

Introdução

Uma parte substancial desta obra tem sido utilizada na primeira disciplina de programação do curso de informática da Faculdade de Engenharia da Universidade do Porto (FEUP). Os trabalhos realizados no âmbito desta disciplina apoiam-se em plataformas onde se evidencia a linguagem Scheme, linguagem de programação pouco difundida em Portugal, apesar da sua larga utilização em importantes universidades americanas e europeias. Em Portugal, para além da FEUP, só conhecemos a sua utilização em cursos de informática do Instituto Superior Técnico (IST) da Universidade Técnica de Lisboa, situação que gostaríamos de ajudar a alterar.

Esta obra apresenta algumas características inovadoras como manual de ensino, pois os autores tiveram a preocupação de ir ao encontro de novas atitudes pedagógicas, apostando no paradigma da aprendizagem activa, onde o aluno é incentivado a ser o actor principal.

Também não foi descurada a crítica que classifica o Scheme como linguagem apenas para uso no ensino da programação, tendo sido reservado espaço para o estabelecimento de pontes desta linguagem com outras tecnologias importantes para a construção de software (unidades de teste, desenvolvimento de GUI, programação OO, aplicações multimédia e jogos).

Em termos de público-alvo, identificam-se os estudantes com interesse na iniciação à programação, sobretudo alunos do ensino superior, mas também alunos do ensino secundário da área da informática.

Aprender programação com a linguagem Scheme é uma opção já com alguns anos em importantes universidades americanas (MIT, Yale e Indiana) e também europeias. Na Faculdade de Engenharia da Universidade do Porto (FEUP), a utilização do Scheme iniciou-se em 1991 na Licenciatura de Gestão Industrial, e em 1994 na Licenciatura de Engenharia Informática e Computação. Em Portugal, fora da FEUP, apenas se conhece o seu uso no Instituto Superior Técnico da Universidade Técnica de Lisboa.

A simplicidade sintáctica desta linguagem permite que o esforço do estudante incida sobretudo na programação e não nas ferramentas de apoio à aprendizagem. Para combater o argumento de que o Scheme é sobretudo uma linguagem de aprendizagem de programação, procurou-se o estabelecimento de pontes entre o Scheme e outras tecnologias importantes para construção de software (unidades de teste, desenvolvimento de GUI, programação OO, aplicações multimédia e jogos). Ficam assim garantidos a ligação do Scheme a outras disciplinas desta área e também a sua quota de espaço na resolução de problemas em ambiente "menos académico".

A versatilidade desta linguagem é aproveitada para introduzir, naturalmente, o paradigma da programação funcional, mas também outros paradigmas de programação como, por exemplo, a orientação por objectos. É, sem dúvida, uma oportunidade rara para o estudante se envolver, mesmo que de uma forma breve, na implementação destes paradigmas e não só na sua utilização como acontece com a generalidade de outras linguagens.

Quando tanto se fala na mudança de paradigma de ensino, aqui aposta-se no paradigma da aprendizagem, pelo incentivo do aluno ao "aprender-fazendo". Este objectivo é procurado através de um estilo de apresentação de conteúdos que promove uma aprendizagem activa, provocando a atenção, a experimentação e a procura de respostas.

*A inserção de perguntas ou pequenos desafios, relacionados com os temas em presença, são lançados para despertar a atenção do estudante.
O que acha desta estratégia de actuação?*

Os conteúdos surgem organizados em Partes, compostas por módulos de uma a duas dezenas de páginas, tanto quanto possível autónomos e que possibilitam uma sequência personalizada de aprendizagem e de aprofundamento de conhecimentos.

Em sintonia com este estilo de apresentação, é também incluído um espaço de interacção ligado ao texto, designado por Laboratório Scheme, que permite o desenvolvimento e teste dos numerosos exemplos e exercícios propostos sem que a atenção do estudante seja muito perturbada pela entrada em cena de outra plataforma de trabalho.

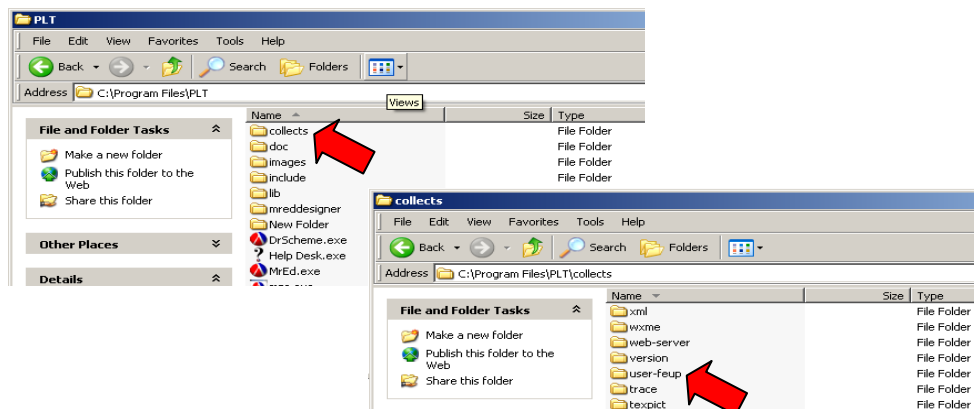
Se está disposto e com vontade de enfrentar este desafio, vai ter que começar por instalar uma implementação do Scheme no seu computador. Combinado?

