Projeto de MAC-211 — 2001
Simulador de Canoagem

Marco Dimas Gubitoso

26 de Março de 2001

1 Introdução

O objetivo deste projeto é o desenvolvimento de um programa composto de várias partes (módulos), ao longo do semestre.

Os módulos serão construídos ao longo de algumas fases (3) com uma fase adicional para a "amarração" e construção do programa final.

Serão consideradas a documentação e a organização do código, além do seu funcionamento. Os módulos devem ser tão independentes quanto possível e a comunicação entre dois módulos só deverá ser feita através de uma interface bem definida.

1.1 Simulação da canoagem

O projeto se resume à construção de um jogo-simulador de canoagem. O usuário do programa deverá controlar um barco solto na correnteza do rio por um número de iterações pré-determinado.

O rio será "gerado" dinamicamente de acordo com a velocidade do barco. A largura do rio e a velocidade da água variam de ponto para ponto.

O usuário poderá fazer uso de dois remos, um de cada lado, tanto para empurrar ou frear o barco e deverá evitar que o barco atinja as margens do rio e de modo a percorrer a maior distância possível.

As principais partes do programa são as seguintes:

- Simulação do rio
- Apresentação gráfica

- Tratamento da entrada do usuário
- Simulação do movimento do barco
- Jogo propriamente dito, incluindo a contagem dos pontos

Estas partes serão desenvolvidas nas etapas descritas nas próximas seções.

O programa dependerá de diversas constantes (tamanho da grade, velocidade média da água, etc). Estas constantes deverão ser definidas como tal no programa, de modo a ser fácil o ajuste para se obter um jogo mais realístico. Isto pode ser feito em C com o pré-processador. Veja a diretiva `#define`.

2 Primeira Fase: Simulação do Rio

O que o usuário verá na tela é a seção do rio onde seu barco se encontra. Esta seção de rio será gerada e atualizada durante o jogo, desta forma não precisamos ter o rio todo representado em memória, o que seria muito custoso em termos de espaço.

Para simular o rio, precisamos calcular a velocidade da água em diversos pontos do leito e a posição das margens. Faremos isso com uma grade. Cada nó da grade terá um indicador representando água ou terra e, no caso de água, a velocidade do rio naquele ponto. Para simplificar, vamos supor que a velocidade aponta sempre para o topo da tela.

Para representar a grade, o modo mais simples é utilizando uma matriz (ou uma lista ligada de linhas, para quem preferir), onde cada elemento corresponde a um nó da grade.

A margem esquerda do rio não poderá atingir a lateral esquerda da janela do jogo, nem se aproximar demais do centro do rio. Uma restrição similar vale para a margem direita. O objetivo desta limitação é garantir facilmente o não estrangulamento do rio.

A matriz (ou lista) deverá ser gerada de baixo para cima. A primeira linha a ser gerada é aquela correspondente à parte debaixo do rio (com relação à tela). As outras linhas são construídas com base nessa.

Algumas recomendações muito importantes:

- Procure separar seu código de acordo com a funcionalidade.
  Por exemplo, a geração de uma linha da matriz é de certa forma independente da geração da matriz. A rotina que gera a matriz completa

