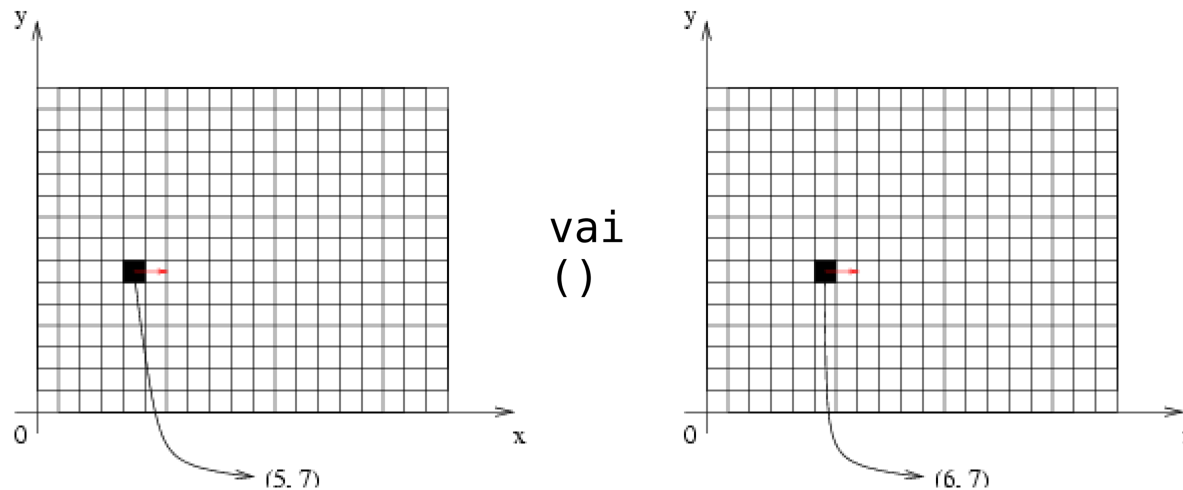
- Opontos claros representam locais de pressão elevada e pontos escuros representam locais de baixa pressão.

- Os robôs devem encontrar os locais mais claros da arena e perfurar nesses locais para maximizar a produção de petróleo.

- Os robôs dispõem apenas das medições realizadas por seus sensores.

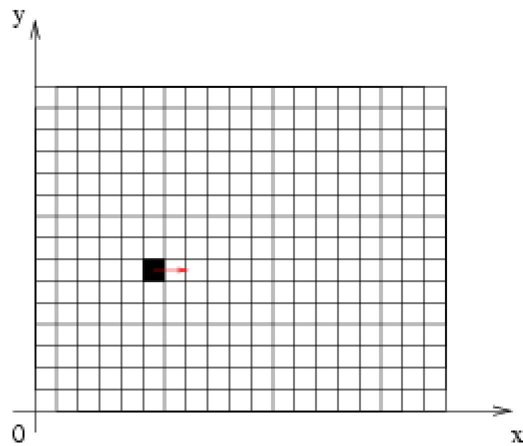
Os robôs

- Cada robô se posiciona sobre uma célula da arena de coordenadas (x_r, y_r).
- O robô está sempre voltado para uma das quatro direções (Norte, Sul, Leste, Oeste).
- A função `vai()`, o programa de um robô o fará mover-se para frente na direção para a qual está voltado.
- A função `volta()` faz com que o robô mova-se para trás:

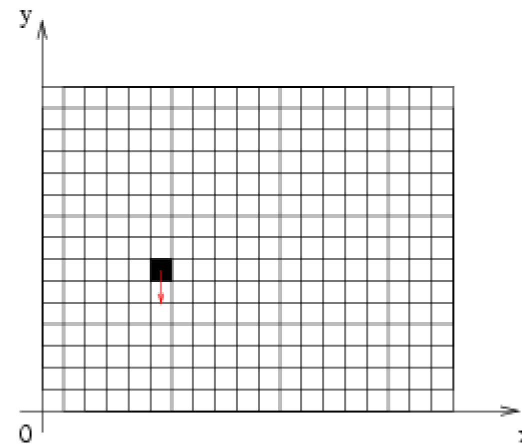


O robô gira 90º para

- a direita usando a função `direita()`
- a esquerda, com o uso das funções e `esquerda()`



direita()



Linguagem Lua

Lua está implementada como uma pequena biblioteca de funções C, escritas em ANSI C, que compila sem modificações em todas as plataformas conhecidas. Os objetivos da implementação são simplicidade, eficiência, portabilidade e baixo impacto de inclusão em aplicações.

