Programação gráfica usando o Gloss

Laboratórios de Informática 1

MIEI

Para construir a interface gráfica do projecto far-se-á uso da biblioteca Gloss. O Gloss é uma biblioteca Haskell minimalista para a criação de gráficos e animações 2D. Como tal, é ideal para a prototipagem de pequenos jogos. A documentação da API da biblioteca encontra-se disponíıvel no em <https://hackage.haskell.org/package/gloss>.

## Instalação

O Gloss pode ser instalado através do utilitário cabal, o gestor de bibliotecas Haskell que faz parte da distribuição Haskell Platform. Para instalar a biblioteca deve-se então utilizar os comandos:

|  |
| --- |
| cabal update  cabal install gloss |

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

## Criação de gráficos 2D

O tipo central da biblioteca Gloss é o tipo Picture. Este permite criar uma figura 2D usando segmentos de recta, círculos, polígonos, ou até *bitmaps* lidos de um ficheiro. A cada um destes diferentes tipos de figura correspondem diferentes construtores do tipo Picture (e.g. o construtor Circle para um círculo - ver documentação para consultar listagem completa dos construtores). Por exemplo, o valor circulo definido abaixo representa um círculo de raio 50 centrado na posição (0,0).

|  |
| --- |
| circulo :: Picture  circulo = Circle 50 |

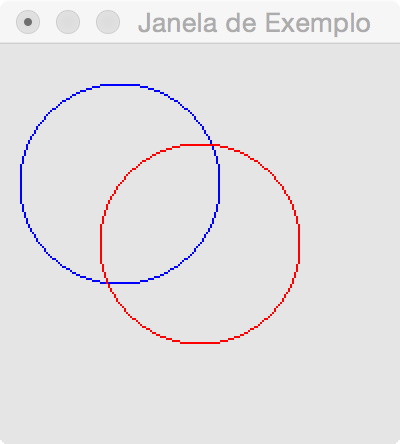
Certos construtores do tipo Picture não representam propriamente figuras, mas antes transformações sobre sub-figuras. Por exemplo, o constructor Translate :: Float -> Float -> Picture -> Picture permite reposicionar uma figura efetuando uma translação das coordenadas. Assim, para posicionar o círculo atrás definido num outro ponto que não a origem bastaria fazer algo como:

|  |
| --- |
| outroCirculo :: Picture  outroCirculo = Translate (−40) 30 circulo |

Outras transformações possíveis são Scale, Rotate e Color. Por último, podemos ainda produzir uma figura agregando outras figuras usando o constructor Pictures :: [Picture] -> Picture, que recebe uma lista de figuras para serem desenhadas sequencialmente (note que essas figuras se podem sobrepôr entre si). Segue-se um exemplo onde se explora essa possibilidade juntamente com outras transformações:

|  |
| --- |
| circuloVermelho = Color red circulo  circuloAzul = Color blue outroCirculo  circulos = Pictures [circuloVermelho, circuloAzul] |

Naturalmente que o objetivo de definir figuras como valores do tipo Picture é podermos visualizá-las no ecrã. Para tal temos de criar uma janela Gloss onde será desenhado 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 |

Para correr o programa basta compilar o ficheiro Haskell usando o ghc e correr o executável. 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 apresentada em cima.[[1]](#footnote-0)

## Programação de jogos

Para além da visualização de gráficos 2D, a biblioteca Gloss permite criar facilmente jogos simples usando a função play da biblioteca Graphics.Gloss.Interface.Pure.Game. Para usar esta função é necessário começar por definir um novo tipo Estado que representa todo o estado do seu jogo. Imagine por exemplo que o estado apenas indica a posição actual de um objecto.

|  |
| --- |
| type Estado = (Float,Float) |

Depois é necessário definir qual o estado inicial do jogo, e como é que um determinado estado do jogo será visualizado com gráficos 2D, ou seja, como se converte para um valor do tipo Picture. No nosso caso, vamos assumir que o nosso estado inicial é a posição (0,0) e que em cada instante de tempo apenas desenhamos um polígono na posição dada pelo estado actual.