Nós vamos utilizar o DrScheme, uma plataforma onde foram testados os muitos exemplos que irá encontrar neste percurso que agora inicia, uma plataforma de domínio público, disponível para sistemas *Linux*, *Mac* e *Windows*.

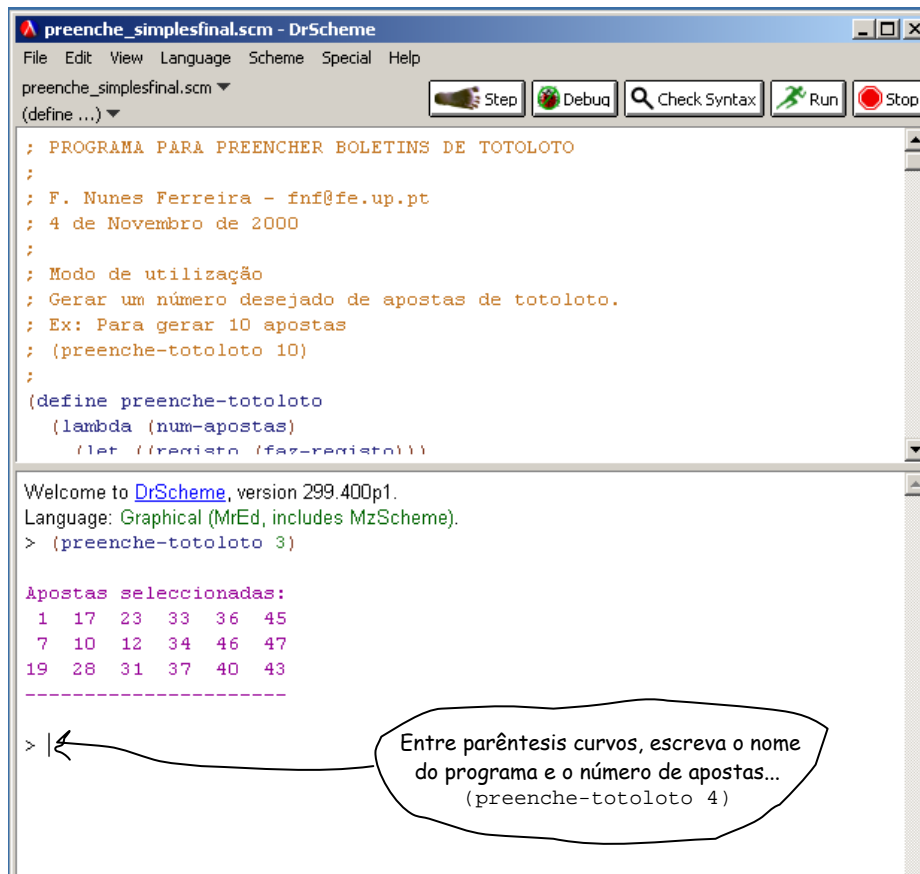
Pode começar por instalar a mesma versão 372 que temos utilizado, mais recentemente, nos exemplos e exercícios que irá encontrar, e que está disponível localmente para [Windows](#), Mac OS X ([Intel](#) e [PPC](#)) e Linux ([Ubuntu](#), [Ubuntu Feisty](#), [Fedora Core 6](#) e [Fedora 7](#)).

Todavia, poderá optar por uma versão mais actualizada, procurando em [PLT Scheme](#), onde, para além do DrScheme, poderá encontrar muitos outros recursos para utilizar neste ambiente.

Em qualquer um dos casos, depois da instalação do DrScheme, deve copiar para o seu directório `PLT/collects/` do DrScheme, o directório [user-feup](#), com código preparado por professores e alunos da FEUP e que lhe será muito útil.



E agora, está tudo preparado para poder experimentar o primeiro, entre muitos, exercício de laboratório, no espaço que se segue.



```
preenche_simplesfinal.scm - DrScheme
File Edit View Language Scheme Special Help
preenche_simplesfinal.scm
(define ...)
Step Debug Check Syntax Run Stop

; PROGRAMA PARA PREENCHER BOLETINS DE TOTOLOTO
;
; F. Nunes Ferreira - fnf@fe.up.pt
; 4 de Novembro de 2000
;
; Modo de utilização
; Gerar um número desejado de apostas de totoloto.
; Ex: Para gerar 10 apostas
; (preenche-totoloto 10)
;
(define preenche-totoloto
  (lambda (num-apostas)
    (let ((registro (faz-registo)))
      ...)))

Welcome to DrScheme, version 299.400p1.
Language: Graphical (MrEd, includes MzScheme).
> (preenche-totoloto 3)

Apostas seleccionadas:
1 17 23 33 36 45
7 10 12 34 46 47
19 28 31 37 40 43
-----
> |
```

Entre parêntesis curvos, escreva o nome do programa e o número de apostas...
(preenche-totoloto 4)

Espaço para testar o Laboratório Scheme.
O ecrã abre com o programa *preenche-totoloto*.

Actue o botão run... tudo recomeça e pode continuar a pedir apostas...

Panorâmica da obra

Esta obra é composta por um conjunto de Partes, que agora são apresentadas de uma forma resumida.