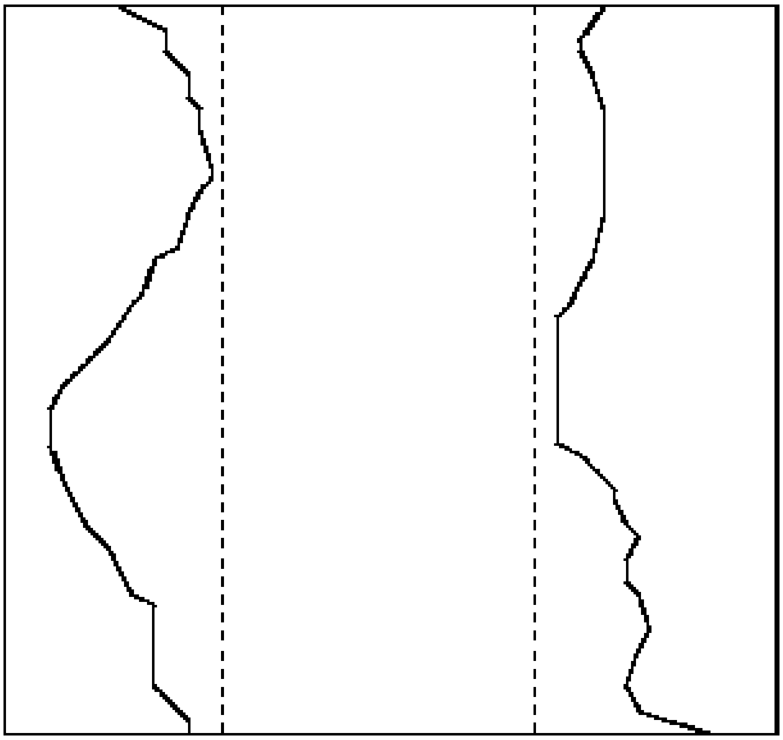

Figura 1: Limitações das margens do rio

deverá chamar a rotina que gera uma linha diversas vezes. A geradora da matriz não precisa se basear em nenhum detalhe de implementação da geradora de linha, a não ser a forma da chamada (cabeçalho).

- Quebre seu programa em vários arquivos, cada um com um conjunto de rotinas relacionado.

2.1 Geração da Primeira Linha

A primeira linha será gerada aleatoreamente com base em uma semente que poderá ser fornecida pelo usuário na linha de comando ou obtida a partir do tempo do sistema.

Deverão ser sorteadas primeiramente as margens e depois os valores da velocidade em cada ponto na água. A soma total das velocidades é o fluxo do rio, que deverá ser mantido constante em todo o rio.

A fluxo deverá ser passado como um argumento do programa. Se o usuário não fornecer este argumento, um valor *default* será utilizado.

Para ajustar as velocidades sorteadas de modo a obter o fluxo indicado, será necessário fazer uma normalização. Se a linha tiver N elementos, v_i for a velocidade na i-ésima posição da linha, $\mathcal{F}$ for o fluxo desejado e ϕ o fluxo obtido no sorteio, vale

$$\phi = \sum_{i=0}^{N} v_i \tag{1}$$

e a normalização é feita corrigindo-se o valor de cada v_i:

$$v_i \leftarrow v_i \cdot \frac{\mathcal{F}}{\phi} \tag{2}$$

É conveniente colocar o código desta normalização em uma função separada. Esta função recebe a linha (ponteiro ou número) e o fluxo desejado como entrada e altera as velocidades de acordo.

2.2 Geração das Linhas Subseqüentes

Uma vez de posse da primeira linha (correspondente à parte de baixo da tela), as outras linhas são geradas a partir dela.

Existem algumas características importantes que devem ser observadas:

- Fluxo constante de água. Normalmente se faz a suposição de que a água é um líquido incompressível. Além disso, não existem fontes nem sorvedouros no rio. [1] Isto significa que o fluxo de água deve ser zero em qualquer superfície fechada dentro do rio (divergente nulo).

  Para efeitos da nossa simulação, vamos considerar apenas o fluxo que corta uma seção transversal do rio. Em outras palavras, o fluxo em cada linha deve ser sempre o mesmo.

- Suavidade na variação das margens do rio. Não queremos variações muito bruscas na largura do rio. As linhas que definem as margens devem ser relativamente suaves. Isso pode ser obtido de diversas maneiras, limitando-se a variação da largura entre uma linha e outra.

[1] Poderia chover, mas não levaremos isso em conta

- Variação da velocidade da água. Vamos permitir que a velocidade da água no rio varie entre uma linha e outra, mas não queremos variações muito bruscas (a menos que se queira simular uma cachoeira, mas não acho que seja o caso em uma versão inicial).

2.2.1 Fluxo de água

Para garantir o fluxo constante, basta renormalizar as velocidades em cada linha, como foi feito com a primeira linha (equação 2).

2.2.2 Suavidade na variação das margens do rio

Como foi dito, isto pode ser feito de diversas maneiras. Tipicamente o que deve ser feito é limitar o quanto cada margem pode variar entre uma linha e outra, mas é importante não violar os limites máximo e mínimo das margens (figura 1).

Conhecendo-se as margens da linha anterior, as novas margens podem ser obtidas somando-se ou subtraindo-se um valor aleatório. Em classe foram discutidas algumas formas de se fazer isso respeitando os limites ao mesmo tempo. Se você tiver alguma outra idéia, esta é uma excelente chance para experimentar.

Pense na possibilidade de se colocar ilhas. Haverá um bônus de 1 ponto para quem implementar de modo satisfatório.

2.2.3 Variação da velocidade da água

Estamos admitindo que a velocidade tem apenas a componente longitudinal, isto é, aponta sempre para cima. Um ponto colado à margem tem velocidade 0.

