A informática é uma aplicação da matemática
e da engenharia com grande sucesso e impacto
no mundo moderno. O advento dos computadores,
das comunicações em rede e da inteligência
artificial, por exemplo, moldam o presente
e definem o nosso futuro de forma imprevisível.

O Hex é um jogo moderno que introduziu
ao mundo os jogos de conexão, jogos matemáticos
onde é necessário conectar diferentes partes
do tabuleiro. A sua originalidade, simplicidade
de regras e profundidade de análise torna-o
um dos melhores jogos que existem.

10 Livros, 10 Regiões, 10 Jogos para aprender e divertir-se

Grécia – Petteia 10/07/08
China – Xiang-Qi 17/07/08
Babilónia – Ur 24/07/08
Egipto – Senet 31/07/08
Índia – Shaturanga 07/08/08
Japão – Shogi 14/08/08
África – Bao 21/08/08
Indonésia – Surakarta 28/08/08
América Pré-colombiana – Awithlaknannai 04/09/08
Europa – Hex 11/09/08

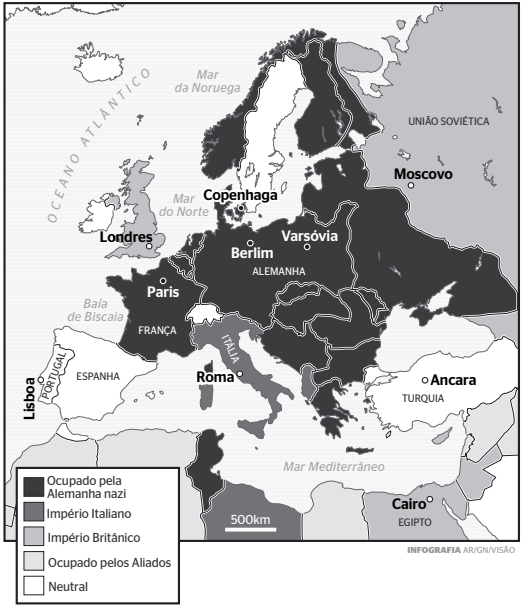
FICHA EDITORIAL
Titulo Europa - Hex
Autor Carlos Pereira dos Santos, João Pedro Neto, Jorge Nuno Silva
Revisão Edimpresa – Carla Monteiro
Impressão e acabamento Norprint
Data de impressão Junho 2008
Depósito Legal 278363/08

A Escrita de Algoritmos

A informática é uma das matemáticas aplicadas com maior sucesso e expansão nos últimos cinquenta anos. A existência física de computadores, máquinas capazes de efectuar uma vasta gama de tarefas diferentes, mudou o mundo moderno e, hoje, a nossa relação com os computadores é muito profunda.

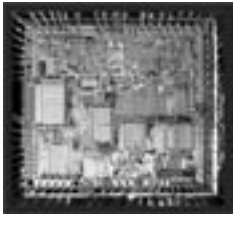
Graças ao trabalho de muitos matemáticos e engenheiros foi possível, na década de 40, a construção dos primeiros computadores, máquinas imensas que ocupavam prédios inteiros.

ENIAC, um dos primeiros computadores (1946-1955), pesava 30 toneladas, tinha 40 metros de comprimento e custou quase 500 mil dólares


OCEANO ATLÂNTICO
Mar da Noruega
UNIÃO SOVIÉTICA
Moscovo
Copenhaga
Mar do Norte
Londres
Varsóvia
Berlim
ALEMANHA
Paris
Baía de Biscaia
FRANÇA
ITÁLIA
PORTUGAL
ESPANHA
Lisboa
Roma
Ancara
TURQUIA
Mar Mediterrâneo
Cairo
EGIPTO
500km
Ocupado pela Alemanha nazi
Império Italiano
Império Britânico
Ocupado pelos Aliados
Neutral
INFOGRAFIA AR/GN/VISÃO

Mapa Europa

Uma década depois, esta primeira geração de computadores deu lugar aos computadores electrónicos, muito mais rápidos e fiáveis. A invenção do *microchip*, no fim dos anos 50, início dos anos 60, permitiu a miniaturização progressiva dos computadores[1] para níveis que seriam impossíveis de imaginar num passado não muito distante.



O *microchip*, uma floresta electrónica em miniatura

Mas este sucesso da engenharia é apenas um dos lados da questão. O facto de termos computadores cada vez mais rápidos, não nos responde ao problema de como descrever à máquina os problemas que desejamos que ela resolva. Esta segunda parte é uma das preocupações do que, hoje em dia, se designa por engenharia informática e, em especial, engenharia de *software*.

Há um conceito muito importante relacionado com esta questão: as linguagens de programação. Uma linguagem de programação é, primeiro que tudo, uma linguagem. Desta forma tem sintaxe, gramática e semântica, respectivamente, como se devem escrever correctamente as palavras, como os diferentes tipos de

[1] Uma curiosidade: desde o século XVII até meados do século XX chamava-se computador – literalmente, *aquele que computa* – à pessoa que executava algoritmos com lápis e papel.

palavras se compõem uns nos outros e qual é o significado do que escrevemos. Mas, ao contrário das linguagens naturais (como o português ou o japonês), as linguagens de programação não podem conter ambiguidades. Se a ambiguidade é algo essencial numa língua humana, ela é totalmente indesejável numa linguagem de computador. Porquê? Para que dois computadores, ao interpretar um mesmo texto dessa linguagem, se comportem exactamente da mesma forma e produzam sempre o mesmo resultado. É esta estabilidade que interessa, tanto aos programadores (que os escrevem) como aos utilizadores (que os usam).

Chamamos programas ao que escrevemos numa linguagem de programação. O que é um programa? Um programa é, basicamente, a descrição de um algoritmo[2]. Um algoritmo é, grosso modo, um conjunto de regras bem definidas que nos informa, a cada momento, o que fazer a seguir. Se usamos uma linguagem natural para descrever ideias, factos, emoções, histórias, usamos uma linguagem de programação para descrever algoritmos. Estas descrições de algoritmos são, normalmente, textos compostos por letras, números e símbolos, para facilitar o trabalho das pessoas que os escrevem. Estes tipos de textos têm, depois, de ser traduzidos numa linguagem binária de zeros e uns, reconhecida pelo computador.