Lua

As seguintes palavras-chave não podem ser usadas como identificadores:

and	**break**	**do**	**else**	**elseif**	
end	**false**	**for**	**function**	**if**	
in	**local**	**nil**	**not**	**or**	
repeat	**return**	**then**	**true**	**until**	**while**

Lua é uma linguagem que distingue maiúsculas de minúsculas.

Ex: **and** é palavra reservada mas And e AND são idendificadores válidos. As a

Convenção: identificadores começando por barra baixa e seguido de maiúsculas são variáveis reservadas internas

Ex: _STATUS

Lua

Os seguintes caracteres ou sequencias de caracteres são válidos

+ - * / ^ = ~= <= >= < > ==

( ) { } [ ] ; : , ...

Lua

Cadeias de caracteres literais são delimitadas por aspas simples ou duplas.

Podem ser incluidos os seguintes caracteres especiais.

\a --- toque de sinal
\b --- retorno
\f --- avanço de página
\n --- nova linha
\r --- retorno de carro
\t --- tabulação horizontal
\v --- tabulação vertical
\\ --- contrabarra
\" --- aspas duplas
\' --- apóstofre
\[--- colchete esquerdo
\] --- colchete direito

Lua

Valores e Tipos

Lua é uma linguagem dinamicamente tipada, ou seja, as variáveis não tem tipos, só os valores como são apresentados.

Tipos básicos:

nil	**boolean**	**number**	**string**
function	**userdata**	**thread**	**table**

Lua

- **nil** é indicativo de valor indefinido

- **boolean** toma os valores false e true.

  Um valor nil é entendido como false se for usado num teste lógico.

- **number** representa um número de ponto flutuante de 8 B. No entanto, também representa números inteiros.

- **string** são cadeias de caracteres que incluem o oitavo bit como informação.

- **function** são variáveis de primeira classe.

  Podem ser armazenadas em variáveis, passadas como argumento de outras funções e ser retornadas como resultado.

- **userdata** permite a atribuição de funções escritas em linguagem C.

- **table** são tabelas ou vetores associativos que permitem a indexação de qualquer tipo de valor, com escessão de nil. Permite compor estruturas de dados de vetores e outras mais complexas.

Atribuições Simples e Múltiplas

Vamos a exemplos de atribuições válidas

x = 3.141592

b = “abcd”

x, y = 5, “xyz”

a, b = 5

a, b = 1, 2, 3

a, b = b, a

Operadores aritméticos

+ (adição)	- (subtração)
* (multiplicação)	/ (divisão)
^ (exponenciação)	unario - (negação)

Operadores Relacionais

== Verifica a igualdade entre os operandos. Resultado **false** se for diferente.

~= Verifica a igualdade entre os operandos. Resultado **true** se for igual.

< Testa se o operando da direita é maior que o da esquerda.

> Testa se o operando da esquerda é maior que o da direita.

<= Testa se o operando da direita é maior ou igual que o da esquerda.

>= Testa se o operando da esquerda é maior ou igual que o da direita.

Os operandos tem que ser, em geral, de mesmo tipo.

Operadores Lógicos

and
or
not

Estes operadores lógicos atuam sobre valores **true**, **false** e **nil**.
Nem sempre as operações retornam **true** ou **false.**

Exemplos

10 or error()	**-> 10**
nil or "a"	**-> "a"**
nil and 10	**-> nil**
false and error()	**-> false**
false and nil	**-> false**
false or nil	**-> nil**

Concatenação

Exemplo:

a = Linguagem

b = Lua

c = a .. “ “ .. b

c contém a cadeia de caracteres **Linguagem Lua**

Precedência

or

and

<   >   <=   >=   ~=   ==

..

+   -

*   /

not   - (unary)

^

Estruturas de controle

if, then, else

if expr **then**
bloco
end

if expr **then**
bloco 1
else
bloco 2
end

Laços de repetição

while, do

while expr **do**
bloco
end

Laços de Repetição

repeat, until

repeat

bloco

until expr

For

2.4.6 - Function Calls as Statements

To allow possible side-effects, function calls can be executed as statements:

```
stat ::= functioncall
```

In this case, all returned values are thrown away. Function calls are explained in 2.5.7.

