ExciteBike

Enunciado da 2ª fase do projeto de LI1 2019/20

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 corridas entre motas. A ideia geral do jogo é percorrer a pista o mais rápido possível, realizando saltos e evitando obstáculos.

Tarefas

Importante: Cada tarefa deverá ser desenvolvida num módulo Haskell independente, nomeado `Tarefa`***`n`***`_2019li1g`***`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. Os *inputs* serão um número real positivo que denota o tempo que passou desde a última atualização (ou seja, o período de tempo para o qual devem reagir), o mapa do jogo, e o jogador (de tipo `Jogador`, já apresentado na 1ª fase) para o qual devem processar o passar do tempo. O resultado deverá ser o jogador após ter sido atualizado. Assume-se que neste instante de tempo **o mapa está fixo**, i.e., o mapa não sofre alterações por parte de qualquer jogador.

Esta tarefa deve ser dividida em duas componentes: 1) uma que calcula a **nova velocidade do jogador** tendo em conta a sua posição actual no mapa, o seu estado, e velocidade actual; e 2) outra que usa a velocidade calculada para **movimentar o jogador no mapa**. Passemos a analisar estas duas componentes em detalhe.

Atualizar a velocidade do jogador

A atualização da velocidade do jogador está dividida em dois casos: 1) quando o jogador está no chão e 2) quando o jogador está no ar. No primeiro caso, a nova velocidade (v') do jogador é calculada através da seguinte fórmula:

$$v' = v + (accelMota - atrito * v) * t$$
$$accelMota = if\ (v < 2\ and\ accelJogador)\ then\ 1\ else\ 0$$

em que v é a velocidade inicial, $atrito$ é o atrito resultante da posição actual do jogador no mapa, t é o período de tempo (em segundos) em questão, e $accelJogador$ corresponde ao campo `aceleraJogador` do estado `Chao`. A correspondência entre valores de atrito e pisos é dada na tabela seguinte:

Piso	Atrito
`Terra`	0.25
`Relva`	0.75
`Lama`	1.50
`Boost`	-0.50
`Cola`	3.00

N.B. A velocidade do jogador no chão **nunca** poderá ser negativa. Para obedecer a esta condição, a velocidade a retornar deve ser **0** sempre que a velocidade calculada (como descrito acima) **for negativa**.

No segundo caso (i.e., quando o jogador está no ar) além da nova velocidade, é preciso atualizar a velocidade causada pela gravidade. A nova velocidade do jogador (v') é calculada da seguinte maneira:

$$v' = v - (resistenciaAr * v * t)$$

em que v é a velocidade inicial, $resistenciaAr$ é o resistência do ar, e t é o instante de tempo em questão. A resistência do ar é `0.125`.

N.B. A velocidade do jogador no ar **nunca** poderá ser negativa. Para obedecer a esta condição, a velocidade a retornar deve ser **0** sempre que a velocidade calculada (como descrito acima) **for negativa**.

A nova velocidade causada pela gravidade do jogador (g') é calculada da seguinte maneira:

$$g' = g + accelGravidade * t$$

Em que g é a velocidade causada pela gravidade atual do jogador, t é o instante de tempo em questão, e $accelGravidade$ é aceleração oferecida pela gravidade, que é definida como `1`.