Nesta secção vamos abordar, de forma muito simplificada, como usar uma linguagem de programação. Como exemplo, utilizaremos uma variante da linguagem designada por Logo, criada em 1967, e usada especialmente como ferramenta de educação. Ilustraremos a explicação com exemplos produzidos na aplicação designada Elica e disponível em *www.elica.net*.

[2] A palavra algoritmo deriva do nome do matemático árabe Al-Kwarizmi (750-830), em cujas obras apresentou vários métodos de resolução algébrica. Este matemático foi assunto do 9.º volume da colecção anterior, *Jogos com História*.

O Logo é uma linguagem de programação que produz desenhos. Pressupõe um objecto especial chamado, tradicionalmente, tartaruga, mas que é apenas uma espécie de lápis digital. Esta interacção rápida entre programas e desenhos foi um dos motivos do sucesso desta linguagem no ensino da programação às camadas mais jovens.

Uma linguagem de programação do tipo Logo contém instruções. As instruções são, por exemplo, ordens que damos à tartaruga para se mover, rodar, desenhar linhas...

A tartaruga tem um estado, ou seja, possui um conjunto de características, cujos valores variam com o tempo. Uma delas é a direcção da sua cabeça, outra é a posição da tartaruga. Outra característica é se, quando se move, desenha ou não desenha linhas no chão.

Comecemos por alguns das instruções mais simples [3]:

- *forward N* – avança *N* pontos na direcção da sua cabeça.
- *backward N* – recua *N* pontos.
- *left N* – roda a cabeça da tartaruga *N* graus para a esquerda.
- *right N* – roda a cabeça da tartaruga N graus para a esquerda.
- *penup* – passa a não desenhar no chão quando se move.
- *pendown* – passa a desenhar no chão quando se move.
- *pendown?* – verifica se a tartaruga está a desenhar.
- *home* – volta à sua posição inicial.

Os comandos são instruções que alteram o estado da tartaruga. Por exemplo, quando ordenamos que ela rode 90 graus para a direita, a sua direcção actual é alterada. Quando mandamos recuar, a sua posição actual é modificada...

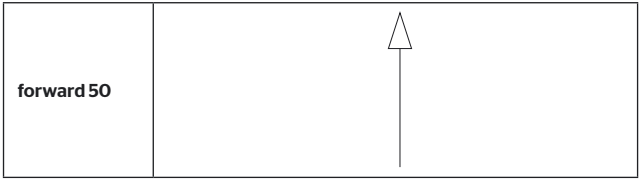
Por outro lado, há instruções que representam interro-

[3] A aplicação Elica é americana e os comandos são descritos por palavras inglesas. Preferimos manter os nomes originais para ser mais fácil experimentar a aplicação. Atenção que em outras aplicações estas instruções poderão ter nomes diferentes.

gações que queremos fazer sobre o estado da tartaruga. Das instruções referidas atrás, *pendown*? é um exemplo de interrogação. A sua execução devolve o valor verdade, se a tartaruga está a desenhar, ou falso, no caso contrário. Não é um comando, porque o estado da tartaruga fica na mesma.

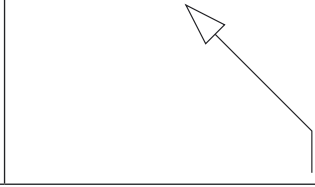
A tartaruga será representada por um triângulo. Antes de executar o programa, a tartaruga encontra-se com a cabeça na vertical:

Seja o seguinte programa e respectivo estado da tartaruga após a sua execução:



Um programa em Logo é uma sequência de comandos Logo. Os comandos de um programa devem ser lidos de cima para baixo.

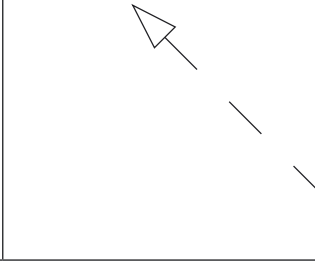
```
forward 15
left 45
forward 50
```



Neste exemplo, após um movimento vertical de 15 pontos, a tartaruga virou 45 graus para a esquerda e andou mais 50 pontos.

Um exemplo que desenha uma linha tracejada:

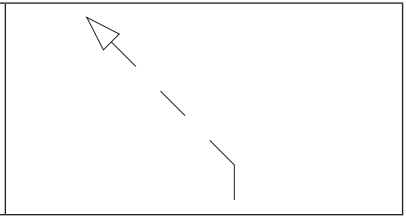
```
forward 15
left 45
forward 15
penup
forward 15
pendown
forward 15
penup
forward 15
pendown
forward 15
```



Neste caso, o programa apresentado é um pouco repetitivo. Quando há repetições deve-se usar um outro tipo de comando denominado por ciclo. O ciclo é uma instrução especial que permite repetir outras instruções um certo número de vezes. No

caso do Logo, usa-se a palavra *repeat* (em português, *repetir*) para representar um desses ciclos:

```
forward 15
left 45
repeat 2 [
 forward 15
 penup
 forward 15
 pendown
]
forward 15
```



A sintaxe desta instrução exige que, após a palavra *repeat*, se indique quantas repetições irá efectuar e depois, entre parênteses rectos, quais as instruções a repetir.

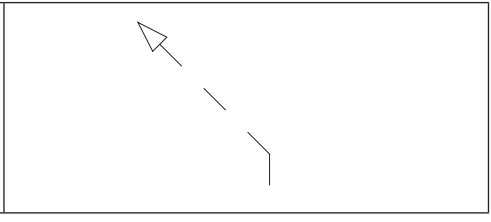
Continuando no exemplo anterior, ainda se nota alguma repetição dos comandos *forward*. Uma forma de eliminar esta repetição é usar um novo comando denominado instrução condicional que, como o nome indica, realiza uma acção se, e só se, uma determinada condição for verdade. No caso da linguagem Logo, esse comando escreve-se *if* (em português, *se*).