Para se obter as velocidades a partir dos valores da linha anterior, procede-se de modo similar ao das margens. Para cada ponto da linha, verifica-se se ele mudou de estado e atualiza seu valor da seguinte forma:

1. Era rio e virou terra — sua nova velocidade é zero.

2. Era terra e virou rio — coloca-se um valor pequeno e aleatório. Alternativamente pode-se pensar em alguma forma de interpolação. Faça experiências.

3. Continua sendo rio — soma-se ou subtrai-se um pequeno valor aleatório na velocidade, tomando-se cuidado para não haver velocidade negativa.[2]

Depois de calculadas as novas velocidades, deve ser feita a normalização para garantir o fluxo constante.

Este procedimento deve ser executado até que toda a matriz seja preenchida. Ele deverá ser chamado para gerar mais linhas durante o jogo.

2.2.4 Linhas adicionais

Depois de completada a matriz, eventualmente será necessária a geração de mais linhas.

As linhas novas irão substituir as linhas presentes na matriz, em um esquema circular. Isto é, a linha mais antiga será substituída por uma nova. Isto pode ser feito de maneira bem simples e eficiente com um buffer circular.

Se você estiver usando uma lista ligada de linhas, basta fazer o final da lista apontar para a cabeça e manter um ponteiro extra para indicar a linha inicial. Inserir uma nova linha neste caso significa trocar a linha mais velha pela nova e atualizar este ponteiro.

Para a implementação via matriz, o procedimento é bem similar. Deve-se ter um inteiro que serve como índice da primeira linha. A matriz deve ser lida deste índice para baixo, até o final, e depois do começo até o valor anterior ao índice.

A linha mais velha é a anterior ao índice. Feita a troca o índice deve ser decrementado se diferente de zero. Se o índice for zero, seu novo valor será dado pelo tamanho da matriz - 1.

2.3 Testes

Procure fazer testes intensivos. Faça um pequeno programa que chama seu gerador de rio sob diversas condições e avalia os resultados.

Alguns pontos que devem ser testados:

1. Robustez. O programa sobrevive em condições especiais, como número de linhas muito grande ou muito pequeno ou fluxo muito próximo de

[2]Velocidades negativas poderiam ocorrer, mas isto traria uma complicação adicional para o simulador

zero? O que acontece se os parâmetros que definem as margens permitirem que elas se toquem?

2. Fluxo. Verificar se o fluxo é mesmo constante em cada linha.

3. Variações. O programa pode medir a variação máxima, mínima e média de cada margem e das velocidades e apresentar um relatório para análise.

   As rotinas de teste devem estar em um arquivo separado, juntamente com a função `main`.

3 Segunda Fase: Visualização Gráfica

Agora que a geração do rio está funcionando, vamos apresentá-lo em uma janela gráfica, no *Xwindow*.

Cada elemento da matriz irá corresponder ao ponto inferior esquerdo de um quadrado de largura D pixels na tela. D deve ser uma constante do programa. Inicialmente o valor de D será 10, mas pode ser alterado em função da resolução e da velocidade do programa final.

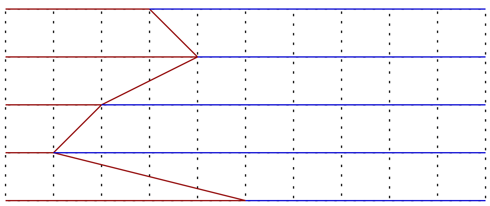

Figura 2: Interpolação

Para preencher os quadrados, você deverá se basear nas linhas superior e inferior, interpolando a figura no meio. Veja a figura 2.

O que aparecerá na tela é a imagem correspondente a um pedaço do rio.

Existe um problema sério com relação à velocidade de desenho. Não procure desenhar pixel por pixel, o melhor é desenhar retângulos ou polígonos diretamente.

