Universidade do Minho
Escola de Engenharia
Departamento de Informática

Projecto de Laboratórios de Informática I

Licenciatura em Engenharia Informática

1º Ano — 1º Semestre

Ano Lectivo 2022/2023

— Fase 2 de 2 —

Data de Lançamento: 13 de Novembro de 2022

Data Limite de Entrega: 02 de Janeiro de 2023

Novembro de 2022

1 Introdução

Neste enunciado apresentam-se as tarefas referentes à segunda (e última) fase do Projecto de Laboratórios de Informática I 2022/2023. O projecto será desenvolvido pelos **mesmos grupos** constituídos para a primeira fase e consiste novamente na resolução de um pequeno conjunto de *Tarefas*, a ser resolvido tendo por base o trabalho já elaborado pelo grupo durante a primeira fase.

2 Adenda à Tarefa 3

A descrição da função `animaJogo` (*cf. Tarefa 3*) no enunciado da fase anterior não foi suficientemente específica, dando azo a situações ambíguas. Para que a possa efectivamente aproveitar na elaboração das próximas tarefas, indicamos de seguida como proceder para as situações ambíguas encontradas.

2.1 Atropelamento

Considere a seguinte expressão:

```
let mapa    = Mapa 3 [(Estrada 2, [Carro, Nenhum, Nenhum])]
    jogador = Jogador (1, 0)
    jogo    = Jogo jogador mapa
in
  animaJogo jogo Parado
```

Uma vez que o jogador é atropelado durante o movimento do carro, o resultado esperado desta expressão é:

```
Jogo jogador (Mapa 3 [(Estrada 2, [Nenhum, Carro, Nenhum])])
```

correspondendo ao estado do mapa no momento em que o jogador perdeu. Por outras palavras, o movimento do obstáculo `Carro` é interrompido ao atropelar o jogador. Pertinentemente, na expressão:

```
animaJogo jogo (Move Esquerda)
```

esperamos que o resultado seja:

```
Jogo (Jogador (0, 0)) mapa
```

Note também que a interrupção de movimento após um atropelamento aplica-se a todos os obstáculos da linha. Mais concretamente, na expressão:

```
let n       = Nenhum
    mapa    = Mapa 6 [(Estrada 3, [Carro, n, Carro, n, n, n])]
    jogador = Jogador (3, 0)
    jogo    = Jogo jogador mapa
in
  animaJogo jogo Parado
```

esperamos que o resultado seja:

```
Jogo jogador
     (Mapa 6 [(Estrada 3, [n, Carro, n, Carro, n, n])])
```

Note que todos os obstáculos se deslocaram, e em apenas 1 unidade, uma vez que o jogador foi atropelado exactamente ao fim de 1 unidade de deslocamento.

2.2 Movimentos quando num tronco

Considere a seguinte expressão:

```
let mapa    = Mapa 3 [(Relva, [Nenhum, Nenhum, Nenhum])
                     ,(Rio 1, [Nenhum, Tronco, Tronco])
                     ,(Relva, [Nenhum, Nenhum, Nenhum])]
    jogador = Jogador (1, 1)
    jogo    = Jogo jogador mapa
in
  animaJogo jogo (Move Cima)
```

O resultado esperado desta expressão é:

```
Jogo (Jogador (1, 0))
     (Mapa 3 [(Relva, [Nenhum, Nenhum, Nenhum])
             ,(Rio 1, [Tronco, Nenhum, Tronco])
             ,(Relva, [Nenhum, Nenhum, Nenhum])])
```

Por outras palavras: apesar do tronco se mover uma unidade para a direita, o jogador moveu-se directamente para a posição acima. O mesmo raciocício se aplicaria caso o jogador se movesse para a posição abaixo.

Já quando o jogador se move na horizontal, a sua posição é equivalente a mover primeiro o tronco e de seguida o jogador. A título de exemplo, o resultado esperado da expressão `animaJogo jogo (Move Direita)` é:

```
Jogo (Jogador (3, 1))
     (Mapa 3 [(Relva, [Nenhum, Nenhum, Nenhum])
             ,(Rio 1, [Tronco, Nenhum, Tronco])
             ,(Relva, [Nenhum, Nenhum, Nenhum])])
```

3 Tarefas

Cada uma das seguintes tarefas deverá ser implementada num módulo (e, por conseguinte, ficheiro) próprio, à semelhança dos módulos usados para as tarefas da primeira fase.

