Tanks

Enunciado da 2ª fase do projeto de LI1 2018/19

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 por grupos de 2 elementos, e consiste em pequenas aplicações em Haskell que deverão responder a diferentes tarefas (apresentadas adiante).

O objetivo do projeto deste ano é implementar um jogo 2D de combates entre pequenos tanques. A ideia geral do jogo é disparar tiros de forma a atingir os tanques adversários, destruindo inimigos e obstáculos no mapa. Os tanques e os tiros deslocam-se numa grelha, existindo diferentes tipos de tiros.

Tarefas

Importante: Cada tarefa deverá ser desenvolvida num módulo Haskell independente, nomeado `Tarefa`***`n`***`_2017li1g`***`xxx`***`.hs`, em que ***`n`*** é o número da tarefa e ***`xxx`*** o número do grupo, que estará associado ao repositório SVN de cada grupo. Os grupos **não devem alterar** os nomes, tipos e assinaturas das funções previamente definidas, sob pena de não serem corretamente avaliados.

Tarefa 4 - Reagir à passagem do tempo

O objectivo desta tarefa é **calcular o efeito da passagem de um instante de tempo** num estado do jogo.

Como intuição: os efeitos principais da passagem do tempo, definido pelo tipo `Ticks`, são a movimentação dos `Disparos` em curso, atualização do número de vidas dos `Jogadores` e a destruição de peças `Bloco Destrutivel`. Os detalhes são apresentados de seguida, devendo ser lidos **cuidadosamente**.

Relembrando, existem 3 tipos de `Disparos`:

- `DisparoLaser`: disparos que se expandem/propagam instantaneamente causando uma explosão[1] em todos os objectos no seu caminho. Disparos do tipo laser apenas são propagados até colidirem um `Bloco Indestrutivel`. Nota: Um disparo laser pode causar múltiplas explosões em simultâneo, diminuindo portanto o número de vidas de todos os tanques atingidos e destruindo todos os blocos destrutíveis e balas de canhão com os quais intersecta.

- `DisparoCanhao`: disparos que avançam uma posição por instante de tempo explodindo nos objetos nos quais embatem. Notas: um disparo do tipo canhão explode apenas no primeiro objecto com o qual colide sendo eliminado após essa explosão; um tanque atingido por uma bala de canhão ficará com o seu número de vidas decrementado, blocos destrutíveis são destruídos, blocos indestrutíveis não são afectados, outras balas de canhão serão destruídas.

- `DisparoChoque`: disparos que impedem outros jogadores de se deslocarem durante um dado período de tempo.

Colisões entre Disparos

Como os disparos interagem entre si e há exatamente 3 tipos de `Disparos`, assume-se por uma questão de conveniência que o efeito dos disparos no estado do jogo deve ser **processado pela seguinte ordem**:

1. Efeitos de `DisparoLaser` no estado (função `tickLasers`);
2. Efeitos de `DisparoCanhao` no estado (função `tickCanhoes`);
3. Efeitos de `DisparoChoque` no estado (função `tickChoques`).

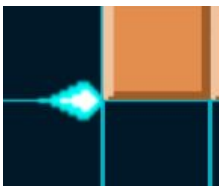
[1] Ler secção “Explosão em elementos”

Raios Laser

- O processamento de um disparo do tipo laser pode ser decomposto em 2 fases: *fase de expansão*, em que o laser é expandido até ao primeiro bloco indestrutível encontrado; *fase de explosão*: em que são processadas as explosões a ocorrer.
- Quando acontece uma colisão entre 2 ou mais lasers, os lasers envolvidos continuam a sua trajectória, não sendo afectados por este evento.
- **Nota importante:** Ler último item da próxima seção, Balas de canhão, pois contém nota informativa que também se aplica aos lasers.

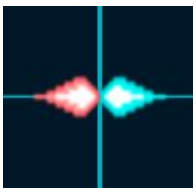
Balas de canhão

- Antes de mover a bala, deve ser verificado se esta deve explodir (ler item seguinte): se sim, explode(m) também o(s) elemento(s) com os quais colide e é removida.
- Deve explodir quando está junto a uma parede (1), de um jogador (2), de uma outra bala (3), ou então se está “de costas” para uma bala que acabou de passar (4). (Nota: balas na mesma posição que estejam na mesma direção ou sejan prependiculares também explodem.)
- Se não explode, então movimenta-se para a posição seguinte (independentemente do que lá está).
- Nota: no caso da figura (1), se existisse uma parede por baixo daquela que é apresentada, ambas seriam afetadas pela explosão da bala de canhão. De forma similar, 2 tanques podem ser atingidos pela mesma bala de canhão. **Esta nota é também válida para os raios laser.**

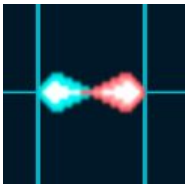


(1)

(2)



(3)



(4)

Disparos de choque

- Se o contador de ticks está a zero, o disparo de choque é removido. Caso contrário, o contador é decrementado.
- Disparos de choque que se sobrepõem não se afetam.