A forma correcta de escrever este comando (a sua sintaxe, portanto) é a seguinte:

if condição
[as acções a efectuar se a condição for verdade]
[as acções a efectuar se a condição for falsa]

O exemplo anterior poderia ser reescrito da seguinte forma:

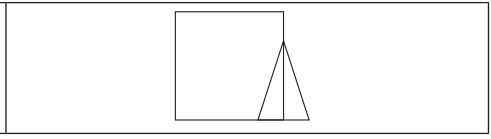
```
forward 15
left 45
repeat 5 [
 forward 15
 if pendown?
  [penup]
  [pendown]
]
```



Ou seja, quando se entra no ciclo, após andar para a frente 15 pontos, faz-se a pergunta se a tartaruga está a desenhar *(if pendown?)*. Se for verdade, executa-se o comando do primeiro parênteses recto *([penup])* e a tartaruga passa a não desenhar. Se for falso, ou seja, a tartaruga não estava a desenhar, executa-se o comando do segundo parênteses recto *([pendown])* e a tartaruga passa a desenhar. Isto significa, então, que a tartaruga alterna o seu estado entre desenhar e não desenhar cada vez que se repete o ciclo. Este comportamento é o indicado para desenhar uma linha tracejada.

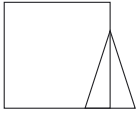
Vejamos outro exemplo onde se desenha um quadrado:

```
repeat 4 [
 forward 20
 left 90
]
```



Considere o leitor que queríamos desenhar vários quadrados, mas gostaríamos de evitar a repetição das instruções anteriores. É possível guardar um conjunto de instruções e dar-lhes um nome novo. De certa forma, o que estamos a fazer é criar uma nova instrução. No caso do Logo, usa-se as palavras *to* e *end* como se fossem os parênteses entre os quais colocamos as instruções novas:

```
to quadrado
 repeat 4 [
  forward 20
  left 90
 ]
end
quadrado
```



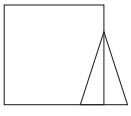
Algumas notas. Após a palavra *to*, temos de escrever o nome da nova instrução (optámos por chamar quadrado). A seguir, escrevemos as instruções que compõem a nossa nova instrução e, quando já tivermos escrito tudo o que queríamos, terminamos com a palavra *end*.

O leitor, porém, pode observar que, após a definição desta nova instrução, escrevemos, na linha seguinte, novamente a palavra quadrado. Porquê? Porque nas primeiras seis linhas do programa anterior apenas definimos a nova instrução, não ordenámos a sua execução. A ordem de desenhar o quadrado só ocorre na última linha do programa, quando mandámos executar a instrução definida anteriormente.

Mas já que estamos a criar as nossas instruções, podemos ser

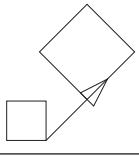
mais exigentes. Por exemplo, a instrução *forward* é acompanhada por um parâmetro numérico que diz o quanto a tartaruga deve avançar (por exemplo, *forward* 50). Porque não adicionar à nossa instrução quadrado uma funcionalidade similar? Desta forma, deixamos à instrução que ordena o desenho do quadrado a decisão de definir qual deve ser o seu tamanho. Na linguagem Logo isso é possível. Basta adicionar, após o nome da instrução nova, uma ou mais palavras (prefixadas com dois pontos) e que podem ser usadas, como parâmetros, pelas instruções seguintes:

```
to quadrado :lado
 repeat 4 [
  forward lado
  left 90
 ]
end
quadrado 20
```



Desta forma, cada vez que ordenamos o desenho de um quadrado, teremos de fornecer a dimensão do seu lado, para que a tartaruga saiba como fazer o desenho. A nova instrução pode ser usada várias vezes:

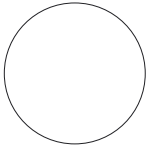
```
quadrado 20
right 45
forward 30
quadrado 35
```



Com as ferramentas apresentadas já podemos construir imagens interessantes.

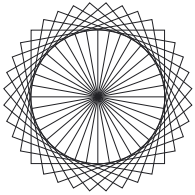
Vejamos alguns exemplos:

```
hideturtle
repeat 72 [
 forward 5
 left 5
]
```



Esta imagem é uma aproximação de uma circunferência ao desenhar-se um polígono de 72 lados. Nestas imagens não aparece a tartaruga porque o comando *hideturtle* torna-a invisível.

```
hideturtle
repeat 36 [
 quadrado 40
 left 10
]
```

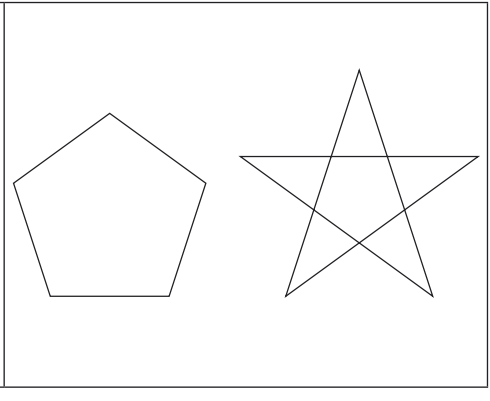


Esta última imagem é composta por 36 quadrados desfasados 10 graus dos seus vizinhos.

Um pentagrama desenha-se como um pentágono, mas fazendo a tartaruga rodar 144 graus em vez de 72 graus.

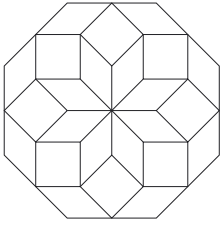
Desta forma, para desenhar os dois, vale a pena criar uma instrução nova designada *penta* que recebe dois parâmetros, *lado* e *angulo*. O primeiro valor determina a dimensão da figura a desenhar, enquanto o segundo determina quanto a tartaruga deve rodar para desenhar o lado seguinte:

```
to penta :lado
     :angulo

 repeat 5 [
   forward lado
   right angulo
 ]
end

hideturtle
right 18
penta 100 144
penup
left 108
forward 50
pendown
penta 50 72
```



Após o desenho do pentagrama, há algumas instruções que fazem a tartaruga avançar, sem realizar desenhos, para o local onde se inicia o desenho do pentágono.

```
hideturtle
repeat 8 [
 repeat 8 [
  forward 25
  right 45
 ]
 right 45
]
```



Esta imagem mostra um exemplo de *ciclos encadeados*, ou seja, ciclos dentro de ciclos, repetições dentro de repetições. Esta possibilidade permite usar a repetição de instruções de forma bastante elaborada, como se pode observar no exemplo anterior.

Recursão ou como dar ordens a si próprio

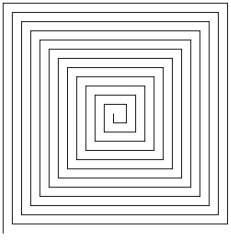
Estas novas instruções que podemos definir possuem uma característica muito poderosa que ainda não falámos: grosso modo, elas podem ordenar a sua própria execução. Chama-se recursão a esta característica e com o seu uso consegue-se realizar tarefas muito complexas de forma sucinta. O nosso objectivo neste texto é apenas mostrar alguns exemplos simples da sua utilidade.