|  |
| --- |
| estadoInicial :: Estado  estadoInicial = (0,0)  desenhaEstado :: Estado -> Picture  desenhaEstado (x,y) = Translate x y poligno  where  poligno :: Picture  poligno = Polygon [(0,0),(10,0),(10,10),(0,10),(0,0)] |

Para implementar a reação a eventos, nomeadamente o pressionar das teclas, é necessário implementar uma função que, dado um valor do tipo Event (definido em Graphics.Gloss.Interface.Pure.Game) e um estado do jogo, gera o novo estado do jogo. No nosso exemplo, vamos apenas alterar o estado conforme o utilizador carrega nas teclas “left”, “right”, “up”, and “down”.

|  |
| --- |
| reageEvento :: Event -> Estado -> Estado  reageEvento (EventKey (SpecialKey KeyUp) Down \_ \_) (x,y) = (x,y+5)  reageEvento (EventKey (SpecialKey KeyDown) Down \_ \_) (x,y) = (x,y-5)  reageEvento (EventKey (SpecialKey KeyLeft) Down \_ \_) (x,y) = (x-5,y)  reageEvento (EventKey (SpecialKey KeyRight) Down \_ \_) (x,y) = (x+5,y)  reageEvento \_ s = s -- ignora qualquer outro evento |

Finalmente, é necessário definir a seguinte função que altera o estado do jogo em consequência da passagem do tempo. Se o jogo estiver a funcionar a uma frame rate fr, o parâmetro n será sempre 1/fromIntegral fr. Vamos assumir para o nosso exemplo que a cada instante de tempo a posição actual é actualizada da seguinte forma:

|  |
| --- |
| reageTempo :: Float -> Estado -> Estado  reageTempo n (x,y) = (x,y-0.1) |

Para colocar todas estas peças a funcionar em conjunto basta definir um programa como o que se segue:

|  |
| --- |
| fr :: Int  fr = 50  dm :: Display  dm = InWindow "Novo Jogo" (400, 400) (0, 0)  main :: IO ()  main = do play dm -- janela onde irá correr o jogo  (greyN 0.5) -- côr do fundo da janela  fr -- frame rate  estadoInicial -- estado inicial  desenhaEstado -- desenha o estado do jogo  reageEvento -- reage a um evento  reageTempo -- reage ao passar do tempo |

## Inclusão de imagens no jogo

É possível carregar ficheiros de imagens externos no formato *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 diretamente na função main do jogo, devendo os gráficos ser incluídos no estado do jogo:

|  |
| --- |
| type EstadoGloss = (Estado, Picture)  ...    main :: IO ()  main = do p <- loadBMP “imagem.bmp”  play dm -- janela onde irá correr o jogo  (greyN 0.5) -- côr do fundo da janela  fr -- frame rate  (estadoGlossInicial p) -- estado inicial  desenhaEstadoGloss -- desenha o estado do jogo  reageEventoGloss -- reage a um evento  reageTempoGloss -- reage ao passar do tempo |

Uma dica adicional é que a biblioteca Gloss, por definição, apenas suporta ficheiros *bitmap* 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 |

Alternativamente, pode-se também utilizar o pacote gloss-juicy que suporta ficheiros de imagens genéricos. Para instalar esta extensão deve executar o comando:

|  |
| --- |
| cabal install gloss-juicy |

Imagens de formato genérico podem então ser alternativamente carregadas da seguinte forma:

|  |
| --- |
| ...  import Graphics.Gloss.Juicy  ...  do  Just img1 <- loadJuicy “imagem.jpg”  Just img2 <- loadJuicy “imagem.png”  ... |

## Funções a implementar

1. Actualize o código de tal modo que o polígono se mantenha em movimento enquanto a tecla estiver a ser pressionada. Dica: tem que estender o estado.
2. Altere o código para começar a desenhar elementos do projecto da disciplina. Por exemplo: motas, peças (com diferentes tipologias e pisos), etc. Eventualmente pense também em como desenhar a matriz do mapa.
3. Pense em como desenhar um mapa (ou uma matriz) de forma a que apareça centrado na janela. Dica: tem que ter em consideração a dimensão da janela.

1. Dependendo do sistema operativo e instalação do OpenGL, pode também ser capaz de interpretar uma aplicação gloss com o interpretador ghci utilizando a opção -fno-ghci-sandbox. [↑](#footnote-ref-0)