Projecto de Laboratórios de Informática I
(2ª fase*)

Sokoban em *Haskell*

2015/2016 — LEI

1 Introdução

Neste enunciado apresentam-se as tarefas referentes à segunda fase do projecto da unidade curricular de Laboratórios de Informática I. O projecto será desenvolvido pelos mesmos grupos constituídos para a primeira fase, e consiste em pequenas aplicações *Haskell* que deverão responder a diferentes tarefas (apresentadas adiante).

O tema do projecto continua a baseiar-se no puzzle *Sokoban* (http://wikipedia.org/wiki/Sokoban), sendo que nesta fase se pretende complementar as tarefas realizadas na fase anterior para produzir um programa *Haskell* com o jogo completo, incluindo uma interface gráfica. No final desta fase do projecto deverá ser possível jogar o *Sokoban* de forma análoga a quando se joga *online* em http://sokoban.info.

Recorda-se que no *BlackBoard* da disciplina, será mantida uma FAQ contendo respostas a questões e esclarecimentos que surjam ao longo do periodo de execução do projecto.

2 Gráficos em *Haskell*

Para construir a interface gráfica do projecto far-se-á uso da biblioteca *Gloss*. *Gloss* é uma biblioteca *Haskell* minimalista para a criação de gráficos 2D. Como tal, é ideal para a prototipagem de pequenos jogos de puzzle ao estilo do Sokoban.

Instalação. O *Gloss* pode ser instalado através do *Cabal*, o gestor de pacotes *Haskell* que faz parte da distribuição *Haskell Platform*. Para se instalar a biblioteca, deve-se então utilizar o comando:

```
$ cabal install gloss
```

*Última actualização: 25 de Novembro de 2015

Uma vez instalada a biblioteca, os programas *Haskell* podem realizar o `import Graphics.Gloss` necessário para utilizar a biblioteca.

Exemplo de Utilização. O tipo central da biblioteca *Gloss* é o tipo `Picture`. Ele representa uma figura 2D como seja um segmento de recta, um círculo, um polígono, ou até um *bitmap* lido de um ficheiro. A cada um destes diferentes tipos de figura irão corresponder diferentes construtores do tipo `Picture` (e.g. o construtor `Circle` para um círculo, etc. — ver documentação para consultar listagem completa dos construtores). Por exemplo, o valor *circulo* apresentado abaixo representa um círculo centrado na posição (0,0).

$$\begin{aligned} &\mathit{circulo} :: \mathit{Picture} \\ &\mathit{circulo} = \mathit{Circle}\ 50 \end{aligned}$$

Certos construtores do tipo `Picture` não representam propriamente figuras, mas antes transformações sobre sub-figuras. Por exemplo, o constructor $\mathit{Translate} :: \mathit{Float} \to \mathit{Float} \to \mathit{Picture} \to \mathit{Picture}$ permite reposicionar uma figura efectuando uma translação das coordenadas. Assim, para posicionar o circulo atrás definido num outro ponto que não a origem bastaria fazer qualquer coisa como:

$$\begin{aligned} &\mathit{outroCirculo} :: \mathit{Picture} \\ &\mathit{outroCirculo} = \mathit{Translate}\ (-40)\ 30\ \mathit{circulo} \end{aligned}$$

Outros transformações possíveis são `Scale`, `Rotate` e `Color`. Por último, podemos ainda produzir uma figura agregando outras figuras. Para tal existe o constructor $\mathit{Pictures} :: [\mathit{Picture}] \to \mathit{Picture}$, que recebe uma lista de figuras para serem desenhas sequencialmente (note que essas figuras se podem sobrepôr entre si). Segue-se um exemplo onde se explora essa possibilidade juntamente com outras transformações:

$$\begin{aligned} &\mathit{circuloVermelho} = \mathit{Color}\ \mathit{red}\ \mathit{circulo} \\ &\mathit{circuloAzul} = \mathit{Color}\ \mathit{blue}\ \mathit{outroCirculo} \\ &\mathit{circulos} = \mathit{Pictures}\ [\mathit{circuloVermelho}, \mathit{circuloAzul}] \end{aligned}$$