Estudemos um exercício. Se quisermos que a tartaruga desenhe uma espiral, podemos fazê-lo da seguinte forma: criar uma nova instrução (chamada espiral, por exemplo) com um parâmetro que determina o tamanho da espiral (seja *lado*) e que faça o seguinte:

a) ande para a frente lado pontos;
b) vire para a direita (e agora vem o truque da recursão...);
c) mande executar espiral, mas agora com um *lado* menor.

O programa poderia ser o seguinte (a linha com a ordem recursiva está a negrito):

```
to espiral :lado
 if lado < 1
  [ ]
  [forward lado
   right 90
   espiral lado-2
  ]
end

hideturtle
espiral 100
```



Cada vez que o comando espiral é executado, o seu *lado* vai diminuindo em duas unidades (devido à expressão *lado-2*). Deste modo, as ordens executadas pelo desenrolar do programa são uma sequência iniciada por espiral 100, espiral 98, espiral 96, espiral 94,...

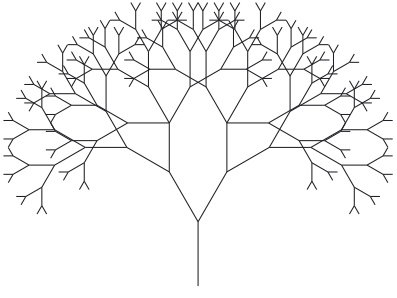
Isto levanta um problema. Se deixássemos continuar este processo, a execução do programa nunca terminaria. Quando chegasse a espiral 2 continuaria com espiral 0, espiral -2, espiral -4,...

Assim, é necessário definir um momento de paragem desta sequência, ou seja, onde a recursão termina. Isso é controlado pelo comando *if* na 2.ª linha do programa acima. Enquanto o va-

lor do parâmetro *lado* não for menor que 1, a execução continua. Quando *lado* for um número menor que 1, o desenho da espiral terminará (não é preciso fazer mais nada, basta terminar, daí os primeiros parênteses rectos estarem vazios).

Outro exemplo, com uma estrutura recursiva similar, mas mais elaborado:

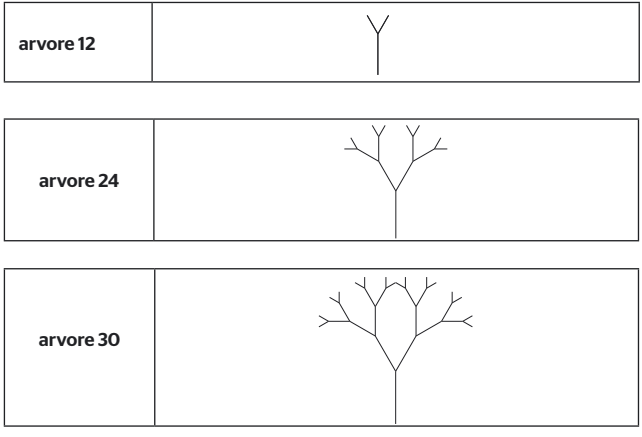
```
to arvore :dim
 if dim < 3
  [ ]
  [forward dim
   right 30
   arvore dim-6
   left 60
   arvore dim-6
   right 30
   backward dim
  ]
end

hideturtle
arvore 50
```



Este é um exemplo de um objecto fractal[4], onde cada ramo da árvore é similar à árvore em si, sendo apenas desenhado numa menor escala. O número de iterações da árvore é dado pelo número do parâmetro *dim* na execução inicial da instrução árvore.

Vejamos o resultado de diferentes execuções desta instrução com dimensões iniciais diferentes.

[4] Os fractais foram apresentados no 5.º volume da colecção *Jogos com História*, livro dedicado ao matemático Benoit Mandelbrot, considerado por muitos o pai da teoria dos fractais.



Quanto menor for o valor inicial *dim*, mais depressa a sequência recursiva de execuções da instrução arvore chegará ao seu término (a paragem é dada quando a condição *dim* <3 for avaliada como verdade).

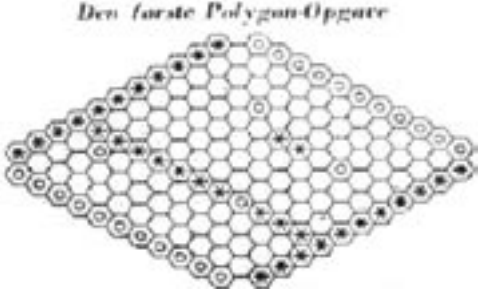
Jogo do *Hex* (Foto retirada da exposição *Jogos Matemáticos Através dos Tempos*, presentemente no Museu de Ciência da Universidade de Lisboa)

Uma breve História do *Hex*

O jogo do Hex foi inventado pelo matemático e poeta dinamarquês Piet Hein que, ao pensar num famoso problema matemático (o problema das quatro cores), teve a ideia de usar um conceito geométrico para conceber um jogo de tabuleiro. Hein apresentou o jogo em 1942. A sua criatividade era de realçar, sendo célebres os seus poemas curtíssimos, os *grooks*. Eis um exemplo:

The road to wisdom? – Well, it's plain
and simple to express:
Err
and err
and err again
but less
and less
and less

Chamou ao jogo *Polygon* no jornal dinamarquês *Politiken*. Este jornal teve uma coluna sobre o jogo, publicando cerca de 50 problemas até Agosto de 1943, sendo esta paragem, provavelmente, devida à anexação nazi do país (Hein tinha relações com a resistência e a sua mulher era judia):



O primeiro problema, publicado em 26 de Dezembro de 1942

Na sua primeira aparição no *Politiken*, o jogo do *Hex* é descrito da seguinte forma:

Gostaria de aprender a jogar Polygon*? Piet Hein inventou um jogo que pode ser praticado com igual prazer tanto por eruditos jogadores de* Xadrez *como por pessoas apenas capazes de pegar numa caneta.*

Pouco a pouco, o *Hex* revelou ser um jogo com um potencial extraordinário e com a boa característica de poder ser alvo de vários níveis de compreensão e análise.