[Parte 1](#)- O essencial sobre Scheme
[Parte 2](#)- Recursividade
[Parte 3](#)- Abstracção de dados
[Parte 4](#)- Procedimentos como objectos de 1ª classe
[Parte 5](#)- Abstracções com dados mutáveis
[Parte 6](#)- Scheme e outras tecnologias
[Parte 7](#)- Exercícios e projectos
[Anexo A](#)- Principais procedimentos do Scheme
[Anexo B](#)- Abstracção Janela Gráfica
[Anexo C](#)- Reutilização de código
[Anexo D](#)- Abstracção Tabuleiro

A [Parte 1](#), *O essencial sobre Scheme*, faz uma breve introdução à linguagem Scheme, considerando os *números*, *expressões*, *símbolos*, *definição de procedimentos* e *estruturas de selecção*. No final desta Parte 1, encontrará uma introdução à programação através do uso de *abstracções*, com exemplos de programas que produzem sons e resolvem labirintos. Para além desta Parte, a sintaxe do Scheme resumir-se-á, fundamentalmente, a notas pontuais ao longo de outras partes da obra e a referências a um dos anexos ([Anexo A](#)- *Principais procedimentos do Scheme*).

*Aqui apenas se sugere um pequeno desvio até ao [Anexo A](#).
O que é que acha de uma linguagem cuja sintaxe se resume num anexo de 10 páginas,
ocupadas, em grande parte, por exemplos?*

A [Parte 2](#), *Recursividade*, considera o conceito da *recursividade* e a definição de *procedimentos recursivos*, procedimentos que se chamam a si próprios. É através deste tipo de procedimentos que se ultrapassa, no Scheme, a ausência dos ciclos que existem na generalidade das outras linguagens de programação. Mas a *recursividade* é muito mais rica do que isto, pois habilitar-nos-á a encontrar ideias para resolver problemas, alguns de grande complexidade, que de outra forma não seriam fáceis de resolver. As chamadas dos procedimentos geram *processos* no computador. A noção de *processo*, seja ele *recursivo* ou *iterativo*, é também considerada, sendo ainda analisada a forma como consome recursos computacionais, *tempo* de CPU e *espaço* de memória. Estes recursos consumidos são avaliados através da *Ordem de crescimento* ou *Ordem de complexidade*. Esta Parte termina mostrando que os procedimentos podem ser vistos como *blocos* ou *caixas-pretas*, sendo ainda apresentadas as vantagens e desvantagens que daí resultam.

A [Parte 3](#), *Abstracção de dados*, mostra como, a partir de dados simples, podemos compor os dados que se adequam à representação dos problemas que pretendemos resolver. Por exemplo, num problema de características gráficas a 2 dimensões (2D), a criação da entidade *ponto* é perfeitamente justificável, sendo constituída por dois valores numéricos que representam as suas coordenadas *x* e *y* no plano. Para este caso, será necessário agrupar dois valores numéricos criando objectos do tipo *ponto-2D*. Na criação de novas entidades a partir de dados simples ou de outras entidades já criadas utilizam-se os *constructores* e o acesso aos elementos das novas entidades é realizado com os *selectores*. É em torno destes constructores e selectores que normalmente se desenvolve uma certa funcionalidade que permite tratar as entidades criadas através de uma linguagem própria, surgindo assim a noção de *abstracção de dados*.

A complexidade dos problemas poderá exigir uma análise que identifique as suas principais tarefas e

destas, as que ainda forem consideradas complexas também serão sujeitas a uma análise idêntica. E o processo repete-se, enquanto a complexidade de alguma tarefa o exigir. Trata-se da abordagem *de-cima para-baixo*, em que se decompõe o problema em problemas cada vez mais simples.

Hoje, depois da leitura desta introdução, imagine que vai tirar o seu carro da garagem para dar uma volta. No entanto, o carro tem uma roda furada. Identifique e escreva num papel, a sequência de operações que deve realizar para tirar o carro da garagem.

Na sequência das operações, vou imaginar que indicou uma designada por “mudar a roda furada”. Por se tratar de uma operação relativamente complexa, dedique-lhe um pouco mais de atenção. E agora, só para esta operação, identifique e escreva num papel a sequência de operações elementares em que ela se pode decompor.

Se tiver conseguido tudo isto, terá escrito um programa. Se não tiver conseguido, não desanime, pois ainda agora está no início... Mesmo que tenha escrito um programa para tirar o seu carro da garagem, explique por que razão o computador não o entenderia. O que faltará fazer?

A [Parte 4](#), *Procedimentos como objectos de 1ª classe*, começa por mostrar que os *objectos de 1ª classe* são, normalmente, os operandos (*números, booleanos, pares e símbolos*), entidades que podem ser processadas e, por isso, reconhecidas como *objectos passivos*. Os procedimentos são *objectos activos*, pois associam-se-lhes actividades de processamento. As propriedades dos objectos que os caracterizam como os *objectos de 1ª classe* são: 1- Ligam-se a símbolos; 2- são passados como argumentos de procedimentos; 3- são devolvidos como valores de retorno de procedimentos; 4- são elementos de estruturas compostas de dados. Nesta Parte, poderá verificar que os procedimentos também apresentam estas quatro características e, por isso, eles próprios são também *objectos de 1ª classe*.

A [Parte 5](#), *Abstracções com dados mutáveis*, mostra que, para além dos *constructores* e *selectores*, também existem *modificadores*. Começam por surgir as chamadas *listas mutáveis* que estarão na base da implementação das abstracções *filas, pilhas e tabelas*. Os *vectores* e as *cadeias de caracteres (string)* são abstracções de dados mutáveis directamente disponibilizadas pelo *Scheme*, e mostra-se o que trazem de novo em relação às restantes abstracções.

