Avaliação Prática de PLC (IF686)

Data: 16/04/2013. Início: 13h00m. Término: 16h00m. Prof.: Fernando Castor

Instruções. Este exercício deverá ser resolvido individualmente. Não é permitido nenhum tipo de comunicação, verbal ou eletrônica, com colegas. Também está proibida a realização de qualquer atividade que demande o uso de redes de comunicação, locais ou de grande área. São permitidos apenas o uso do compilador GHC (ou do interpretador GHCI) e de um editor de texto. Qualquer outra ferramenta, incluindo navegadores Web, ferramentas para leitura de email, ferramentas de ajuda (*help*) e exemplos de código em Haskell são proibidas. Alunos flagrados usando qualquer uma das ferramentas/tecnologias proibidas receberão nota 0,0 (ZERO) na primeira parte da avaliação prática.

Ao final da avaliação, a solução para o exercício deverá estar implementada em um arquivo chamado `<login_do_aluno>.hs`. Por exemplo, para o aluno Fernando José Castor de Lima Filho, o nome do arquivo com a solução deverá ser `fjclf.hs`. Solicita-se que, no momento em que o professor e os monitores declararem que a avaliação terminou, toda a atividade se encerre **imediatamente**. Nesse momento, o professor e os monitores passarão com um *pen drive* copiando as soluções. É importante enfatizar que código que não compila não implica necessariamente em uma questão completamente perdida. As soluções serão corrigidas não apenas com base nos resultados que produzem, mas também no que pode ser inferido, a partir do código, sobre o raciocínio dos alunos.

Exercício. Exceto pelo clarão esporádico de um relâmpago, tudo está escuro. Não é possível ver a lua nem as estrelas. A densa floresta forma uma barreira intransponível. Todos parecem afastados por uma sensação pairante de calamidade iminente. Apenas um grupo de monges percorre o caminho de volta para seu monastério.

Após muito caminharem, eles avistam algumas luzes à distância. "Lá deve estar o vilarejo e de lá chegamos ao monastério,"um deles comenta. Ao se aproximarem, porém, percebem o erro. As luzes eram subproduto de aparições demoníacas, conjuradas pelo acúmulo de energia negativa no local. É tarde demais para voltar atrás e, antes que possam pensar duas vezes, eles estão cercados por um grupo de monstros que bloqueia sua rota de fuga. Não resta escolha exceto tentar abrir o caminho à força.

Cada monge escolhe aleatoriamente um monstro para acertar e cada golpe desferido por um monge remove 1.000 pontos de vida do monstro atingido. Os golpes de cada monge são tão rápidos que nenhum ponto de vida é regenerado enquanto um golpe é executado. Cada monge consegue executar 1 golpe a cada 2 segundos.

Monges também são misericordiosos. Se um monge decide golpear um monstro que está com menos pontos de vida do que o impacto de sua força, ele espera pacientemente o monstro se regenerar o suficiente antes de agir, sem mudar de monstro. A cada segundo, cada monstro regenera entre 20 e 200 pontos de vida.

No total temos, então, o embate entre **8 monges** e **10 monstros**. Todos os monstros possuem inicialmente **10.000 pontos de vida**, e podem ter no máximo **20.000**. O mínimo de pontos de vida para um monstro é **0**. A simulação deve terminar quando, no total, exatamente **2.000.000** pontos de vida tenham sido regenerados[1][2] entre todos os monstros (não computados pontos que seriam regenerados se o monstro já não estivesse com o máximo de pontos de vida).

Em sua solução:.

- Todos monstros serão representados por valores do tipo `TVar Int`.
- A quantidade total de pontos de vida regenerados (soma dos pontos regenerados de todos os monstros) também deve ser armazenada em um `TVar Int`.
- Não é permitido usar variáveis mutáveis, `MVars`, exceto por uma para controlar o momento em que a simulação termina.
- Cada monge executa em uma *thread* separada.
- Os pontos de vida de cada monstro são regenerados por uma *thread* separada. Você pode usar uma única *thread* que regenera todos os monstros, um de cada vez, ou uma *thread* por monstro.
- Pontos de vida são regenerados atomicamente. Ou seja, se 45 pontos de vida serão adicionados, novos pontos só serão computados um segundo depois.
- A biblioteca de Haskell para gerar números aleatórios é `System.Random`. Para este exercício, será necessária apenas a função `randomRIO :: Random a => (a, a) -> IO a`, que gera um número aleatório na mônada `IO` e dentro de uma faixa de valores, por exemplo, a expressão `(randomRIO (0,9) :: IO Int)` produz um número aleatório entre 0 e 9 dentro da mônada `IO`.
- A função de Haskell para fazer uma *thread* esperar é `threadDelay :: Int -> IO ()`, onde seu parâmetro corresponde ao número de milissegundos pelo qual a *thread* esperará.

[1] Este é o ponto em que os monges percebem que os monstros são imortais e que eles não têm chance. Essa observação faz com que os bravos monges decidam bater em retirada.

[2] Devido à sua imortalidade, os monstros nem se dão ao trabalho de atacar os monges. Ao invés disso, apenas esperam pacientemente que eles desistam e fujam, para poder continuar a carnificina.

- Seu programa deve ter uma função `main()`, deve seguir todas as recomendações apresentadas acima, compilar corretamente no GHC passando-se para ele as opções `--make -threaded -rtsopts`, não deve entrar em *deadlock*, não deve ter condições de corrida, não deve violar restrições de atomicidade e deve evitar que inconsistências ocorram, por exemplo, que sejam regenerados mais ou menos que 2.000.000 pontos de vida no total ou que um monstro tenha pontos de vida negativos ou mais que 20.000.