2.4.7 - Local Declarations

Local variables may be declared anywhere inside a block. The declaration may include an initial assignment:

```
stat ::= local namelist [`=´ explist1]
namelist ::= Name {`,´ Name}
```

If present, an initial assignment has the same semantics of a multiple assignment (see 2.4.3). Otherwise, all variables are initialized with nil.

A chunk is also a block (see 2.4.1), so local variables can be declared in a chunk outside any explicit block. Such local variables die when the chunk ends.

The visibility rules for local variables are explained in 2.6.

Regras de visibilidade

```
x = 10                  -- global
  do
    local x = x         -- novo `x', com valor 10
    print(x)            --> 10
    x = x+1
    do
      local x = x+1     -- outro `x'
      print(x)          --> 12
    end
  print(x)              --> 11
  end
print(x)                --> 10
```

Primeiro robô

Os programas responsáveis pelo controle controle dos robô devem ser escritos pelo aluno usando a linguagem Lua.

```
    w, h = tamanho()
1:  m = min(w, h)
2:  while 1 do
3:      for i = 1, random(1, m) do
4:          if not vai() then
5:                  for i = 1, random
6: (1,9) do esquerda() end
7:          end
8:      end
9:        for i = 1, random(1,9) do
10: esquerda() end
    end
```

Sem rumo

Temos um robô que perambula aleatoriamente pela arena.

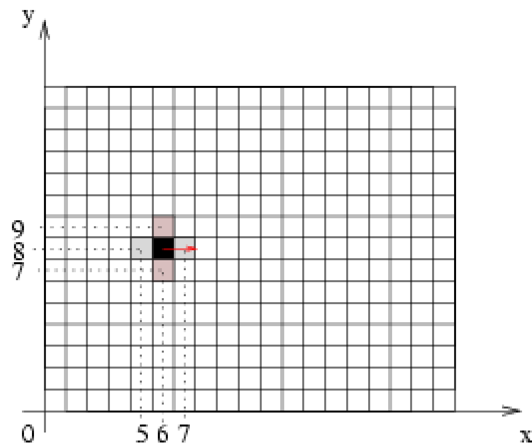
Comandos entendidos pelo robô: `tamanho()`, `vai()` e `esquerda()`.

Comandos de lua: `random()` e `min()` são comandos da linguagem Lua.

O programa ilustra a estrutura que será seguida pela maioria dos alunos: um loop (linhas 3-10) executa repetidamente o mesmo conjunto de ações.

De modo a decidir para onde se deslocar e onde realizar as perfurações, os robôs devem usar as duas funções de medição de pressão: `pressao()` e `deltas()`.

`pressao()` retorna o valor da pressão na célula em que o robô se encontra. A `delta()` retorna dois valores, que informam como a pressão está variando na direção para a qual o robô está virado (direção tangencial) e na direção perpendicular a direção do robô (direção normal).



Seja P(x,y) o valor da pressão na célula (x,y).

Assim, na figura acima, a função `deltas()` retornaria os valores `dt` e `dn` tais que `dt` = P(7,8) - P(5,8) e `dn` = P(6,7) - P(6,9).

Um robô útil

```
1:  while 1 do
2:      dt, dn = deltas()
3:      if abs(dt) > abs(dn) then
4:           if dt < 0 then direita()
5: direita() end
6:      else
7:          if dn > 0 then direita()
8:          else esquerda() end
9:      end
10:     vai()
:          if  pressao()  >  0.7  then
11: perfura() end
:   end
```

Busca gradiente

Funções dos robos: `deltas()`, `pressao()` e `perfura()`

Função definida pela linguagem Lua: `abs()`

O método usado pelo robô acima é conhecido como busca gradiente, e pode ser usado como base para vários algorítmos mais sofisticados.

O tempo

O fator limitante do jogo é o tempo.

Todas as ações executadas pelo robô levam um determinado número de unidades de tempo para serem concluídas.

Nesse meio tempo, o programa do robô fica paralisado, aguardando a conclusão de sua ação.

A unidade básica de tempo é o tempo necessário para que um robô se desloque de uma célula para a vizinha.