Nesta Parte é ainda introduzido o tema *ficheiros*, tendo em conta a necessidade de guardar em disco dados de uma sessão para outra sessão de trabalho.

Perante várias abstracções, as fornecidas pela linguagem e as criadas pelo programador, poderá não ser fácil decidir qual a melhor em cada situação ou até se não será necessário criar uma nova. No decurso desta Parte procura-se analisar as grandes diferenças entre *vectores* e *listas mutáveis*. As listas são estruturas de dados dinâmicas, de grande flexibilidade, pois é possível juntar-lhes ou retirar-lhes elementos. As listas têm um acesso do tipo *sequencial*, pois para chegar a um dos seus elementos obriga a passar por todos os elementos que o antecedem. Os *vectores* são normalmente estruturas estáticas, de dimensão fixa, mas todos os seus elementos são igualmente acessíveis, através do respectivo índice. Por seu turno, as *árvores de pesquisa binária* surgem como uma estrutura que procura reunir o melhor dos dois mundos, o melhor dos *vectores* e das *listas*.

Do que acabou de ser dito, conseguirá já identificar o que será o melhor dos vectores? e o melhor das listas? Nesta Parte, poderá ver como se comparam as árvores de pesquisa binária em termos do que têm de melhor os vectores e as listas.

Na [Parte 6](#), Scheme e outras tecnologias (GUI, jogos, Unidades de teste, Executáveis e Programação OO), o objectivo principal é mostrar que o Scheme não é só aquela linguagem essencialmente usada para aprender a programar. Isso fica provado com: o desenvolvimento de interfaces gráficas (GUI) para os programas Scheme; o desenvolvimento de programas multimédia, em especial jogos, sobre plataformas multimédia de acesso gratuito; o teste ao funcionamento correcto do código desenvolvido, pela associação de unidades de teste sempre prontas na ajuda à identificação de falhas; a geração de código executável a partir de programas Scheme, permitindo a execução desse código em computadores que não tenham Scheme instalado; e com a demonstração da flexibilidade do Scheme para introduzir conceitos do paradigma de programação orientada por objectos, tanto do ponto de vista da implementação desses conceitos como do respectivo uso.

A [Parte 7](#), *Exercícios e projectos*, não é mais do que um conjunto de enunciados de exercícios e projectos, poucos com pistas de solução, de forma a colocar o estudante perante problemas de programação completamente desligados deste ou daquele tema. Quando se coloca um exercício no final de uma parte, uma primeira pista de solução fica implicitamente estabelecida, mas no mundo real, os problemas não aparecem rotulados ou associados a partes ou a capítulos de qualquer livro. Perante um problema, torna-se necessário encontrar uma ideia para o resolver, vaga no início, mas sucessivamente mais refinada. Só à medida que as várias tarefas e sub-tarefas do problema vão sendo identificadas, é que se clarificam as associações com as matérias tratadas. Surge a forma de representar os dados do problema, acompanhada de um certo conjunto de construtores, selectores e até modificadores, ou seja, surge a *abstracção de dados*. E é sobre a abstracção idealizada, com as tarefas e sub-tarefas identificadas, que se refina o desenvolvimento da solução.

Programas desenvolvidos em Scheme e alguns dos projectos propostos

A aprendizagem da programação deverá ser sistematicamente praticada no computador, o que justifica a inclusão de numerosos exemplos, exercícios e também alguns projectos de pequena/média dimensão. No caso particular dos projectos, procurou-se que fossem particularmente atractivos, pois exigem algum tempo e esforço e também alguma imaginação. É um desafio que se coloca e que transmitirá, a quem o vencer, um elevado grau de autoconfiança e gosto pela programação.

Desde já se apresentam exemplos de programas desenvolvidos em Scheme, alguns deles de autoria de alunos, onde se mostra o que se poderá fazer com o Scheme num curso de iniciação à programação.

Programa 1 - *troca e conta-trocas* - Troca quantia expressa em cêntimos e calcula número de trocas distintas

```
; (conta-trocas quantia)
; devolve numero de hipoteses de troca de uma quantia

; (troca quantia)
; devolve hipotese de troca com menos unidades monetarias

(define unidades-monetarias
  ; lista das unidades monetarias especificadas em centimos
  (list 50000 20000 10000 5000 2000 1000 500 200 100 50 20 10 5 2 1))

Welcome to DrScheme, version 299.400p1.
Language: Graphical (MrEd, includes MzScheme).
> (conta-trocas 3)
2
> (conta-trocas 5)
4
> (troca 3)
2 centimo
1 centimo
ok
> (troca 5)
5 centimo
ok
>
```

Espaço para testar os programas *troca* e *conta-trocas*.
O ecrã abre com os programas a testar.

Programa 2 - *adivinhar* - Adivinha um número pensado pelo utilizador.

```
> (adivinhar 5)

Pense num numero entre 0 e 31!

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31
P1- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31
P2- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0

4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31
P3- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
P4- O numero pensado esta' no conjunto (1=sim, 0=nao)? 1

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
P5- O numero pensado esta' no conjunto (1=sim, 0=nao)? 0

O numero pensado foi: 13
>
```

5 significa que o programa vai adivinhar um número entre 0 e 31, no final de 5 perguntas.