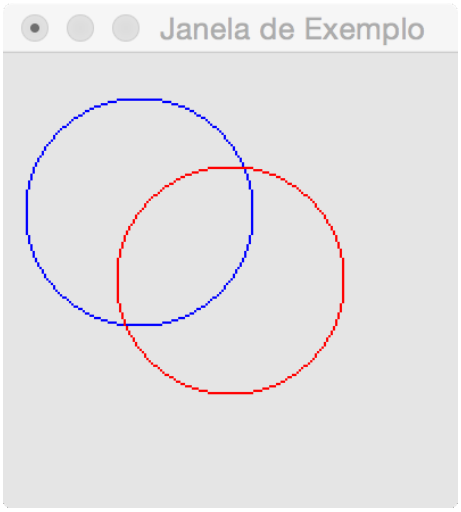
Naturalmente que o objectivo de definir figuras como valores do tipo `Picture` é podermos visualizá-las no ecrã. Para tal temos de criar uma janela *Gloss* para lá desenhar o conteúdo da figura. O fragmento de código que se segue permite visualizar a figura `circulos` definida atrás:

```
window :: Display
window = InWindow "Janela de Exemplo" -- título da janela
                  (200, 200) -- Dimensão da janela
                  (10, 10)  -- posição no ecrã

background :: Color
background = greyN 0.8

main :: IO ()
main = display window background circulos
```

De notar que a convenção no *Gloss* é que a posição com coordenadas $(0, 0)$ é o centro da janela. Assim, o resultado obtido será a janela:



Para além da visualização de gráficos 2D, a biblioteca *Gloss* incopora funcionalidades que permitem controlar eventos como teclas pressionadas ou o movimento do rato. Em anexo são incluídos exemplos adicionais que servem de ponto de partida para explorar essas funcionalidades no projecto.

A documentação da API da biblioteca encontra-se disponível no url `https://hackage.haskell.org/package/gloss`.

3 Tarefas Computacionais

A segunda fase do projecto de LI1 compreende três tarefas computacionais, identificadas pelas letras `D`, `E` e `F`.

D - Sequência de comandos

Nesta primeira tarefa ainda não se fará uso da biblioteca gráfica *Gloss*. Em vez disso, pretende-se animar o jogo *Sokoban* ainda em modo texto, extendendo o programa realizado na primeira fase para executar toda uma sequência de comandos. Concretamente, pretende-se continuar o que foi feito na tarefa C da primeira fase, mas em vez de se executar um só comando, pretende-se agora executar toda uma sequência de comandos até que o puzzle termine (ou não existam mais comandos).

Algumas considerações sobre o programa pretendido nesta tarefa:

- O formato de entrada é como definido nas tarefas da primeira fase, diferindo apenas no facto que agora existem, depois do mapa e das coordenadas do boneco/caixas, uma linha adicional contendo uma sequência não vazia de comandos (`U`, `D`, `L` ou `R`).

- O programa deverá executar os comandos em sequência.
- Se for atingida a configuração final com todas as caixas arrumadas, o programa deve imprimir uma linha contendo o texto "`FIM <tick_count>`", onde `<tick_count>` representa o número de comandos realmente executados (i.e. sem contar os comandos inválidos que não provocaram qualquer alteração de estado). Depois deve terminar independentemente de existirem, ou não, comandos ainda não processados.
- Se forem executados todos os comandos e a configuração final não for atingida, deve deve surgir o texto "`INCOMPLETO <tick_count>`" e terminar (onde `<tick_count>` representa o mesmo que no ponto anterior).

E - Dimensões de uma `Picture`

Com vista a permitir uma maior familiarização com a biblioteca *Gloss*, pretende-se realizar um pequeno exercício sobre o tipo de dados fundamental dessa biblioteca: o tipo `Picture`. O objectivo é o de determinar as dimensões do menor rectangulo envolvente de uma `Picture`.

- A figura deve ser lida de `stdin` com recurso ao método `readPicture` disponível na biblioteca `GlossExtras` (fornecida[1]). Essa biblioteca deve portanto ser importada pelo programa que realiza a tarefa.
- O programa deve ignorar os construtores `ThickCircle`, `Arc`, `ThickArc` e `Text`, já que nesses casos não é evidente qual o rectangulo envolvente da figura.
- O resultado do programa deverá ser uma única linha impressa com as dimensões do rectangulo envolvente da figura, i.e. "⟨larg⟩ ⟨alt⟩", onde ⟨larg⟩ e ⟨alt⟩ correspondem respectivamente ao arredondamento para inteiros (obtidos por usando a função `round`) da largura e altura do rectangulo envolvente, separados por um único espaço.

Observação: é possível combinar as transformações `Translate`, `Rotate`, `Scale` e `Circle` por forma a que o cálculo da respectivo rectangulo envolvente não seja imediato (pressupõe uma manipulação algébrica complexa e/ou uma estratégia de cálculo elaborada). Anuncia-se de qualquer forma que esses casos serão marginais no conjunto de testes realizados pelo sistema de submissão `mooshak` ($< 20\%$).

