Exercício Programa 2 - Parte 1 - Guia do Desenvolvedor

19 de maio de 2013

1 Introdução

Integrantes:

- Victor Sanches Portella - Nº USP: 7991152
- Mateus Barros Rodrigues - Nº USP: 7991037
- Lucas Dário - Nº USP: 7991152

2 Módulos e Tipos

Em nosso programa, criamos os seguintes tipos de primeira classe:

- **Terreno** Usado para representar cada "pixel"do jogo, contendo como informação seu tipo (Água ou Terra) e a velocidade.
- **linhaT:** Usado para representar cada linha do rio. Ela possui a sequência de terrenos que a formam, e indicação de suas margens e tamanho da linha.
- **Rio:** Representa o Rio em si, com seu fluxo de água, a sequência de linhas que formam o rio, o número de colunas de pixels e o tamanho mínimo que o rio deve ter de água.
- **List:** Representa uma lista duplamente encadeada circular com cabeça que opera com objetos do tipo Item[1].

[1] Item foi definido como um vetor de objetos do tipo linhaT, mas pode ser facilmente mudado no cabeçalho desse módulo

Módulos usados em nosso programa:

- **utils.h** Biblioteca que contém funções de uso geral.
- **terreno.h** Biblioteca que contém funções para manipulação de objetos do tipo Terreno.
- **testes.h** Biblioteca que contém as funções para execução de testes de certos parâmetros do rio em determinadas condições.
- **linhaT.h** Biblioteca para manipulação de objetos do tipo linhaT.
- **list.h** Biblioteca para manipulação de objetos do tipo List.

Agora, daremos uma explicação sobre o funcionamento dos tipos e das principais funções presentes nos módulos já mencionados.

2.1 Terreno

Terreno é o objeto que usamos para representar cada pixel. Ele é definido por um tipo e uma velocidade. Por definição, todo bloco de terra tem velocidade 0. Além disso, o tipo do bloco será um caractere, que é o mesmo que será impresso na tela. Por isso, temos os seguintes defines no nosso terreno.h:

AGUA	.
TERRA	#

2.2 linhaT

O tipo linhaT é usado para representar uma linha do rio, ou seja, uma sequência ordenada de objetos do tipo Terreno. Nela, temos as informações de tamanho da linha, as margens direita e esquerda e uma flag para indicar se a linha tem ou não um ostáculo.

As funções **setFluxo** e **getFluxo** funcionam de forma a executar o processo de normalização explanado no enunciado do exercício.

Já função **igualaFluxo** recebe dois objetos do tipo linhaT. A primeira linha será usada como base para a segunda linha. Com isso, definiremos as velocidades de cada Terreno da linha 2 baseado nas velocidades da linha 1. Para isso, definimos temporariamente o fluxo da linha 1 como 1. Com isso, aplicamos a seguinte regra:

- Caso o terreno da linha 2 esteja adjacente a alguma terra, a velocidade desse terreno é automaticamente 0, independentemente das outras regras.

- Caso o terreno da linha 1 fosse terra e se transformou em água na linha 2, a velocidade dessa água será 0.1.

- Caso o terreno da linha 1 erá água e continuou como água na linha 2, é sorteado um número inteiro x, $-1 < x < 1$, e a velocidade da linha 2 será igual a $V_1 + x$, sendo V_1 a velocidade do terreno da mesma coluna na linha 1.

Explicação de algumas funções importantes:

- **int geraObstaculo(linhaT, int):** Essa função cria um obstáculo na linha passada como argumento, com o tamanho passado. Caso não seja possivel criar um obstáculo do tamanho pedido, a função retorna 0. Caso contrário, ela cria o obstáculo, coloca a velocidade das águas adjascentes ao obstáculo como 0, e retorna 1. É importante ressaltar que, caso o obstáculo fosse fazer com que o fluxo zerasse, a criação do obstáculo é cancelada é a função retorna 0.

- **linhaT geraLinha(linhaT, tamMin):** Essa função ira gerar uma nova linha baseada na linha passada como argumento. Para isso, usará dentro dela a função *igualaFluxo* para manipular as velocidades, e definirá as margens da nova linha com base nas antigas, sendo que as margens tem igual probabilidade de aumentar, diminuir ou se manter, e as margens variam sómente de 1 pixel de uma linha para outra, para evitar mudanças bruscas.