Espaço para testar o programa *adivinhar*.
O ecrã abre com o programa a testar.

Também pode jogar com um número diferente de 5 perguntas. Pode tentar...

Programa 3 - *labirinto* - Um pequeno animal percorre um labirinto, com visualização num tabuleiro

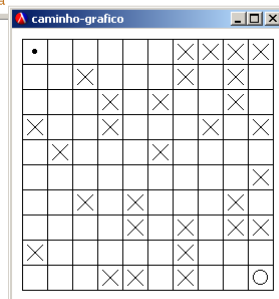
Normalmente, o pequeno animal parte da célula do canto superior-esquerdo do tabuleiro e tem como objectivo alcançar a célula do canto inferior-direito... mas o melhor é experimentar. E atenção ao som introduzido na versão do aluno Luís Santos, sobretudo quando o animal atinge o objectivo (e não se poderá dizer que seja muito bem educado!...).

```
; LABIRINTO
; Baseado no codigo caminho2.scm do professor Fernando Nunes Ferreira
; inf@fe.up.pt
; Modificado por Luis Santos
; ei00008@fe.up.pt
;
; correr o programa -> (caminho)
;
; Opções do Menu
; 1-> Definir obstaculos um a um
; 2-> Definir uma nova posicao de partida

> (caminho)
Lado do tabuleiro em celulas : 10
Lado da celula em pixels: 25

1-Definir obstaculo manualmente
2-Obstaculos aleatorios
3-Definir posicao partida
4-Definir posicao final
5-
6-
7-Andar (metodo Luis Santos)
8-Andar (metodo Prof. FNF)
9-Limpar trajetoria

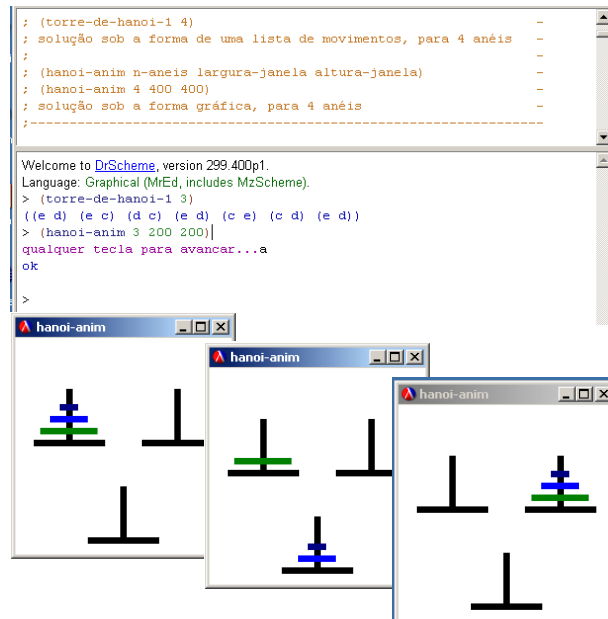
10- fim
=>2
numero de obstaculos? 35
```



Espaço para testar o programa *labirinto*.
O ecrã abre com o programa a testar.
Tente as várias opções do menu.

Programa 4 - *torre-de-hanoi-1* e *hanoi-anim* - A torre de Hanoi com solução textual e gráfica

Os anéis colocados inicialmente na torre da esquerda (e) devem, no final, estar todos na torre da direita (d), usando-se a torre central (c) como auxiliar. Só se pode movimentar um anel de cada vez e nunca um anel de menor diâmetro poderá estar debaixo de um anel de maior diâmetro.



Espaço para testar os programas *torre-de-hanoi-1* e *hanoi-anim*.

O ecrã abre com os programas a testar.

Para testar a solução textual com 3 anéis:

```
> (torre-de-hanoi-1 3)
```

Para testar a solução gráfica com 4 anéis, numa janela gráfica de 400 x 400:

```
> (hanoi-anim 4 400 400)
```

Tente um número variado de anéis, tanto na solução textual como na gráfica.

Programa 5 - Jogo *hangman* - cabe ao utilizador adivinhar uma palavra

O jogador é desafiado a adivinhar uma palavra, indicando letras, uma a uma. Nesta tarefa pode contar com algumas ajudas do computador, nomeadamente, o comprimento da palavra e, das letras já tentadas, o computador fornece a lista das que estão erradas e a posição na palavra das letras indicadas correctamente.

```
> (hangman)  
palavra a adivinhar: (* * * * * *)  
letras erradas: ()  
Proxima letra: o  
errou...  
palavra a adivinhar: (* * * * * *)  
letras erradas: (o)  
Proxima letra: a  
palavra a adivinhar: (* a * a * * a)  
letras erradas: (o)  
Proxima letra: v  
palavra a adivinhar: (* a * a v * a)  
letras erradas: (o)  
Proxima letra: p  
palavra a adivinhar: (p a * a v * a)  
letras erradas: (o)  
Proxima letra: l
```

```

palavra a adivinhar: (p a l a v * a)
letras erradas: (o)
Proxima letra: r
palavra a adivinhar: (p a l a v r a)
letras erradas: (o)
Acertou...

```

Espaço para testar o programa *hangman*.
O ecrã abre com o programa a testar.

Na listagem do programa, veja as palavras que podem aparecer. Tente juntar novas palavras, tendo o cuidado de separar as letras por um espaço. Actue o botão run e recomece o jogo...e até pode surgir alguma das palavras que juntou às iniciais.