Piet Hein (1905-1996)

Alguns anos depois, em 1948, um estudante da Universidade de Princeton, nos EUA, inventou o ***Hex*** de forma independente. Esse estudante chamava-se John Nash, o futuro Nobel da Economia e personagem altamente mediatizada por ser figura principal do filme oscarizado, *A Beautiful Mind*. Por essas paragens era comum chamar-se ao jogo *John* ou *Nash*. O *Hex* pode ser igualmente praticado nas casas hexagonais de um tabuleiro em forma de losango ou nas intersecções de um tabuleiro com casas quadrangulares cortadas com uma diagonal. Devido a esse facto, parece ser verdade que o que inspirou Nash foi o padrão da parede de uma casa de banho. Como nos EUA, é costume chamar-se *John* às casas de banho, ficou feito um trocadilho com a designação do jogo.

John Nash (hoje com 80 anos)

Nash mostrou o jogo a outro estudante de Princeton, David Gale (1921–2008), que imediatamente verificou o seu potencial. Gale construiu um tabuleiro que era partilhado pela comunidade universitária de Princeton.

Em 1952, uma companhia de jogos, a *Parker Brothers*, comercializou este jogo chamando-lhe *Hex*. Provavelmente, a companhia terá recolhido informação através de alguém proveniente de Princeton.

A primeira comercialização: *Hex*, o jogo zig-zag

O jogo do *Hex* ganhou parte da sua fama actual a partir de um artigo de Martin Gardner na sua célebre coluna na revista *Scientific American*, atribuindo a autoria a Piet Hein, mas referindo também Nash e a sua demonstração que garante a vitória ao primeiro jogador no caso deste jogar de forma perfeita (falaremos sobre este facto mais à frente). No seu artigo, Gardner

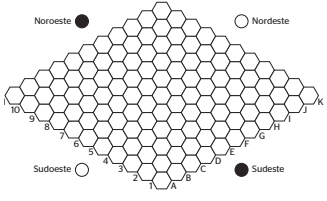
propôs vários problemas interessantes, bem como um exemplo de estratégia ganhadora para um tabuleiro não simétrico.

O *Hex* é, por excelência, um exemplo de um jogo moderno. Classificando os jogos quanto ao seu objectivo, existem jogos muito antigos com objectivo de captura, de ganho de território, de bloqueio, de alinhamento, etc. No entanto, jogos em que o objectivo consiste em criar um grupo de peças que una dois lados de um tabuleiro — *jogos de conexão* — não eram conhecidos antes do séc. XX. O *Hex* é um jogo de conexão. Existem outros, tais como o *Y*, o *Nex*, o *Gonexão* (inventado por João Pedro Neto), o *Jade*, etc.

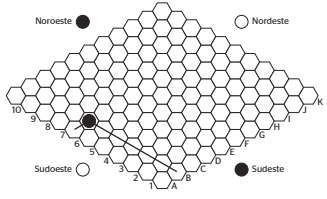
O *Hex* é um daqueles jogos cuja relação entre a simplicidade das regras e a riqueza do jogo é quase perfeita. É fantástico constatar que um jogo com regras tão simples pode ser tão rico de estratégia. Existem alguns livros dedicados à estratégia do *Hex*, entre os quais destacamos o *Hex Strategy*, escrito por Cameron Browne.

Regras do jogo do *Hex*

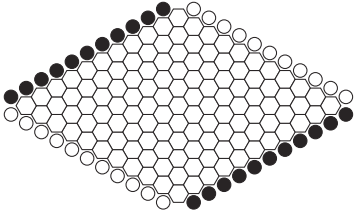
O jogo envolve dois jogadores, dispondo de peças de cores diferentes (digamos que um dos jogadores joga com peças brancas e o outro com peças negras). Nesta colecção utilizaremos um tabuleiro 11×11, mas o *Hex* pode ser praticado com tabuleiros de outras dimensões. Não há convenção universal para a cor que começa. É usual sortear-se:



Por uma questão de organização, utilizaremos um sistema de coordenadas para referir as casas do tabuleiro. A título de exemplo, mostramos na próxima figura a casa *B7* assinalada:



Os círculos que apresentamos nas figuras servem para assinalar os detentores das margens. As casas *A1*, *A11*, *K1* e *K11* pertencem às margens de ambos os jogadores. No seu artigo inicial, em 1942, Piet Hein ilustrou as margens com o esquema seguinte:

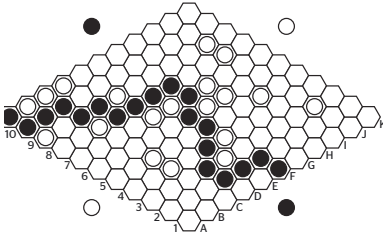


Cada jogador coloca, à vez, uma peça da sua cor dentro de um hexágono. Duas peças de um jogador dizem-se unidas se os hexágonos onde se encontram partilharem uma aresta:

Peças unidas ou conexas

O objectivo do jogador das negras consiste em conseguir um caminho negro que una as margens sudeste e noroeste. Quer isto dizer que o jogador das negras pretende conseguir ter um

grupo conexo de peças suas em que pelo menos uma do grupo pertença à margem sudeste e que pelo menos outra do grupo pertença à margem noroeste. O objectivo do jogador das Brancas é análogo, ou seja, consiste em conseguir um caminho branco unindo as margens sudoeste e nordeste.



Exemplo de posição vitoriosa negra

Neste jogo não há capturas. Cada jogador coloca, à vez, uma peça sua numa casa do tabuleiro e esta fica nessa casa até ao fim do jogo.

Há também uma última regra denominada regra do equilíbrio. Quem já compreende minimamente o *Hex* sabe que o primeiro jogador que coloque a sua primeira peça num hexágono razoavelmente central ganha vantagem considerável. Sendo assim, desde a sua nascença que se constatou este problema no *Hex*: o primeiro a jogar está favorecido. Teve de se arranjar forma de tornear o problema. O assunto pode ser resolvido indo buscar uma ideia a uma curiosa questão envolvendo uma partilha.

