

# Documentação EP3-A Canoagem

## Manual do Desenvolvedor

Fellipe Souto Sampaio<sup>1</sup> Gervásio Protásio dos Santos Neto <sup>2</sup> Vinícius Jorge  
Vendramini <sup>3</sup>

MAC 0211 Laboratório de Programação I  
Prof. Kelly Rosa Braghetto

Instituto de Matemática e Estatística - IME USP  
Rua do Matão 1010  
05311-970 Cidade Universitária, São Paulo - SP

---

<sup>1</sup>Número USP: 7990422 e-mail: [fellipe.sampaio@usp.com](mailto:fellipe.sampaio@usp.com)

<sup>2</sup>Número USP: 7990996 e-mail: [gervasio.neto@usp.br](mailto:gervasio.neto@usp.br)

<sup>3</sup>Número USP: 7991103 e-mail: [vinicius.vendramini@usp.br](mailto:vinicius.vendramini@usp.br)

## 1 Introdução

Esta documentação apresenta uma completa descrição do funcionamento do exercício programa 2-B e seus algoritmos, dividido através de sua implementação nos .c e .h .

## 2 Pixel

Para modelar o funcionamento da grade decidiu-se implementar um matriz de tipo pixel. Nesta estrutura existe dois campos, *velocidade* e *tipo* que guardam a velocidade e o tipo do ponto respectivamente. Para lidar com esta estrutura foi implementado as seguinte funções:

- *velocidade* - Retorna a velocidade de um ponto.
- *tipo* - Retorna o tipo de um ponto.
- *setaVelocidade* - Muda a velocidade do ponto para o valor fornecido.
- *setaTipo* - Muda a natureza do ponto para o tipo fornecido.

## 3 Rio

Esta interface lida com toda operação dos pontos, desde a geração das margens, ilhas, velocidade da água entre outras. Temos duas funç principais, *primeiraLinha* e *proximaLinha*, as quais estão disponíveis para outras interfaces. Seus métodos internos são chamados para construção das linhas, cada qual exercendo uma funcionalidade própria. O fluxo de funcionamento das duas funç está descrito nos anexos 1 e 2 respectivamente, onde cada vértice representa um método aplicado, o valor das aretas é a ordem em que são chamadas e por fim a orientação do grafo informa o fluxo de informações trocadas entre a funç chamadora e a chamada.

### 3.1 primeiraLinha

Esta função tem a funcionalidade de criar a primeira linha da grade, que servirá como linha base para linhas subsequentes. Chama-se o método *aleatorizaPrimeiraMargem* para definir tamanhos aleatórios para as margens esquerda e direita, dentro do limite estipulado pela variável *limiteMargens*. Após isso chama-se o método *insereIlha* que tem o trabalho de inserir um obstáculo na porção central do rio de acordo com a probabilidade inserida pelo usuário. Um valor default definido em *main.c* é atribuído a probabilidade e a distância mínima entre ilhas, caso o usuário não faça a inserção destes valores. Inteiros aleatórios são gerados para simular a probabilidade

P de existir ilhas ou não, seu tamanho será aleatoriamente gerado dentro dos limites da terceira porção da grade. Com intuito de evitar o encontro de uma das margens com uma dada ilha é considerado que **a distância mínima entre os dois deve ser de 2 pontos**. Aplica-se o método *velocidadeDaAguaPrimeiraLinha* a cada ponto de água que não seja vizinho a uma margem. Esta velocidade é a soma de um número aleatório no intervalo  $[-0.5, 0.5]$  somado a velocidade do ponto anterior apenas se o ponto anterior não for margem. A função também lida com casos especiais, como valores aleatórios negativos e pontos anteriores a vizinhos de margens. Matematicamente a operação realizada é a seguinte:

$$v_i = v_{i-1} + \lambda, \lambda \in [-0.5, 0.5] \quad (1)$$

Ao fim de todos estes passos, de geração à inserção, é necessário ajustar o fluxo da água na linha. Primeiro suave-se a velocidade dos pontos para que a diferença entre eles não seja tão discrepante, isto é feito em *suavizaVelocidade*, e então a velocidade dos pontos é normalizada em *normaliza* com o uso da formula de interpolação:

$$\phi = \sum_{i=0}^N v_i \quad (2)$$

$$v_i \leftarrow v_i \cdot \frac{\Phi}{\phi} \quad (3)$$

onde  $\Phi$  é o fluxo inserido pelo usuário.