Programa 6 - *jogo-damas* - Jogo das damas num tabuleiro visualizado no ecrã

Este jogo foi concebido e desenvolvido pelo aluno Pedro Sousa, mas apresenta ainda algumas limitações, nomeadamente, as pedras que atingem a última linha não permitem mais movimentos (*dama mestre*) e também não são admitidos movimentos para “comer” mais do que uma peça.

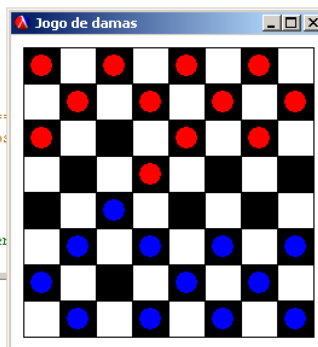
```

; Jogo feito por:
;           Pedro Daniel Sousa
;           e-mail: ei050608fe.up.pt
;
; Ano lectivo: 2005/2006
;
; =====
; Chamada de bibliotecas necessarias ao jogo

(require (lib "swgr.scm" "user-feup"))
(require (lib "tabuleiro.scm" "user-feup"))

; =====
; Cria uma janela com um tabuleiro, ajustado
; nao necessita argumentos
;
(define jogo-damas
  (lambda ()
    (display "Lado do tabuleiro, de damas, e"
      (let ((dim (read)))

```

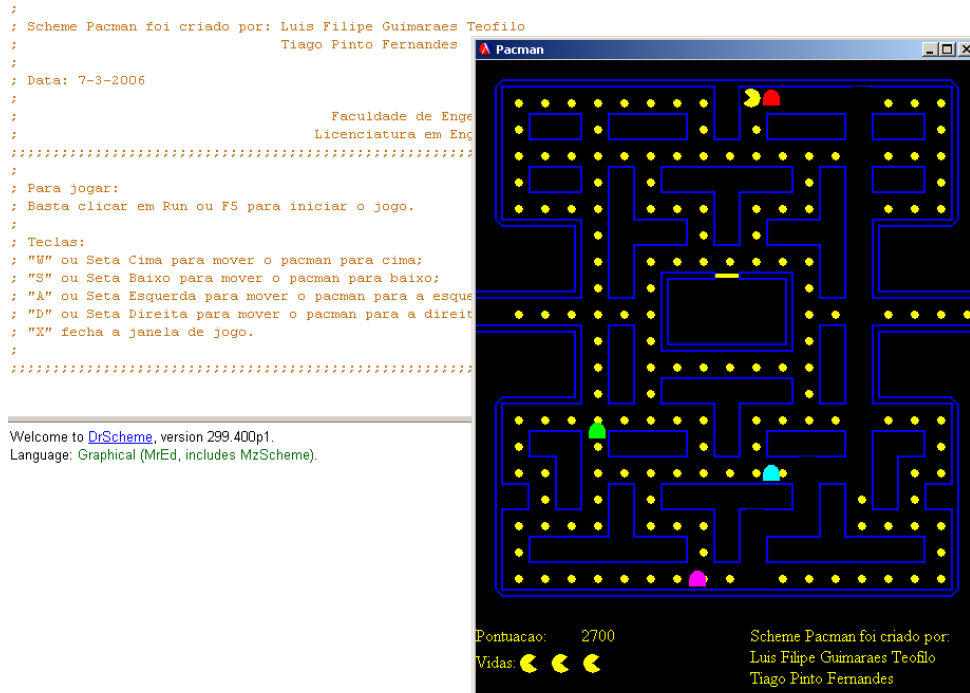


Espaço para testar o programa *jogo-damas*.
O ecrã abre com o programa a testar.

*Quando aprender um pouco mais de Scheme e programação, não querará completar este programa?
Fica o desafio...*

Programa 7 - Jogo *pacman*

Este programa foi desenvolvido pelos alunos Tiago Fernandes e Luís Teófilo e oferece uma certa interactividade através das teclas *w* (cima), *s* (baixo), *a* (esquerda), *d* (direita) e *x* (fim). Trata-se de um jogo que exige alguma rapidez e habilidade, mas o melhor é experimentar.

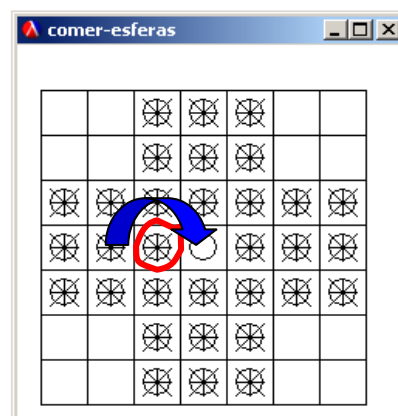


Espaço para testar o programa pacman.
O ecrã abre com o programa a testar.