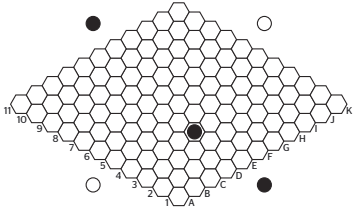
Imagine o leitor que tem de dividir uma tarte com um amigo. Como fazer para essa divisão ser justa? A resposta é surpreendentemente simples e muitos já a aplicaram no dia-a-dia: um parte e o outro escolhe. Com este entendimento, quem parte terá de se preocupar com a justiça. Se a divisão originar uma fatia escandalosamente maior do que a outra, quem escolhe recolhe obviamente a maior.

Como podem dois amigos dividir de forma justa uma saborosa tarte de maçã?

A regra do equilíbrio, *pie rule* em inglês (*pie* significa tarte), baseia-se exactamente no mesmo conceito: no primeiro lance, o segundo jogador pode trocar de cores ficando com a

jogada efectuada pelo adversário. O objectivo da inclusão desta regra é retirar a força a uma primeira jogada que seja demasiado centralizada. Com esta regra, o primeiro jogador já não deve jogar um lance demasiado central, caso contrário o segundo jogador opta por trocar.

Vejamos um exemplo da regra em acção. Imagine-se que o primeiro jogador jogava uma peça preta em *E4*:

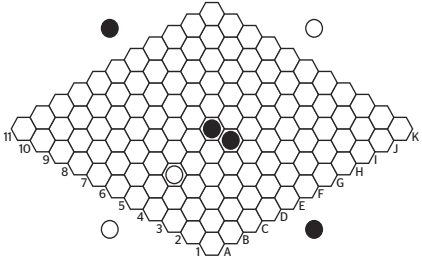


O segundo jogador, ao considerar a jogada bastante centralizada, podia dizer: *Eu fico com essa jogada e a partir de agora jogo com as Negras, tu a partir de agora jogas com as Brancas. É a tua vez.*

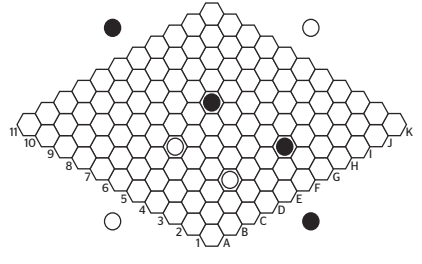
Alguma estratégia

A) Conexões:

O *Hex* é um jogo de conexão e, como tal, a primeiríssima coisa que um jogador de *Hex* deve pensar é na forma de unir as suas peças. Vejamos o exemplo seguinte:



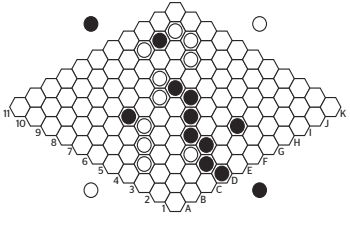
Embora se encontrem unidas, as peças negras de *F5* e *F6* estão demasiado perto uma da outra, não tendo muita influência para a globalidade do tabuleiro. Vejamos outro exemplo:



Aqui, as peças negras encontram-se afastadas de mais e já é necessário ter em conta uma jogada defensiva branca em *G5*.

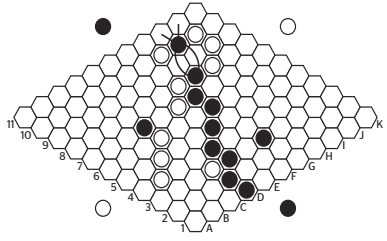
Uma das estruturas de conexão mais fortes do *Hex* é a que se apresenta no esquema seguinte (usualmente denominada de ponte):

Um principiante, quando experimenta o *Hex* pela primeira vez, por vezes, tem a tendência de juntar as suas peças. No entanto, ao olhar para esta importante estrutura podemos compreender a razão pela qual, na maioria das vezes, esse procedimento é pouco económico. Repare o leitor que, embora as duas peças negras ainda não se encontrem unidas, já nada o pode impedir. Isso acontece devido ao facto de as Negras terem à sua disposição duas hipóteses distintas de conexão. Sendo assim, se as Brancas defenderem uma delas, as Negras conectam com a outra. No fundo, trata-se de uma ameaça dupla de união. A título de exemplo analisemos, no próximo diagrama, uma jogada negra ganhadora aproveitando esta ideia:



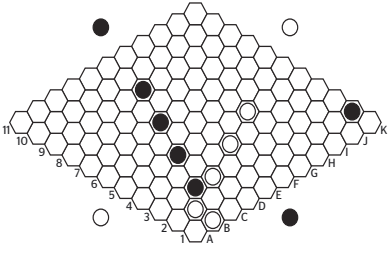
Jogam as Negras e ganham

Aqui é possível a criação de pontes vitoriosas com a jogada em *H8*:



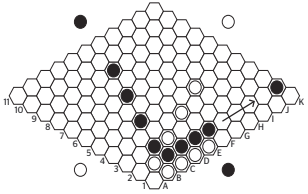
B) Escada:

Vejamos agora um conceito estratégico muito importante, que se baseia na utilização de um ponto de apoio. Observe-se a seguinte posição:



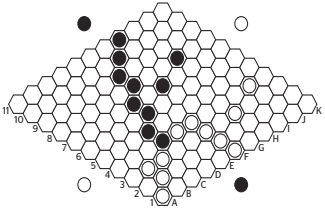
Jogam as Negras

O procedimento ganhador consiste em aproveitar a influência da peça em *K2*. A jogada inicial é em *C2* e, em seguida, com uma sequência de ameaças em linha recta, conseguir-se-á o apoio pretendido. A este tipo de procedimento dá-se o nome de *escada*.



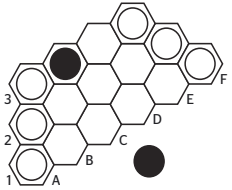
C) Zonas de conexão garantida:

Da mesma maneira que os jogadores de *Xadrez* sabem que um Rei e uma Dama ganham a um Rei isolado e outros conceitos do género, os jogadores de *Hex* reconhecem certo tipos de padrões geométricos que repetem na sua prática de jogo. Vejamos um exemplo:

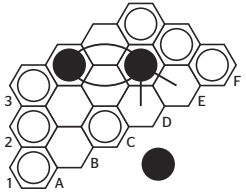


Jogam as Negras

Um jogador experimentado de *Hex* sabe que a zona ilustrada na figura seguinte traduz o espaço suficiente para garantir uma conexão negra à margem sudeste. É a este tipo de disposições geométricas que chamamos zonas de conexão garantida. Esta espécie de trapézio repete-se em inúmeros jogos de *Hex*:



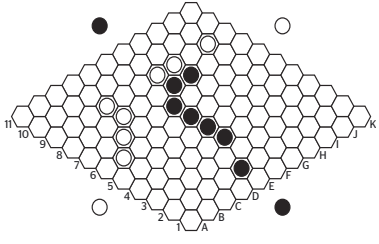
Repare o leitor que contra a defesa branca em *C1*, as Negras podem forçar a conexão com a jogada em *E2*:



No exemplo inicial, colocar uma peça em *D3* garante uma vitória para as Negras.

