Métodos de Programação I
Projecto Prático I
Relatório de desenvolvimento

Bruno Renato Afonso Costa Duarte (43119)
Jorge Amílcar da Silva Pereira (43102)
Nuno André Job Pinto (43190)

Universidade do Minho, Departamento de Informática
Outubro/Novembro de 2005

01 introdução

Este relatório é referente à proposta que nos foi apresentada para a elaboração de um programa em *Haskell* que, através de um interpretador de comandos, permita jogar um jogo *Su-Doku*. Para além disto, foi-nos pedido o desenvolvimento de estratégias de solução que, em última instância, resolveriam na totalidade qualquer puzzle *Su-Doku*.

Este documento tem como intuito explicar a estratégia que foi escolhida para codificar esse mesmo programa, assim como as dificuldades e especificidades inerentes ao desenvolvimento deste projecto prático.

Em relação à estrutura, este documento é composto por uma introdução, uma secção de análise e especificação, uma secção de concepção da resolução e uma conclusão.

O conteúdo, por sua vez, pretende abarcar a forma como o projecto foi elaborado e os passos tomados para a concretização dos objectivos propostos.

02 descrição informal do problema

Su-Doku é um puzzle lógico cujo objectivo é preencher uma grelha 9x9 na qual cada linha, coluna e caixa 3x3 tenham todos os números entre um e nove. Inicialmente um puzzle *Su-Doku* tem algumas células já preenchidas, o que permite que, com alguma paciência e habilidade lógica, resolvamos o puzzle.

O problema foi-nos apresentado dividido em duas partes distintas: construção de um editor de jogo e codificação de estratégias de solução.

O editor de jogo é composto por um interpretador que aceita os seguintes comandos:

- **read** (Lê ficheiro com a informação de um tabuleiro *Su-Doku*).
- **show** (Visualiza o estado actual do tabuleiro).
- **set** (Permite realizar uma jogada no tabuleiro actual).
- **undo** (Permite voltar ao estado anterior à última jogada).
- **quit** (Sai do interpretador).
- **cands** (Mostra o conjunto de alternativas de jogadas admissíveis).
- **showCands** (Mostra todos os conjuntos candidatos das posições livres).

- **simplify** (Realiza todas as jogadas que não pressupõem escolhas).
- **autosolve** (Resolve o puzzle de forma automática).

03 concepção da resolução

Começámos o projecto pela questão de valorização, já que entendemos que seria muito complicado integrar uma *monad* de estado já programada com *wxHaskell*. Após o fim da codificação da parte gráfica, deparámo-nos com a aparente impossibilidade de integração da referida *monad* no código. Foi-nos então sugerido por um docente da cadeira que apresentássemos o nosso trabalho «como estava» e fizéssemos um «pequeno aparte», no qual mostrássemos os nossos conhecimentos sobre *monads*, particularmente a de estado.

O algoritmo desenvolvido para a resolução do problema foi sendo criado à medida que nos deparávamos alguns problemas. Com os testes, assim como com mais algumas horas dedicadas à investigação de procedimentos e sintaxe até então por nós desconhecidos, essas dificuldades iniciais foram deixando de o ser e, assim, passámos à codificação de uma questão extra na qual pretendemos mostrar o nosso conhecimento da *monad* de estado.

Desenvolvemos as nossas próprias rotinas de verificação da forma que achamos mais conveniente, eficaz e adequada ao problema, isto para que fosse possível uma melhor modelação do programa e um maior aprofundamento no conhecimento da linguagem em questão.

Perante a análise do problema achamos apropriado usar os seguintes tipos:

- **Linha** (Lista de inteiros).
- **Grelha** (Lista de linhas).
- **Posicao** (Tuplo de dois inteiros; x e y).
- **Posicoes** (Lista de posições).
- **Movimento** (Tuplo de uma posição e um inteiro; o valor).
- **Movimentos** (Lista de movimentos).
- **Candidato** (Tuplo de uma posição e uma linha; os valores possíveis).
- **Candidatos** (Lista de candidatos).
- **Amb** (Guarda os elementos necessários para o *wxHaskell*).

As constantes são três grelhas *Su-Doku* diferentes usadas na fase de testes do programa e a *grelhaInicial*, uma grelha vazia. A razão de existirem três grelhas para testes prende-se com o facto de todas serem diferentes: uma (*exemplo*) é um normal *Su-Doku* de jornal, outra (*infinito*) é a que foi usada para exemplificar

um *Su-Doku* com infinitas soluções, e a última (*samurai*) é o mais difícil puzzle *Su-Doku* que encontrámos na Internet[1].

04 conclusão

Terminado o projecto, podemos concluir que alcançámos os nossos objectivos - o programa funciona, aparentemente, sem qualquer falha.

A aliar ao que foi dito acima, aprendemos a usar com facilidade a *monad* de estado e IO, para além de termos aumentado exponencialmente o nosso conhecimento de programação em *Haskell*. Este tipo de tarefas promovem também a nossa capacidade de raciocínio, necessidade de cumprimento de prazos, capacidade de adaptação a diferentes situações, criatividade e aperfeiçoamento.

É também importante referir que não codificamos, pelo menos até à altura em que este relatório foi escrito, nenhuma função que resolvesse um puzzle *Samurai* (combinação de 5 puzzles sudoku). Esta foi a nossa opção visto que entendemos que seria o equivalente a reprogramar tudo e tal valorização não justificaria os esforços que puderam assim ser aplicados noutras vertentes da aplicação.

05 sites consultados

- http://www.setbb.com/phpbb/index.php?mforum=sudoku
  Fórum de programadores de *Su-Doku*.
- http://www.wikipedia.org
  Enciclopédia online gratuita.
- http://www.puzzle.jp/keys/sudoku_keys-e.html
  Estratégias de solução avançadas para puzzles *Su-Doku*.
- http://www.haskell.org/tmrwiki/SolvingSudoku
  Como resolver um puzzle *Su-Doku* em Haskell.

[1] De acordo com a opinião geral de um fórum de programadores de *Su-Doku*.