[1] A biblioteca está disponível no `BlackBoard` – ficheiro `GlossExtras,hs`

F - Interface gráfica

Nesta última tarefa pretende-se realizar o jogo *Sokoban* completo, tirando partido das facilidades gráficas oferecidas pela biblioteca *Gloss*.

Como ponto de partida, deve-se entender esta tarefa como oferecendo ao utilizador um meio para jogar o jogo *Sokoban* similar ao que é possível no sítio `sokoban.info`. O programa deve ler do `stdin` a descrição do mapa e posição do boneco e coordenadas das caixas, e depois abrir uma janela gráfica que torne possível interagir com o jogo associando comandos a teclas específicas.

Note no entanto que esta tarefa se trata acima de tudo de uma "tarefa aberta", onde se estimula que os alunos explorem diferentes possibilidades para melhorar o programa final. Algumas sugestões de extensões/melhoramentos:

- Considerar variantes do *puzzle* como sejam:

  Caixas leves: eliminar a restrição do boneco só conseguir empurrar uma caixa, passando a ser possível empurrar um qualquer número de caixas seguidas;

  Caixas coloridas: as caixas assim como os locais de arrumação têm associadas cores. O jogo só termina quando as caixas estiverem arrumadas nos locais da cor apropriada.

- Enriquecer as funcionalidades da interface, como seja a inclusão da capacidade de realizar **undo** (i.e. desfazer as últimas jogadas); **restart** (recomeçar o jogo); visualizar o **score** (número de jogadas já realizadas); etc.

- Considerar o carregamento de mapas com uma sequência de jogadas iniciais pré-definida, tal como definido na `Tarefa D`, e animá-las em modo de "replay" na interface gráfica.

- Considerar multiplos níveis (mapas).

Dada a natureza desta tarefa, ela não será objecto de avaliação pelo sistema de submissão online `Mooshak`.

4 Entrega e Avaliação

A data limite para entrega de todas as componentes da segunda fase do projecto é **3 de Janeiro de 2016**, e a respectiva avaliação terá um peso de 40% na nota final da UC. As tarefas computacionais D e E deverão ser submetidas na plataforma `mooshak`, sendo que estas serão desde logo objecto de uma avaliação automática por parte da plataforma (com um peso

descriminado abaixo). Cada grupo é responsável por submeter na plataforma `mooshak` unicamente programas da sua autoria[2]. A avaliação final do projecto compreende ainda uma sessão de apresentação a ser agendada para a primeira semana de aulas de Janeiro.

Para além dos programas submetidos na plataforma `mooshak`, será considerada parte integrante do projecto todo o material de suporte à sua realização depositado no repositório SVN do respectivo grupo (código, documentação, ficheiros de teste, etc.). A utilização das diferentes ferramentas abordadas no curso (como `haddock`; *SVN*; LaTeX; etc.) deve seguir as recomendações enunciadas nas respectivas sessões laboratoriais. A avaliação desta fase do projecto terá em linha de conta todo esse material, atribuindo-lhe os seguintes pesos relativos:

Avaliação automática das tarefas computacionais (D e E)	20%
Avaliação qualitativa das tarefas computacionais e do processo de desenvolvimento	40%
Relatório (e utilização do LaTeX)	30%
Utilização do SVN, testes e documentação do código	10%

A nota final é atribuída independentemente a cada membro do grupo em função da respectiva prestação.

[2]Os programas submetidos irão ser processados por ferramentas de detecção de plágio e, na eventualidade serem detectadas cópias, estas serão consideradas fraude dando-se-lhes tratamento consequente.

A Programação em *Gloss*

A.1 Interacção com o utilizador

Para facilitar na definição de um jogo interactivo em *Gloss*, sugere-se a utilização da seguinte função:

```
joga
  :: mundo                          -- Estado inicial do jogo.
  → (mundo → Picture)               -- Função que desenha o estado actual do jogo.
  → (Event → mundo → mundo)         -- Função que altera o estado do jogo.
  → IO ()                           -- Ação IO que abre o jogo numa nova janela.
```

A definição da função é apresentada na Figura 1: ela cria uma janela *Gloss* com as dimensões que se entenderem apropriadas, e invoca a função *play* do *Gloss* que inicia o ciclo de interacção. Note que é uma função que retorna *IO*, pelo que tipicamente é incluída na função `main` do programa. Note ainda que é a responsabilidade do programador definir um tipo concreto apropriado para o que se designou por *mundo*, que deve modelar todo o estado do seu jogo. O tipo de dados *Picture*, definido em Graphics.Gloss.Data.Picture, contém a representações dos gráficos suportados pelo *Gloss*, tal como formas geométricas (e.g., linhas, polígonos, círculos), texto ou ficheiros de imagens externos. O tipo de dados *Event*, definido em Graphics.Gloss.Interface.Pure.Game, modela o tipo de eventos a que o jogo pode reagir como, por exemplo, o pressionar de teclas no teclado ou o movimento do rato.