Após definidas as velocidades, os fluxos são normalizados com a função **setFluxo**, definindo o fluxo como o antigo fluxo da linha1.

2.3 List

List é um tipo de primeira classe usado para representar uma lista duplamente encadeada circular com cabeça. A ideia do funcionamento dela é que ela é constituida de nós, cada um com uma conexão para o seu próximo e seu anterior. Toda list tem um *item Atual*, sendo esse um nó da lista o qual o cliente tem acesso. E é através dele que navegaremos pela lista. A maioria das funções se baseia nele, como veremos.

A lista sempre tem um nó que define o fim/começo da lista, que chamaremos de **EOL** (*End Of List*). Nessa implementação de lista, definimos o tipo *Item* como um vetor de objetos do tipo emphlinhaT

Explicação geral das principais funções:

- **List listInit():** Ao chamar a função, é retornada uma lista do tipo List, contendo um único nó(*EOL*).

- **mvEOL(List), mvNext(List), mvPrev(List):** Essas são funções para navegação através da lista, ou seja, elas mudam o *item Atual* para o qual a lista aponta. As ações dessas funções são, respectivamente, apontar o *item Atual* para o *EOL*, para o próximo nó do *item Atual* e para o nó anterior do *item Atual*.

- **Item getItem(List):** Essa função retorna o item apontado atualmente pela lista (o *item Atual*). Caso o *item Atual* seja o EOL, a função retorna NULL.

- **removeItem(List), insertItem(List,Item):** Essas são as funções de remoção e inserção de itens na lista, respectivamente. A primeira deleta da lista o nó do *item Atual*[2], e define o novo *item Atual* como o nó anterior do nó que foi deletado. Ja a segunda função insere o Item como um nó e o posiciona como o próximo do *item Atual* da lista.

- **int isEOL(List):** Uma função que retorna 1 caso o nó para o qual a sua lista está apontando seja o *EOL*, e 0 caso contrário.

- **int emptyList(List):** Essa função retorna 1 caso a lista contenha sómente o EOL, e 0 caso contrário.

2.4 Rio

Rio é um tipo de primeira classe usado para representar o Rio em si. Nele temos as informações do fluxo, tamanho (linhas e colunas), uma List de objetos linhaT que constituem o rio, além do tamanho mínimo que o rio precisa ter. Explicação das principais funções de manipulação do Rio:

- **Rio alocaRio(int, int, float, int):** Essa função retorna um objeto do tipo Rio, sendo os parametrôs, respectivamente, para as seguintes informações:

  1. Número de linhas
  2. Número de colunas
  3. Fluxo
  4. Tamanho mínimo do rio

[2]Mas caso o cliente tente deletar o EOL, a função não faz nada.

- **int atualizaRio(Rio):** Essa função atualiza o rio. Sendo as linhas numeradas de cima para baixo de 1...N, a função deleta a N-ésima linha, e gera uma nova linha baseada na linha 1, e a coloca no começo da lista. Ou seja, a linha nova vira a linha 1, a linha 1 vira a 2, etc. Além disso, o atualiza rio retorna um int dependendo se a geração da linha foi um sucesso, ou se deu algum erro. Tabela dos valores retornados:

  + **SUCESSO_ATUALIZA**: Sem erros, rio atualizado com sucesso.
  + **FALHA_ATUALIZA**: Erro ao tentar gerar uma nova linha. Não é recomendado que se use a função desenhaRio() nesse caso. Normalmente, rios sem linhas ou erros internos do sistema causam esse erro.
  + **FALHA_OBST**: Erro ao tentar criar um obstáculo na nova linha. Esse erro acontece quando o tamanho da barreira iria zerar o fluxo do rio, ou caso o tamanho da barreira fosse exceder o tamanho do rio.

- **desenhaRio(Rio):** Essa função desenha o rio na stdout. Ou seja, imprime, em ordem, todos os terrenos das linhas da lista de objetos linhaT.

- **LinhaT getLinha(Rio,int):** Retorna a i-ésima linha do Rio, sendo que as linahs são numeradas de 1...N, de cima para baixo, sendo que N é o número total de linhas.