Os comandos dos robôs

`vai()`

Desloca o robô uma célula para frente. A função retorna o valor `1` se a ação foi completada com sucesso, ou `nil` caso algum obstáculo tenha impedido o robô de se deslocar. (Duração:1)

`volta()`

Desloca o robô uma célula para trás. A função retorna o valor 1 se a ação foi completada com sucesso, ou `nil` caso algum obstáculo tenha impedido o robô de se deslocar. (Duração:1)

`direita()`

Faz com que o robô vire 90º para a direita. A operação sempre completa com sucesso. (Duração:1)

`esquerda()`

Faz com que o robô vire 90º para a esquerda. A operação sempre completa com sucesso. (Duração:1)

`pressao()`

Retorna o valor da pressão na célula onde o robô se encontra. O valor da pressão está sempre no intervalo [0,1]. A operação sempre completa com sucesso. (Duração:1)

`deltas()`

Retornar dois valores, `dt` e `dn`, onde `dt` é o gradiente na direção tangencial ao movimento do robô, e `dn` é o gradiente na direção normal ao movimento do robô. A operação sempre completa com sucesso. (Duração:5)

posicao()

Retorna dois valores, `x` e `y`, tal que (x,y) é a a célula onde o robô se encontra. A operação sempre completa com sucesso. (Duração:1)

perfura()

Perfura um poço na célula onde se encontra o robô. A função retorna 1 se a perfuração ocorreu com sucesso, ou `nil` caso haja uma outra perfuração muito próxima. (Duração:50)

tempo()

Retorna o número de unidades de tempo que restam antes do fim do jogo. A operação sempre completa com sucesso. (Duração:1)

tamanho()

Retorna dois valores, `w` e `h`, as dimensões da arena. A operação sempre completa com sucesso. (Duração:1)

Como jogar

O jogo é implementado como dois programas independentes:

um programa servidor

um programa cliente.

O servidor é responsável por pela simulação da arena.

Os diversos clientes se conectam com o servidor e são responsáveis pela execução dos programas dos robôs.

O servidor

O servidor deve ser executado antes dos clientes, e existem várias opções que controlam o ambiente a ser simulado por ele, dentre elas:

`NUM_ROBOTS=<num>`	Controla o número de robôs pelo qual o servidor esperará antes de iniciar a competição.
`TIME_LIMIT=<num>`	Controla o número de unidades de tempo que durará a competição.
`MAP_FILE=<file>`	Controla o mapa com a distribuição de pressões que será usado na competição.

Assim, por exemplo, para iniciar um servidor que esperará por 4 robôs competidores, o usuário deve executar o seguinte comando:

```
server.exe NUM_ROBOTS=4
```

Para iniciar um servidor que esperará por 12 robôs competidores, carregando o mapa "`map1.bmp`", o usuário deve executar o seguinte comando:

```
server.exe NUM_ROBOTS=12 MAP_FILE=map1.bmp
```

Os clientes

Um cliente deve ser executado para cada robô participando da competição. O programa cliente aceita os seguintes parâmetros:

```
client.exe <team> <cor> <file>
```

Assim, para rodar um robô definido no arquivo "`probe.lua`", no time "`fulano`", com cor vermelha, o usuário deve executar a seguinte linha de comando:

```
client.exe fulano red probe.lua
```

Juntando tudo

Para organizar uma pequena partida, com dois robôs no time "pedro azul" (usando o arquivo "`probe.lua`" e dois robôs no time "joao verde" (usando o arquivo "`grad.lua`"), competindo no mapa "`map4.bmp`", durante 2000 unidades de tempo, o jogador deve usar a seguinte seqüência de comandos num prompt do Windows:

```
start server.exe NUM_ROBOTS=4 MAP_FILE=map4.bmp TIME_LIMIT=2000

start client.exe pedro blue probe.lua

start client.exe pedro blue probe.lua

start client.exe joao green grad.lua

start client.exe joao green grad.lua
```

Download

O programa pode ser baixado num arquivo zip seguindo-se o link abaixo:
robots.zip

O programa também está disponível em código fonte completo, pelo link abaixo:
robots.src.zip

Links úteis

Site da linguagem está disponível em dois mirrors:

www.tecgraf.puc-rio.br/lua

www.lua.org