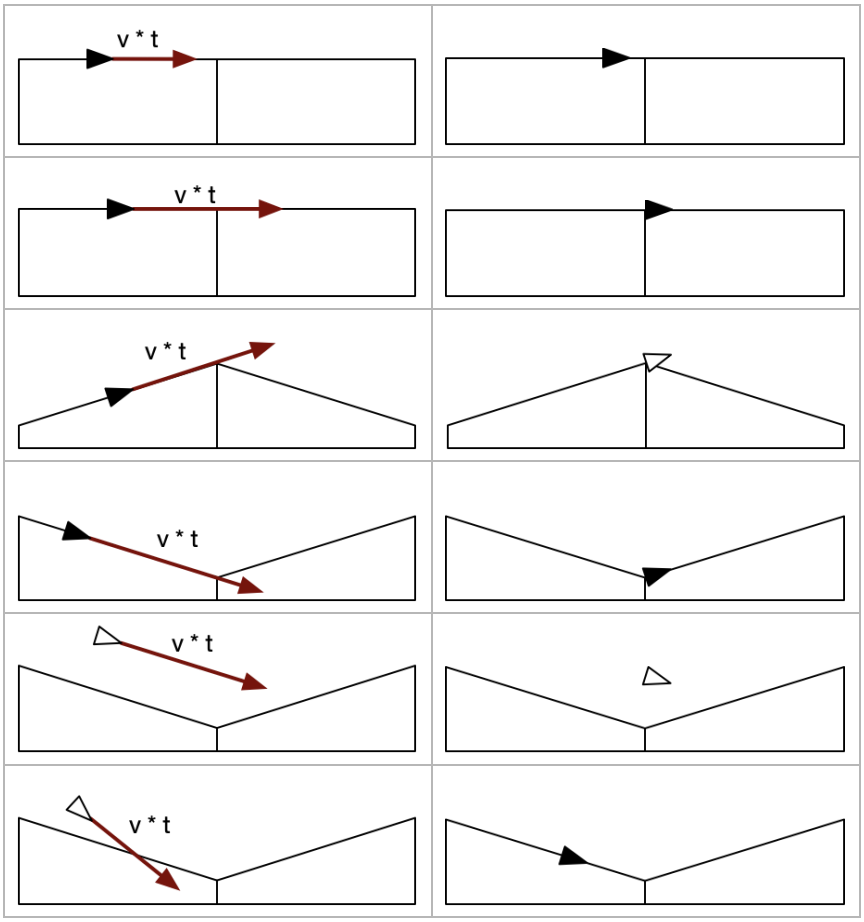
Movimentar o jogador

Após a actualização da velocidade do jogador e a sua gravidade, temos que movimentar o jogador no mapa de acordo com a nova velocidade. O jogador movimenta-se pelo período de tempo t em questão ou até bater no chão ou chegar ao fim da peça (note que, como simplificação, o resto do tempo é "desperdiçado"). Em detalhe, a tarefa de movimentar o jogador divide-se nos seguintes casos, para cada estado inicial do jogador:

1. `Morto`: se a diferencia do `timeoutJogador` menos o tempo t for maior que `0`, apenas o devemos decrementar pelo tempo t em questão. Caso contrario, o estado do jogador é alterado para `Chao False` mantendo a velocidade a zero.
2. `Chao`: movimenta-se com velocidade constante na superfície da peça pelo período de tempo t em questão ou até chegar ao limite desta. Se atingir o limite da peça, o jogador passa para a próxima peça e fica no `Chao` com o mesmo estado de `aceleraJogador` (se a inclinação da próxima peça for maior ou igual que a inclinação da peça atual) ou no `Ar` com `inclinacaoJogador` igual à direção da peça atual e velocidade `gravidadeJogador` a zero (se a inclinação da próxima peça for menor que a inclinação da peça atual). Em qualquer dos casos, `velocidadeJogador` é preservada.
3. `Ar`: movimenta-se com velocidade constante pelo período de tempo t ou até chegar ao limite da peça (no eixo dos X), mantendo-se no `Ar`, ou bater contra o chão. O jogador morre ao bater no chão se a diferença entre a `inclinacaoJogador` e a inclinação do chão for ≥45º, ficando `Morto` com `timeoutJogador` de `1`s e com a

`velocidadeJogador` a zero. Caso não morra, o jogador fica no `Chao False` com a mesma componente da velocidade que trazia paralela à pista.[1]

A tabela em baixo exemplifica algumas destas transições, em que um triângulo preenchido denota uma mota no `Chao` e um não preenchido uma mota no `Ar`, e o vetor vermelho denota o potencial movimento da mota para a velocidade atual v e o período de tempo t.