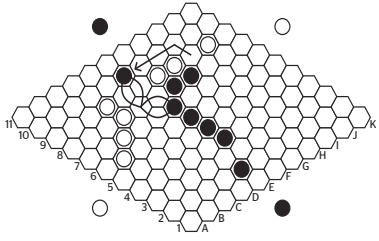
D) Preparação de Escada:

Já vimos que o conceito de escada se baseia na busca do apoio de certa peça. Por vezes, a peça de apoio é previamente colocada no tabuleiro com recurso de uma dupla ameaça. Repare-se no exemplo seguinte:



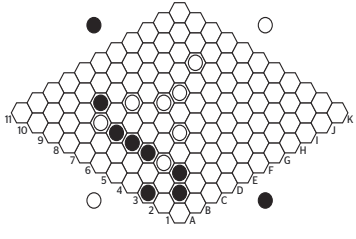
Jogam as Negras

A invasão negra em *I9* só pode ser defendida com a colocação de uma peça branca em *J11*. Neste momento, ainda não há um apoio para uma futura escada. Um procedimento ganhador é a jogada das Negras em *F10*, que ganha um tempo ameaçando a conexão em *E8*. Esta peça em *F10* servirá de futuro apoio para uma escada proveniente da invasão em *I9*.



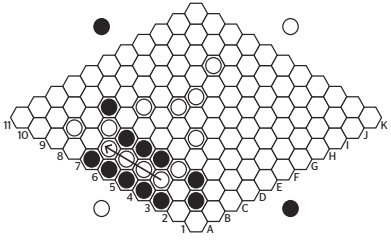
E) Contra-escada:

Tendo em conta os conceitos anteriores, vejamos um procedimento de defesa muito comum no *Hex*:

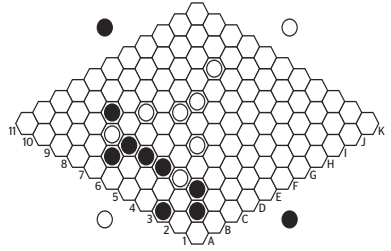


Jogam as Negras

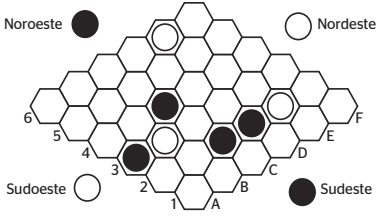
A posição negra parece desesperada devido às ameças brancas em *D8* e em *B4*. Por exemplo, se as Negras jogarem em *D8*, as Brancas fazem uma *escada*:



O leitor pode observar que a boa defesa consiste no primeiro lance em *B7*. Uma vez que o objectivo deste lance é impedir uma futura *escada*, chama-se a este procedimento *contra-escada*.



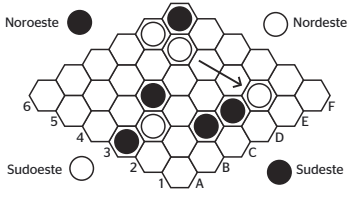
Posta esta lista de conceitos, analisemos um problema retirado do *Politiken*:



Jogam as Brancas e ganham (*Politiken*, 28 de Dezembro de 1942)

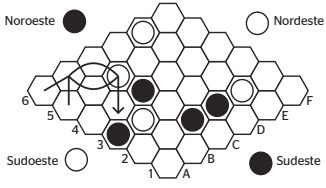
Ao tratar este problema, o jogador das Brancas pode pensar da seguinte forma:

a) Analisando com atenção, pode concluir-se que, mesmo que sejam as Negras a jogar, não conseguem impedir que a peça em *E6* se conecte à margem nordeste. Se as Negras defenderem em *F6*, as Brancas constroem uma escada, jogando em *E5* indo ao encontro da ajuda da sua peça que está *E2* (ver esquema seguinte).

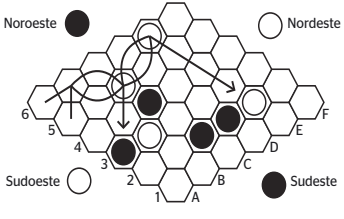


Sendo assim, as Brancas não devem jogar na zona de conexão de nordeste, uma vez que essa conexão já está garantida. Este procedimento é fundamental no *Hex: não se perde tempo em conexões que já estão garantidas.* Repare-se também no papel fundamental da peça em *E2*. O *Hex* é um jogo de *influência.*

a) Jogando uma peça em *C5* cria-se uma dupla ameaça (em *B6* e em *B4*):



Consequentemente, a jogada em *C5* garante a conexão a sudoeste e, consequentemente, ganha a partida. Tal como era feito no *Politiken*, a solução pode ser apresentada em esquema da seguinte forma:

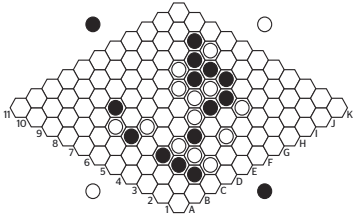


René Descartes, grande sábio francês nascido em 1596, escreveu uma obra sobre um método para ajudar a humanidade a pensar bem *(Discurso do Método)*. Um dos vários princípios por ele apontado é o seguinte: *(...) Dividir a dificuldade nas parcelas possíveis e necessárias para melhorar a sua resolução. (...)*. Um problema de *Hex* é geralmente um exemplo cristalino de aplicação deste princípio. Normalmente, o jogador divide o seu problema nas várias conexões possíveis, algumas garantidas e outras por garantir, tentando conjugar tudo para conseguir a decisão global.

Este pequeno resumo estratégico é o possível numa publicação como esta, mas já serve para dar uma pequena ideia sobre o conteúdo do *Hex*. O leitor interessado pode procurar mais informação em *www.hexwiki.org*.