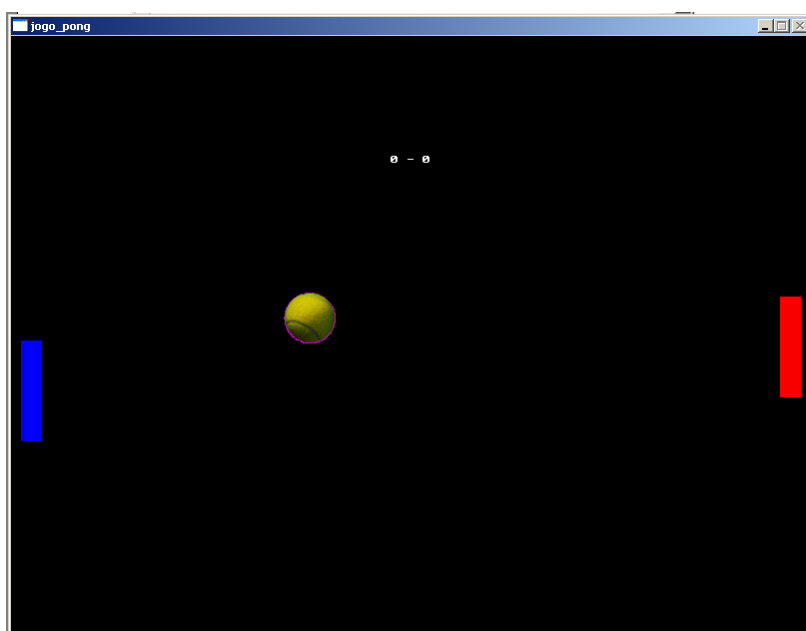
Quanto a projectos propostos, distinguem-se apenas os mais significativos.

Na Parte 3, surgem três propostas de projecto: um primeiro com elevado nível de dificuldade, designado por *Colorir Mapas*, relacionado com o problema de colorir um mapa de países com um número limitado de cores, evitando que a mesma cor seja utilizada para colorir países com fronteiras adjacentes; um segundo projecto, o *Jogo das minas*, simula um terreno de minas, em variadíssimas situações, desde as minas muito ricas às muito pobres. De acordo com uma forma original de se deslocar no terreno, um mineiro vai visitando as minas, recolhendo (nas ricas) ou perdendo (nas pobres) riqueza; o terceiro projecto da Parte 3, *Lançamento-de-projecteis*, reveste a forma de um jogo electrónico simplificado, com interface gráfica, onde se procura alcançar, com um projectil, um objecto colocado aleatoriamente no espaço para testar a pontaria de quem joga. Trata-se de um projecto com visualização gráfica animada. O *CODEC* é o projecto introduzido na Parte 4 que trata, de uma forma muito simplificada, o problema da codificação e decodificação de mensagens.

Na Parte 5 distinguem-se dois projectos: o primeiro, *helicóptero-digital*, em que se controla, através do teclado, um helicóptero imaginário; no segundo, *comer-esferas*, imagine que se retira uma das esferas na base indicada na fotografia, criando assim um buraco. Agora, uma esfera “come” outra se passar por cima dela para ocupar o buraco. Na figura, a esfera na base da seta, na sua passagem para o buraco “come” a esfera assinalada.

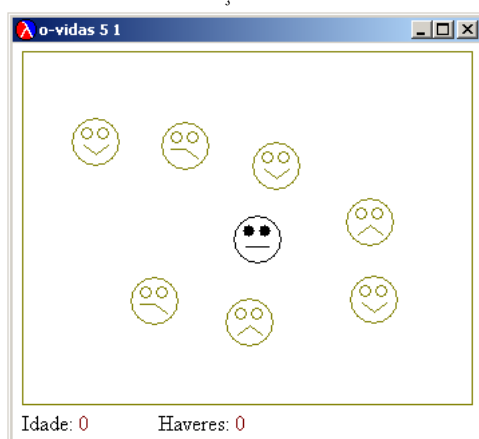


Na Partes 6 destacam-se dois projectos. O jogo Pong, tendo por base uma plataforma para desenvolvimento de programas com interface multimédia. Esta plataforma, designada por Allegro, encontra-se documentada em [Allegro: Multimedia library and game utilities](http://allegro.cc/).

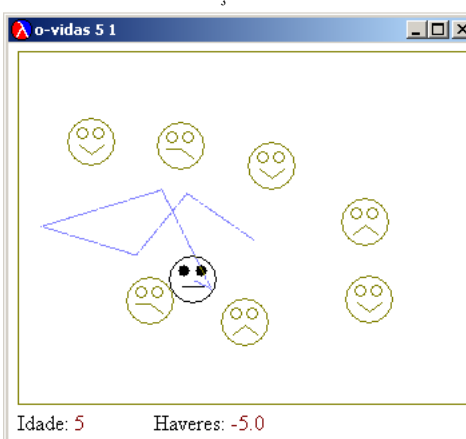


O jogo de o-vidas é um projecto com um grau de dificuldade já bastante elevado e aparece com uma implementação orientadas por objectos. Simula o percurso de uma criatura num tabuleiro, designado por o-vidas, que muda de trajectória quando embate nas paredes ou em criaturas distribuídas aleatoriamente no tabuleiro...