Explosão em elementos

- Se um disparo (laser/canhão) explode num jogador, este decrementa o seu número de vidas até que este seja igual a 0.
- Se um disparo (laser/canhão) explode num bloco destrutível este é destruído (removido do mapa).
- Os blocos indestrutíveis não são afetados por explosões.
- Se uma explosão ocorre numa bala de canhão, esta é removida.
    - Observação 1: Garanta que para o caso apresentado nas figuras 3 e 4 ambas as balas devem ser removidas.
    - Observação 2: Disparos do tipo Laser fazem explodir balas de canhão.
- Se uma explosão ocorre num disparo laser, nada acontece.

Funções a implementar

O objectivo desta tarefa é definir as funções

```
tickLasers  :: Estado -> Estado
tickCanhoes :: Estado -> Estado
tickChoques :: Estado -> Estado
```

que avançam o estado do jogo em exatamente um instante de tempo, retornando o estado correspondente ao próximo instante de tempo. Consulte a documentação online da Tarefa 4 (https://li1.lsd.di.uminho.pt/doc/src/Tarefa4_2018li1g000.html) para mais detalhes.

Tarefa 5 - Implementação do Jogo em `Gloss`

O objectivo desta tarefa é implementar o jogo completo usando a biblioteca `Gloss`. Um breve tutorial de introdução a esta biblioteca do Haskell pode ser encontrado no site da disciplina (https://docs.google.com/document/d/1C4hhip3AK_zy8QeSv58vH_yTnosFYbbseCbOw8al3qI/edit?usp=sharing). Como ponto de partida deve começar por implementar uma versão com uma visualização gráfica simples. Apesar de dever ser construída inicialmente sobre as tarefas anteriores, esta tarefa trata-se no entanto acima de tudo de uma "tarefa aberta", onde se estimula que os alunos explorem diferentes possibilidades extra para melhorar o aspecto final e jogabilidade do jogo. Sugestões de extras incluem, por exemplo:

- Gráficos visualmente apelativos (como, por exemplo, 2.5D);
- Suportar diferentes câmaras ou perspectivas;
- Menus de início e fim do jogo;
- Mostrar informação sobre o estado do jogo;
- Mostrar o tempo e o número de objetos destruídos por cada jogador;
- Permitir jogar diferentes mapas e/ou carregar mapas definidos pelo utilizador (Tarefa 1);
- Permitir jogar contra outros jogadores e contra *bots* (robôs da Tarefa 6);
- Suportar diferentes armas com diferentes propriedades;
- Suportar novas funcionalidades dos tanques ou dos mapas.

No cerne desta tarefa estão as funções da Tarefa 2 (`jogada`), que permite aos jogadores efetuarem jogadas, e da Tarefa 4 (`tick`), que faz evoluir o jogo ao longo do tempo. Note que apesar do tipo interno do `Estado` do jogo ser usado por estas funções, é provável que necessite de criar um novo tipo que contenha informação adicional relevante para a execução do jogo (denominado pelo nome `EstadoGloss` no guião `Gloss` da disciplina). O tipo `Estado` definido no módulo `LI11819` deve no entanto permanecer inalterado.

Tarefa 6 - Implementar um Robô

O objectivo desta tarefa é implementar um robô que jogue Tanks automaticamente. A estratégia de jogo a implementar fica ao critério de cada grupo, sendo que a avaliação automática será efectuada colocando o robô implementado a combater com diferentes robôs de variados graus de "inteligência".

Em cada instante, o robô tem apenas conhecimento do estado atual do jogo, e pode tomar uma única decisão usando o tipo `Jogada` definido anteriormente. Para definir a estratégia, deve assumir que:

- O jogo **acaba passados 200** `Ticks`, ganhando o jogador **com mais pontos**.
- Um jogador ganha:
    - 1 ponto por cada parede destrutível que destrua;

    - 4 pontos por cada vida que retire a outro jogador;
    - 4 pontos por cada vida que ainda possua no final do jogo; um jogador com 0 vidas pontua na mesma.
- O robô recebe sempre um `Estado` do jogo válido de acordo com as tarefas anteriores.
- Cada jogador começa com 6 vidas, 3 disparos do tipo choque e 3 disparos do tipo laser.
- Existe a possibilidade de empates.