3.1 Tarefa 5 – Deslize do mapa

O objectivo desta tarefa é implementar a função:

```
deslizaJogo :: Jogo -> Jogo
```

que, intuitivamente, faz com que a última linha do mapa desapareça, ao mesmo tempo que uma nova linha no topo do mapa seja criada. Mais concretamente:

1. A última linha do mapa deve ser removida, enquanto que, por outro lado, uma nova linha deve ser adicionada ao topo do mapa. Em particular, o mapa resultante mantém o tamanho (*i.e.* comprimento) do mapa original.

2. A coordenada y do jogador deve ser aumentada (em 1 unidade), reflectindo assim o efeito de que o jogador "ficou para trás".

3.2 Tarefa 6 – Aplicação gráfica completa

O objectivo desta tarefa consiste em aproveitar todas as funcionalidades já elaboradas nas outras *Tarefas* e construir uma aplicação gráfica que permita um utilizador jogar. Para o interface gráfico deverá utilizar a biblioteca *Gloss*[1], que terá oportunidade de estudar durante as aulas práticas.

O grafismo e as funcionalidades disponibilizadas ficam à criatividade dos alunos, sendo que, **no mínimo**, a aplicação deverá:

1. Ser capaz de gerar um mapa e permitir ao utilizador jogá-lo.

2. Implementar o efeito de deslize do mapa ao fim de um determinado tempo (à escolha do grupo.)

3. Implementar um sistema de pontuação, *e.g.* em função do tempo que o jogador está a jogar, ou quão longe o jogador já conseguiu chegar.

[1] `https://hackage.haskell.org/package/gloss`

4. Detectar quando o jogador perde e reagir adequadamente, *e.g* mostrando uma mensagem, reiniciando o jogo, *etc.*

Sugestões de funcionalidades que serão valorizadas:

- Criar um *bot*.
- Guardar um jogo em progresso (vulgo *save game*) e continuá-lo mais tarde (na mesma posição).
- Disponibilizar um editor de mapas.
- Aumentar a dificuldade do jogo (*e.g.* manipulando as velocidades) ao longo do tempo de jogo.
- Adicionar novos tipos de obstáculos com características próprias – *e.g.* uma tartaruga que emerge (e submerge) ao fim de algum tempo, permitindo ao jogador usá-la para atravessar um rio; condutores simpáticos que abrandam ou param à espera que o jogador atravesse; *etc.*
- *Etc.*

4 Entrega e Avaliação

A data limite para conclusão de todas as tarefas desta segunda fase é de **02 de Janeiro de 2023** e a respectiva avaliação terá um peso de **55%** na nota final da UC. A submissão será feita automaticamente através do GitLab onde, nessa data, será feita uma cópia do repositório de cada grupo, sendo apenas consideradas para avaliação os programas e demais artefactos que se encontrem no repositório nesse momento. O conteúdo dos repositórios será processado por ferramentas de detecção de plágio e, na eventualidade de serem detectadas cópias, estas serão consideradas **fraude** dando-se-lhes tratamento consequente.

Para além dos programas *Haskell* relativos às tarefas será considerada parte integrante do projecto todo o material de suporte à sua realização armazenado no repositório do respectivo grupo (código, documentação, ficheiros de teste, *etc.*). A utilização das diferentes ferramentas abordadas no curso (como *Haddock* ou *git*) 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:

Componente	**Peso**
Avaliação automática e qualitativa das Tarefas 5, 6, e de valorização	80%
Documentação do código	10%
Quantidade e qualidade dos testes	5%
Utilização do sistema de versões	5%

A nota final será atribuída **independentemente** a cada membro do grupo em função da respectiva prestação. A avaliação automática será feita através de um conjunto de testes que não serão revelados aos grupos. A avaliação qualitativa incidirá sobre aspectos da implementação não passíveis de serem avaliados automaticamente (como a estrutura do código ou elegância da solução implementada).

A avaliação global do projecto incluirá uma discussão do trabalho realizado, numa sessão presencial, que poderá incluir (re)escrita de código.