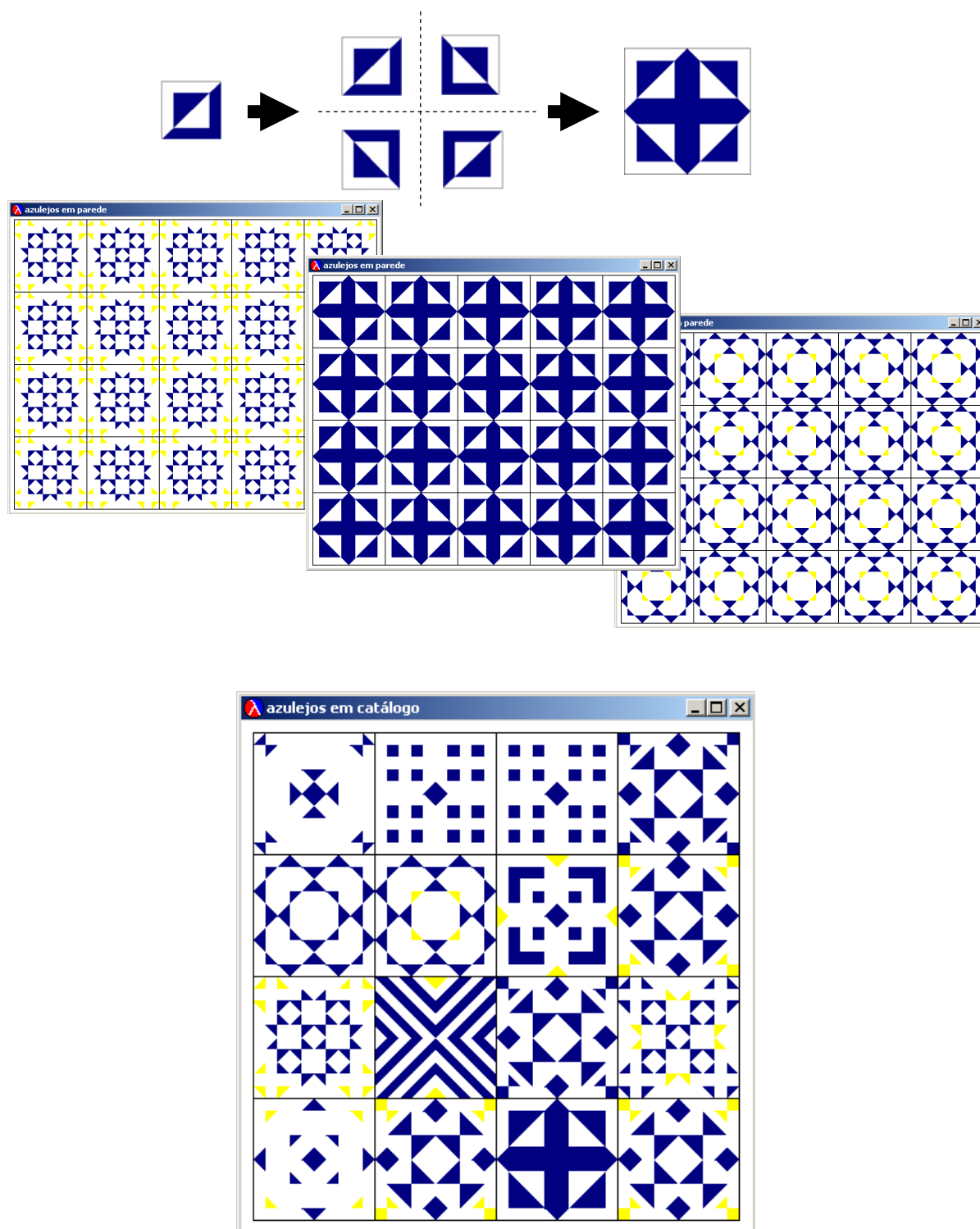
Situação inicial



Situação final



São vários os projectos incluídos na Parte 7 e apenas se distinguem três: *Caminho-mais-curto*, uma espécie de labirinto em que os percursos são caracterizados pelo respectivo comprimento e procura-se o caminho mais curto entre um ponto à escolha e uma saída; *Labirinto*, relacionado com uma das várias técnicas de resolução de labirintos; *Pintura de azulejos*, um programa de apoio à concepção de azulejos, começando por elementos, passando por azulejos isolados e terminando em catálogos de azulejos.



Bibliografia e outros recursos de apoio

Esta obra surge da experiência de leccionação com o Scheme, na primeira disciplina de programação do curso de informática da Faculdade de Engenharia da Universidade do Porto (FEUP), iniciada no ano lectivo 1994/95. Neste já longo percurso, alunos e docentes foram acompanhados, entre outros, por dois livros excepcionais, de inquestionável valor.

Harold Abelson, Gerald Jay Sussman, Julie Sussman
Structure and Interpretation of Computer Programs, 2nd edition
McGraw-Hill Book Company, 1985, second edition, 1996

George Springer and Daniel P. Friedman
Scheme and the Art of Programming
McGraw-Hill, Sixth printing, 1993

Outros livros merecem também uma referência, nomeadamente,

James Little, Vincent Manis
The schematics of computation
Prentice-Hall, ISBN:0-13-834284-9, 1995

e um outro por ser o primeiro publicado em português.

João Pavão Martins e Maria dos Remédios Cravo
Programação em Scheme: introdução à programação utilizando múltiplos paradigmas
IST Press, Instituto Superior Técnico, ISBN: 972-8469-32-2, 2004.

Para uma descrição mais aprofundada do Scheme, recomenda-se a consulta de

[Revised \(5\) Report on the Algorithmic Language Scheme](#)

Nesta altura, já terá visitado [PLT Scheme](#), onde encontrou o DrScheme para instalar no seu computador e muitos outros recursos para utilizar neste ambiente de programação. Entre a grande diversidade de documentação lá existente, distingue-se a publicação:

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi
How to Design Programs
An Introduction to Computing and Programming
The MIT Press, Cambridge, Massachusetts, 2003.

F. Nunes Ferreira

António Coelho

Julho - 2008