O objectivo desta tarefa é definir a função

```
bot :: Int -> Estado -> Maybe Jogada
```

que dado o identificador de um jogador e um `Estado` do jogo, devolve uma possível `Jogada` a realizar pelo robô. O tipo `Maybe` modela a possibilidade de o robô ficar parado. Consulte a documentação online da Tarefa 6 para mais detalhes:

https://li1.lsd.di.uminho.pt/doc/src/Tarefa6_2018li1g000.html

Relatório

Nesta fase deve ser escrito um relatório, dividido em 3 partes, incidindo sobre a realização das Tarefas 3, 5 e 6. Estas 3 partes do relatório deverão ser escrito com recurso ao Haddock, como comentários do código relativo às próprias tarefas. A documentação gerada por cada uma destas 3 tarefas deverá ser organizada em secções, e deverá ter *pelo menos* as seguintes secções:

- Introdução - Descrever sumariamente o desafio e os resultados;
- Objetivos - Indicar claramente, e por palavras vossas, as estratégias utilizadas e objetivos finais da tarefa;
- Discussão e conclusão - Sumariar o principais resultados obtidos.

Para mais detalhes sobre como deve elaborar um relatório técnico de um trabalho prático consulte http://sweet.ua.pt/pf/Documentos/Guia%20redaccao%20relatorios.pdf.

Dica: Para adicionar imagens à documentação Haddock (e visualizar as mesmas correctamente na página da disciplina), deve criar uma pasta `images` na raiz do seu repositório SVN, e utilizar caminhos relativos para a mesma. Por exemplo, guardar uma imagem em `images/exemplo.png`, e na documentação do ficheiro `src/Tarefa3_2018li1gXYZ.hs` incluir essa mesma imagem com a sintaxe `<<images/exemplo.png título>>`.

Sistema de *Feedback*

O projecto inclui um Sistema de *Feedback*, também alojado em http://li1.lsd.di.uminho.pt que visa fornecer suporte automatizado e informações personalizadas a cada grupo e simultaneamente incentivar boas práticas no desenvolvimento de *software* (documentação,

teste, controle de versões, etc.). Esta página *web* é actualizada regularmente com o conteúdo de cada repositório SVN e fornece informação detalhada sobre vários tópicos, nomeadamente:

- Resultados de testes unitários, comparando a solução do grupo com um oráculo (solução ideal) desenvolvido pelos docentes;
- Relatórios de ferramentas automáticas (que se incentiva os alunos a utilizar) que podem conter sugestões úteis para melhorar a qualidade global do trabalho do grupo;
- Visualizadores gráficos de casos de teste utilizando as soluções do grupo e o oráculo.

As credenciais de acesso ao Sistema de *Feedback* são as mesmas que as credenciais de acesso ao SVN.

Para receber *feedback* sobre a Tarefa 4 e qualidade dos testes, deve ser declarada no ficheiro correspondente à tarefa as seguinte lista de testes:

```
testesT4 :: [Estado]
testesT4 = [...]
```

Estas definições podem ser então colocadas no ficheiro correspondente da Tarefa 4 para que sejam consideradas pelo Sistema de *Feedback*. As tarefas 5 e 6, sendo abertas, não requerem a definição de testes. Note que uma maior quantidade e diversidade de testes garante um melhor *feedback* e ajudará a melhorar o código desenvolvido.

Entrega e Avaliação

A data limite para conclusão de todas as tarefas desta primeira fase é **28 de Dezembro de 2018 às 23h59m59s (Portugal Continental)** e a respectiva avaliação terá um peso de 50% na nota final do projeto. A submissão será feita automaticamente através do SVN: nesta 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 3 tarefas, será considerada parte integrante do projeto todo o material de suporte à sua realização armazenado no repositório SVN do respectivo grupo (código, documentação, ficheiros de teste, relatório, etc.). A utilização das diferentes ferramentas abordadas no curso (como Haddock, SVN, 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:

Componente	**Peso**

Avaliação automática da Tarefa 4	15%
Avaliação qualitativa da Tarefa 5	25%
Avaliação automática e qualitativa da Tarefa 6	25%
Qualidade do código	10%
Utilização do SVN e testes	10%
Relatório sob a forma de documentação Haddock do código	15%

A avaliação automática será feita através de um conjunto de testes que **não** serão revelados aos grupos. A avaliação automática da Tarefa 6 consiste em correr os robôs dos alunos contra robôs de "inteligência" variada. Os alunos deverão ter atenção ao tempo de execução das soluções visto que *timeouts* resultam na derrota nesse jogo. A avaliação qualitativa incidirá sobre aspectos de qualidade de código (por exemplo, estrutura do código, elegância da solução implementada, etc.), qualidade dos testes (quantidade, diversidade e cobertura dos mesmos), documentação (estrutura e riqueza dos comentários) e bom uso do SVN como sistema de controle de versões. A avaliação do relatório incidirá sobre a sua estrutura e organização de ideias, e sobre a diversidade de operadores sintáticos do Haddock que conseguir demonstrar.