Os desenhos e a animação será feita com base na biblioteca *xwc*, que nada mais é do que uma casca em cima da biblioteca básica do *Xwindow*, a

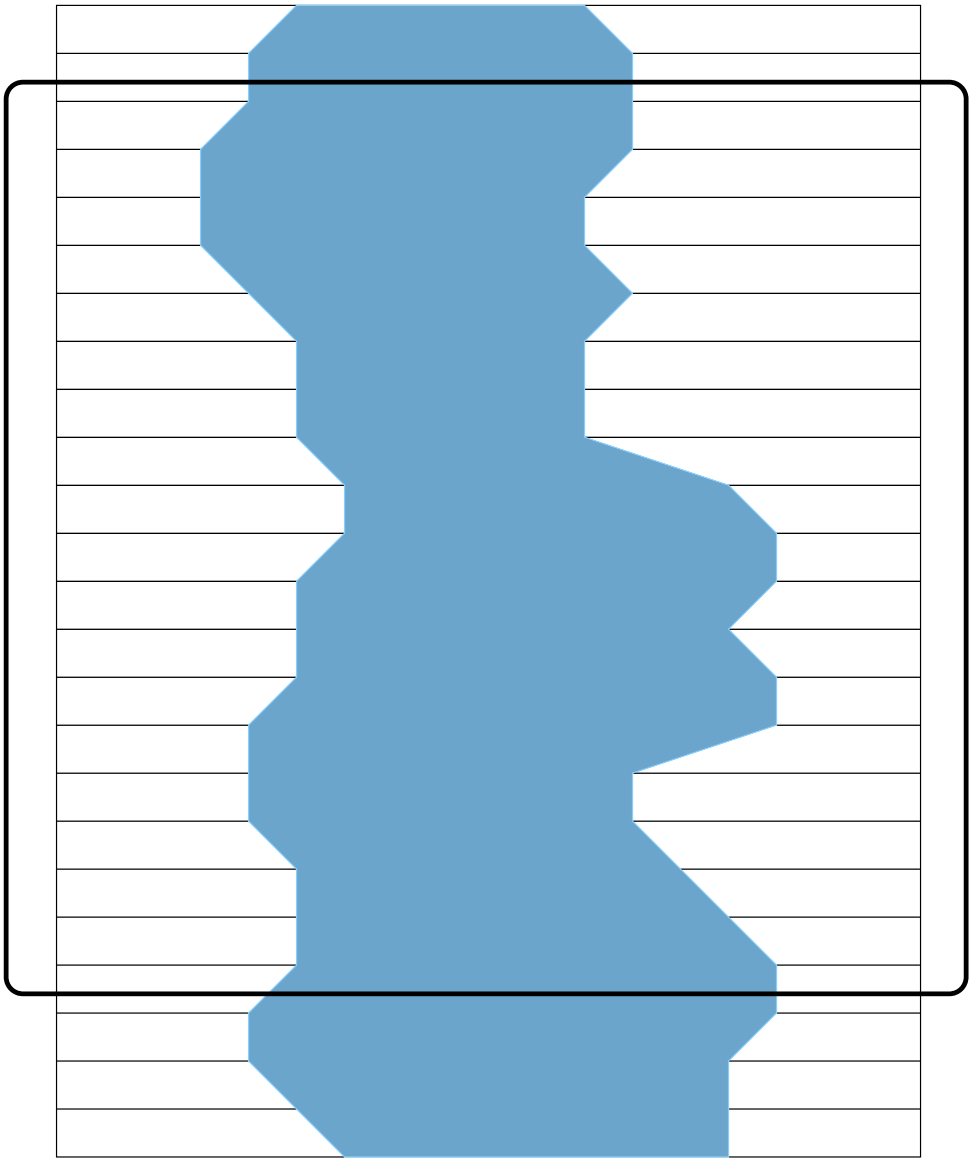

Figura 3: Desenho do rio

`libX11`. O *xwc* é simples o suficiente para permitir facilmente a construção de extensões. Veja a seção 3.1.

Durante o jogo, a tela deverá “rolar” de acordo com a velocidade do barco. Nesta fase, usaremos uma velocidade constante. Como pode ser visto na figura 3, existiram mais linhas desenhadas do que as que aparecem na tela. Isto tornará a apresentação mais suave.

Coloque as funções de desenho em um módulo à parte. Procure também fazer um módulo dedicado à animação.

3.1 xwc

Esta seção apresenta um pequeno resumo das funções do *xwc*. Para mais detalhes, veja `xwc.h`, `xwc.c` e principalmente os programas de testes. Em particular, `teste2.c` contém testes de animação.

O objeto básico de trabalho no xwc é do tipo `WINDOW`. Este tipo corresponde a janelas e áreas de trabalho.

3.1.1 Inicialização e finalização.

Para abrir uma janela e inicializar o sistema, use a função

- `InitGraph(int larg, int alt, char *nome)`
  Esta função retorna um ponteiro para um objeto WINDOW, com largura `larg`, altura `alt` e nome `nome`. A janela é criada e apresentada. **Esta função deve ser chamada antes de qualquer outra do `xwc`.** Pode ser chamada mais de uma vez.