Exemplo de uma partida

Em 2009, vai realizar-se na Covilhã a final da quinta edição do Campeonato Nacional de Jogos Matemáticos (ver *http://ludicum.org/*). Este evento, que conta na organização com algumas das mais destacadas instituições portuguesas relacionadas com a matemática, tem-se revelado um grande sucesso, contando com dezenas de milhares de participantes do ensino básico e secundário. Um dos jogos escolhidos tem sido o *Hex*. Para se observar o bom nível atingido por alguns dos nossos jovens, escolhemos, como exemplo, uma partida da segunda edição do campeonato.



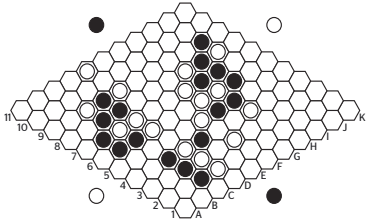
2CNJM, Aveiro, 2006
Jogador das Negras: Pedro Jorge; Jogador das Brancas: Carlos Louro
Jogam as Negras

1.*B6*

Excelente jogada, utilizando o conceito da *contra-escada*. A jogada 1. *D7* era má devido à sequência 1…*C5* 2. *D5 E6*.

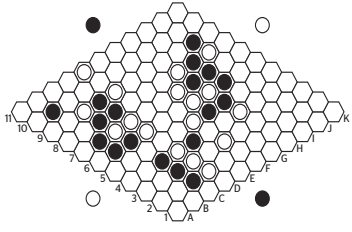
1…*D7* 2. *B7 C9* 3. *C8 E9* 4. *D9 E11*

Repare-se agora como o jogador das Negras realiza e uma preparação de escada com recurso a uma ameaça dupla:



5. *B10*

Ao realizar esta jogada, as Negras ameaçam simultaneamente uma conexão em *A8* e uma incursão em *D10* seguida de uma *escada*, aproveitando o apoio de *B10*. O jogador das Brancas abandonou.



Posição depois de 5. *B10*

A matemática e o *Hex*

Além do interesse que o *Hex* tem como jogo, os matemáticos consideram-no muito interessante por ter várias conexões directas com tópicos matemáticos (alguns deles nada triviais). Uma demonstração que é possível fazer com todo o rigor é a de que não pode haver empates no *Hex*. Embora a demonstração formal saia fora do âmbito deste texto, podemos dar uma imagem do que se passa da seguinte forma: imagine-se que as peças negras são terra e as peças brancas são água. Se enchermos alea-

toriamente o tabuleiro de *Hex* de peças negras e brancas, no caso em que não haja um caminho de terra unindo SE a NO é porque a água flui de NE a SO. Queremos dizer com isto que é impossível preencher o tabuleiro de *Hex* sem que haja um caminho ganhador para um dos jogadores. Como curiosidade, é interessante referir que o colega de John Nash, David Gale, desmonstrou uma surpreendente equivalência entre o facto de o jogo do *Hex* não ter empates e um resultado avançado de matemática.

No já citado artigo de Martin Gardner é explicado como John Nash demonstrou que, sem regra de troca, o primeiro jogador ganha sempre um jogo de *Hex*. Vejamos o argumento:

1.º Uma vez que não há empates, tem de existir uma estratégia ganhadora para o primeiro jogador ou para o segundo jogador.

2.º Assuma-se que o segundo jogador tem uma estratégia ganhadora, tendo em vista obter um absurdo.

3.º Tendo em conta a hipótese anterior, o primeiro jogador pode adoptar a seguinte estratégia: joga uma peça num local qualquer e em seguida joga a estratégia ganhadora do segundo jogador. Uma vez que no *Hex* uma jogada extra *nunca é prejudicial*, a primeira jogada não provoca nenhuma má interferência com a dita estratégia ganhadora. Sendo assim, com esta atitude, o primeiro jogador ganhará, o que contraria a hipótese de haver estratégia ganhadora para o segundo jogador.

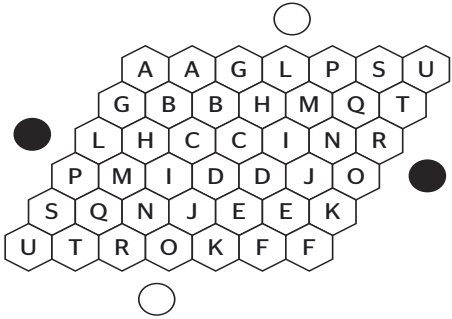
Conclusão: Não pode haver estratégia ganhadora para o segundo jogador, o que implica que é o primeiro que tem estratégia ganhadora.

Notas:

Esta demonstração não é construtiva no sentido em que não explicita a estratégia ganhadora. Sendo assim, não estraga o prazer de jogar. É sabido que em jogos abstractos como o *Xadrez*, *Damas*, etc., existe um número finito de jogos possíveis (mas absolutamente gigantesco!). Sendo assim, a árvore de variantes possíveis é finita e, admitindo que ambos os jogadores a conhecem e a percorrem da melhor maneira (segundo o ponto de vista de cada um), existe um resultado determinado à partida. Em jogos como o popular *Jogo do Galo*, o ser humano tem capacidade para listar todas as hipóteses, compreendendo que, não havendo erros, o jogo termina empatado. Em casos como o *Xadrez*, nem o melhor computador da actualidade consegue determinar o resultado. É o facto de nós *não conhecermos* a *boa estratégia* que torna os jogos interessantes.

No *Hex*, com a regra da troca, é o segundo jogador que ganha. O argumento é simples: se o primeiro jogador efectuar um movimento ganhador, o segundo jogador troca e ganha; se o primeiro jogador efectuar um movimento perdedor, o segundo jogador fica com uma estratégia ganhadora ao seu dispor.

No seu artigo, Gardner mostra uma interessante estratégia para tabuleiros não simétricos. A ideia consiste em utilizar um método de casas conjugadas que garanta ao segundo jogador uma vitória. Repare-se na seguinte figura:



Sempre que o primeiro jogador (peças negras) fizer uma jogada, bastará ao segundo jogador (peças brancas) fazer uma jogada na letra correspondente. Esta construção simétrica elaborada por Gardner garante a vitória ao segundo jogador. O sucesso desta estratégia deve-se ao facto dos caminhos vitoriosos mais curtos do primeiro jogador serem mais compridos do que os caminhos mais curtos do segundo.