Na Figura 1 é apresentado um *template* completo que pode servir de base ao desenvolvimento de um jogo simples em *Gloss*.

A.2 Inclusão de *BitMap*s nas figuras

É possível carregar ficheiros de imagens externos no formato Windows Bitmap, com extensão `bmp`. Para tal, pode usar a função *loadBMP* :: *FilePath* → *IO Picture* disponibilizada pelo módulo Graphics.Gloss.Data.Bitmap. Como esta é uma função de I/O, deve ser executada directamente na função *main* do jogo, acrescentado o gráfico correspondente à descrição do estado do jogo:

```
type Estado = (..., Picture)
main = do
  bola ← loadBMP "bola.bmp"
  joga (..., bola) ... ...
```

Uma dica adicional é que o gloss apenas suporta ficheiros `.bmp` não comprimidos. Em sistemas Unix, pode-se utilizar a ferramenta `convert` distribuída com o ImageMagick para descomprimir um ficheiro bitmap:

```
$ convert compressed.bmp -compress None decompressed.bmp
```

```
module Main

import Graphics.Gloss                          -- interface gloss
import Graphics.Gloss.Data.Picture             -- importar o tipo Picture
import Graphics.Gloss.Interface ∘ Pure.Game    -- importar o tipo Event

main = joga mapaInicial desenhaJogo reageEvento

  -- Uma representação do estado do jogo.
type Estado = ...

  -- O estado inicial do jogo.
mapaInicial :: Estado
mapaInicial = ...

  -- Função que desenha o jogo.
desenhaJogo :: Estado → Picture
desenhaJogo = ...

  -- Função que altera o estado do jogo.
reageEvento :: Event → Estado → Estado
reageEvento = ...

joga :: mundo → (mundo → Picture) → (Event → mundo → mundo) → IO ()
joga mapa desenha reage = play
  (InWindow "Novo Jogo" (800, 600) (0, 0))   -- tamanho da janela do jogo
  (greyN 0.5)                                -- côr do fundo da janela
  45                                         -- refresh rate
  mapa                                       -- mapa inicial
  desenha                                    -- função que desenha o mapa
  reage                                      -- reage a um evento
  (λt m → m)                                 -- não reage ao passar do tempo
```

Figura 1: Template gloss.

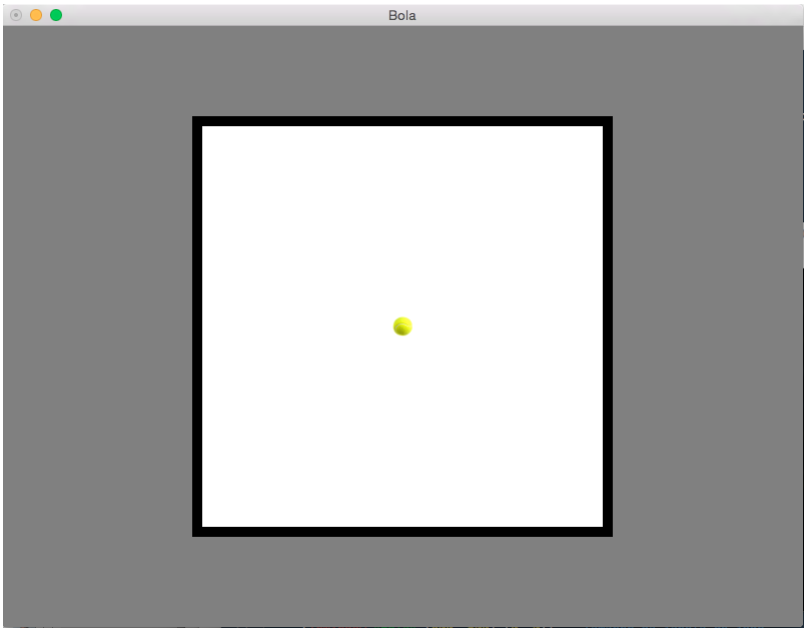


Figura 2: Janela do jogo de exemplo.

A.3 Exemplo completo

No BB, pode encontrar um exemplo de um jogo completo da utilização de *Gloss*, que consiste em desenhar uma caixa com uma bola dentro, que se pode mover livremente em 4 direções sem poder sair de fora da caixa (Figura 2).