Para encerrar e liberar os recursos alocados, use

- `CloseGraph()`
  Termina o processamento com janelas, libera memória e demais recursos.

3.1.2 Desenhando

Existem várias funções de desenho. Todas elas permitem a especificação de uma cor, indicada através de um número inteiro. A correspondência entre número e cor depende da tabela (*colormap*) ativa no momento. Para alocar entradas nesta tabela e definir suas próprias cores, use:

- *WNamedColor(char *nome)*
  Recebe o nome de uma cor e retorna o índice correspondente na tabela. Se não houver mais entradas disponíveis, retornará o índice de uma cor próxima.
  O nome pode ser retirado de `/usr/X11R6/lib/X11/rgb.txt`, ou indicado por `#rrggbb`, onde `rr`, `gg` e `bb` indicam a intensidade, em hexadecimal, correspondente às componentes vermelha, verde e azul, respectivamente (2 dígitos cada).

No que segue, em geral, `win` se refere à janela destino e `c` é a cor.
As funções de desenho propriamente ditas são:

- `WPlot(WINDOW *win, int x, int y, int c)`
  Desenha um ponto em `win`, com a cor `c`, na posição (x,y)

- `WLine(WINDOW *win, int x1, int y1, int x2, int y2, int c)`
  Desenha uma linha de (`x1`, `y1`) a (`x2`, `y2`).

- `WRect(WINDOW *win, int x, int y, int w, int h, int c)`
  Desenha um retângulo em (`x`, `y`), de largura `w` e altura `h`.

- `WFillRect(WINDOW *win, int x, int y, int w, int h, int c)`
  Idem, mas preenche o retângulo.

- `WArc(WINDOW *win, int x, int y, int a1, int a2, int w, int h, int c)`
  Desenha um arco de elipse inscrito no retângulo indicado por `x`, `y`, `w` e `h` (como em `WRect`) começando no ângulo `a1` e terminando em `a2`. Uma unidade de ângulo corresponde a 1/64 de grau.

- `WPrint(WINDOW *win, int x, int y, char *msg)`
  Desenha o texto `msg` na posição (`x`, `y`) da janela.

- `WCor(WINDOW *win, int c`
  Seleciona a cor `default`. Serve para `WPrint` por exemplo.

- `WClear(WINDOW *win)`
  Limpa a janela.

Existem outras funções de desenho no `libX`, que podem ser usadas. Dê uma olhada em `xwc.c` para ver como isso pode ser feito.

3.1.3 Área de rascunho

Áreas de desenho independentes de janelas são chamadas "`PICs`". Um `PIC` pode ter qualquer tamanho, mas deve estar associado a uma janela já existente[3].

Todas as funções de desenho podem ser indistintamente aplicadas a `PIC`s da mesma forma que são aplicadas a janelas. Um `PIC` é equivalente a `WINDOW *`.

As funções específicas de `PIC`s são:

- `NewPic(WINDOW *win, int w, int h)`
  Retorna um novo `PIC` associado a `win`, de largura `w` e altura `h`

- `FreePic(PIC pic)`
  Destroi `pic`.

- `PutPic(PIC pic1, PIC pic2, int x0, int y0, int w, int h, int x, int y`
  Desenha o retângulo indicado por `x0`, `y0`, `w` e `h` de `pic2` em `pic1`, posição (`x`, `y`).

3.1.4 Máscaras

Máscaras são figuras de duas cores que restringem a cópia de um pic em outro. Apenas os pixels correspondentes a pontos pintados na máscara são copiados. Isto permite criar áreas tranparentes em retângulos. Na verdade as janelas de formatos genéricos (*oclock* por exemplo) usam esta técnica.

As funções de desenho se aplicam a máscaras da mesma forma que se aplicam a pics e janelas. A única restrição são as cores. Use apenas 0 e 1 como índice de cor.

As funções específicas são:

- `NewMask(WINDOW *win, int w, int h)`
  Cria uma máscara de tamanho `w`×`h` adequada para objetos associados à janela `win`.

- `SetMask(PIC p, MASK mask`
  Associa a máscara `mask` ao pic `p`. Toda vez que `p`, ou parte de `p` for copiado para outro lugar, a imagem será "filtrada" por `mask`.