Funções a implementar

O objectivo desta tarefa é definir as funções

```
acelera :: Double -> Mapa -> Jogador -> Jogador
move    :: Double -> Mapa -> Jogador -> Jogador
```

que, para um dado mapa, alteram o estado de um jogador durante um dado período de tempo. Estas duas funções são combinadas na função pré-definida:

```
passo :: Double -> Mapa -> Jogador -> Jogador
```

[1] Note que, para este último caso de cálculo da nova velocidade, não deve considerar o efeito da gravidade.

Para mais detalhes, consulte a documentação online da Tarefa 4.

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. 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 acima de tudo de uma "tarefa aberta", onde se estimula que os alunos explorem diferentes possibilidades para melhorar o aspecto final e jogabilidade do jogo. Sugestões de extras incluem, por exemplo:

- Gráficos visualmente apelativos;
- Suportar diferentes câmaras ou perspectivas[2];
- Menus de início e fim do jogo;
- Mostrar informação sobre o estado do jogo (como o tempo, ou as munições);
- 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);
- Editor de mapas (e.g., baseado nas instruções da Tarefa 3);
- Gravar/carregar mapas comprimidos (e.g. usando as funções `constroi`/`desconstroi` da Tarefa 3)
- Suportar diferentes armas com diferentes propriedades;
- Suportar novas funcionalidades das motas ou dos mapas.

Mais exemplos de extras podem ser encontrados em trabalhos de alunos de anos anteriores.

No cerne desta tarefa estão as funções da Tarefa 2 (`jogada`), que permite aos jogadores efetuarem jogadas, e da Tarefa 4 (`passo`), 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 cada grupo necessite de criar um novo tipo que contenha informação adicional relevante para a sua implementação do jogo (denominado pelo nome `EstadoGloss` no guião `Gloss` da disciplina). O tipo `Estado` definido no módulo `LI11920` deve no entanto permanecer inalterado.

Tarefa 6 - Implementar um Robô

O objectivo desta tarefa é implementar um robô que jogue *ExciteBike* 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".

[2] O visualizador *online* utiliza uma perspectiva 2.5D, o que por si só conta uma **valorização**, mas existem muitas outras perspectivas possíveis, incluindo soluções mais simples como ver sempre o mapa por cima ou de lado.

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 40 segundos, ganhando o jogador que percorreu primeiro a maior distância, ou quando pelo menos um jogador chega ao fim da pista.
- O robô recebe sempre um `Estado` do jogo válido de acordo com as tarefas anteriores, e executa uma jogada a cada 0.2 segundos. A jogada é processada no sistema de avaliação usando os oráculos das funções `jogada` e `passo` das tarefas anteriores, pelo que os alunos podem usar as suas próprias versões para simular as consequências das jogadas.[3][4]
- Cada jogador começa com 4 munições de cola em `colaJogador`.
- 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.

Relatório

Nesta fase deve ser escrito um relatório, dividido em 3 partes, com maior ênfase sobre a realização das Tarefas 3, 5 e 6. O relatório deverá ser escrito com recurso ao Haddock, sob a forma de comentários por cada tarefa. A documentação gerada para cada tarefa deverá ser organizada em seções, incluindo *pelo menos* as seguintes seções:

- 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 os principais resultados obtidos.

Para mais detalhes sobre como deve elaborar um relatório técnico de um trabalho prático consulte o seguinte guião.

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

[3] Ao invés da Tarefa 4, o tempo não é "desperdiçado" após uma colisão ou atingindo o limite de uma peça, continuando a mota a movimentar-se durante o tempo restante. Para simular tal comportamento, pode estender a função `passo :: Double -> Mapa -> Jogador -> (Jogador,Tempo)` para retornar também o tempo restante.

[4] Cada jogada pode demorar um tempo máximo de execução 1 segundo.

`images/exemplo.png`, e na documentação do ficheiro `src/Tarefa3_2019li1gXYZ.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 :: [(Double,Mapa,Jogador)]
testesT4 = [...]
```

Estas definições devem 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 **27 de Dezembro de 2019 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 na disciplina (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 desenvolvidos pelos docentes. Os alunos deverão ter atenção ao tempo de execução das soluções visto que *timeouts* na computação de jogadas 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.