[3] Isto não é uma restrição real. Mas simplifica a interface, no nível em que o xwc trabalha

- `UnSetMask(PIC p)`
  Libera `p` de filtragem por máscaras, até o próximo `SetMask`.

Veja `teste2.c` para ver uma utilização interessante de máscaras.

3.1.5 XPM

O formato XPM é um formato especial para indicar figuras no Xwindow. É um formato caro em termos de tamanho, mas fácil de mexer e documentar (é uma cadeia de caracteres, legível). `teste3.c` ilustra a utilização de arquivos e figuras neste formato.

As funções são:

- `ReadPic(WINDOW *w, char *fname, MASK m)`
  Lê a imagem no arquivo de nome `fname` e desenha na janela (ou pic) `w`. Se o arquivo especificar uma cor '`None`' e se `m` for não nulo, será gerada uma máscara para a imagem e colocada em `m`.

- `WritePic(PIC p, char *fname), MASK m`
  Grava a imagem em `p`, mascarada por `m` no arquivo `fname`

- `MountPic(WINDOW *w, char **data), MASK m`
  Desenha a imagem descrita em `data` na janela `w`, com a eventual máscara em `m`.

4 Terceira fase: Dinâmica

Esta fase tratará da apresentação e movimentação do barco, bem como da interação com o usuário durante o jogo.

Novamente todos os parâmetros usados no código devem ser definidos em termos de constantes (`defines`), para permitir um ajuste fino e garantir a jogabilidade.

A parte central desta fase será calcular a nova posição e orientação do barco dada sua posição atual e a iteração com o usuário.

A iteração com o usuário será feita através de captura de eventos de teclado pela janela do jogo. Isto é feito por meio do *xwc*, como descrito no `teste2.c`, da última versão.

Defina teclas para remadas em cada lado do barco. Uma sugestão são as setas para a esquerda e para a direita. Cada vez que o usuário pressionar uma das setas, a velocidade do lado correspondente do barco em relação à água será aumentada de uma constante, que chamaremos de δ_v.

4.1 O barco

O barco terá valores associados para descrever sua situação:

- $\vec{Vi}$ — velocidade inercial do barco. Esta é a velocidade do barco no momento do cálculo. $Vi = |\vec{Vi}|$.
- θ — orientação do barco. É o ângulo que o barco faz com a vertical.
- posicionamento — distância das margens do rio, distância percorrida.

A cada iteração, estas variáveis deverão ser atualizadas e o barco reapresentado na posição nova. A velocidade inercial do barco Vi deverá sofrer uma atenuação correspondente à resistência da água. No mundo real, a resistência cresce com a velocidade relativa da água. Aqui faremos uma aproximação: Vi será multiplicada por uma constante da ordem de 80%[4].

A atualização do ângulo (θ) é mais complicada. O novo ângulo será dado pela diferença da velocidade de cada lado do barco em relação à agua. Esta velocidade pode ser aumentada por "remadas" do usuário.

Um procedimento para achar a nova orientação do barco é o seguinte, mas provavelmente não é a melhor aproximação. Você pode tentar métodos alternativos que façam sentido:

1. Sejam v_e e v_d as velocidades da água do lado esquerdo e direito do barco.
2. Sejam N_e e N_d o número de remadas em cada lado. Estes números podem ser verificados olhando-se para os eventos com o xwc.
3. Seja L a largura do barco.
4. Seja $\vec{Va}$ a velocidade que a água mais remadas adicionam ao barco. $Va = |\vec{Va}|$.
5. $v_e += N_e * \delta_v$

[4] ou um valor mais adequado, dependendo do resultado.

6. $v_d+ = N_d * \delta_v$
7. $\cot(\alpha) = \frac{v_d - v_e}{L}$
8. $Va = \frac{v_d + v_e}{2}$
9. $\vec{Va}$ tem módulo Va e orientação α.
10. A nova $\vec{Vi}$ é dada por

$$0.8\vec{Vi} + \vec{Va}$$

Com $\vec{Vi}$ atualizada, é possível calcular a nova posição do barco. O deslocamento horizontal é dado pela componente horizontal de $\vec{Vi}\Delta t$ e o deslocamento vertical — que indica o quanto deve-se "avançar" o rio, e quanto deve-se adicionar ao placar do jogador — é dado pela componente vertical de $\vec{Vi}\Delta t$, onde Δt é o intervalo de tempo correspondente a uma iteração.

O cuidado adicional que deve ser feito é verificar se o barco se chocou com as margens, ou alguma ilha.