### 3.2 proximaLinha

A geração das linhas subsequentes funciona de modo semelhante a função *primeiraLinha*. Cada nova linha utiliza a anterior como base para sua criação. As diferenças residem nos métodos que lidam com as margens e com a velocidade das linhas. O método *aleatorizaMargem* analisa as margens anteriores e sorteia um valor inteiro no intervalo  $[-1, 1]$ . Este valor é somado as margens da linha anterior e atribuindo-se à nova, respeitando sempre o limite das margens, isso garante que a mudança entre a margem anterior e atual seja suave. Para geração de velocidades na nova linha o método *velocidadeProximaLinha* é empregado. Com intuito de preservar a suavidade da velocidade entre pontos é tomado como base a velocidade do ponto anterior e realizado o produto desta com um valor aleatório real no intervalo  $[0.9, 1.1]$ , e no caso do ponto anterior ter velocidade zero, realiza-se um outro cálculo.

## 4 Grade

Esta interface concentra o método de criação da matriz e dos frames utilizados durante o programa. O método *initGrade* aloca dinamicamente uma matriz de tamanho  $altura \times largura$ , as quais tem valor default de 30 por 100 respetivamente. Em *criaPrimeiroFrame* cria-se a primeira imagem que será impressa, a construção das linhas acontece de baixo para cima, indo da ultima linha da matriz até a primeira. A mesma ideia é utilizada em *criaProximoFrame*, entretanto nesta é criada apenas uma nova linha e inserida na posição  $(indice + alturaDaGrade - 1) \bmod alturaDaGrade$ . Um método de desalocação de memória, *freeGrade*, foi implementado para desalocar a memória requisitada ao final do programa.

## 5 Util

Implementado nesta interface está a função *getArgs* que faz a leitura dos parâmetros passado pelo usuário e atribui as suas respectivas variáveis. Após a chamada de *getArgs* um outro método, *corrigirArgs*, é chamado para analisar as entradas fornecidas pelo usuário com intuito que corrigir possíveis parâmetros que comprometam a correta execução do programa.

## 6 main

Esta é a interface client do programa, nela são chamadas todas outras funções que realizam o trabalho bruto de criação, gerenciamento e impressão da simulação. No anexo 3 está descrito o fluxo de informações trocadas entre o cliente e suas diversas interfaces. Para encerrar a execução do programa o usuário deve pressionar o comando **ESC**, que sai do looping principal do programa. Ao fim a memória alocada é devolvida ao sistema e as interfaces do Allegro são finalizadas.

## 7 teste

Esta interface client é semelhante a main, diferindo-se na sua utilidade. Este client faz a execução dos métodos do programa com intuito de testar se as saídas do programa são concretas e confiáveis. A principal função de teste é *calculaVariacoes* que desempenha diversos cálculos. São gerados 50 frames e aplicado o método *calculaVariacoes* procurando os seguintes valores:

- Número de linhas com fluxo correto
- Velocidade máxima que um ponto de água adquiriu
- Velocidade mínima que um ponto de água adquiriu

- Velocidade média entre todos os pontos criados
- Comprimento máximo da margem esquerda
- Comprimento mínimo da margem esquerda
- Comprimento médio da margem esquerda
- Comprimento máximo da margem direita
- Comprimento mínimo da margem direita
- Comprimento médio da margem direita

Por fim esses valores são impressos na tela de forma diagramada.

## 8 rotinaTeste

Esta interface concentra os dois métodos de teste, *testaCorrecao* e *calculaVariacoes*. Na primeira testa-se se o fluxo de uma linha é igual o fluxo desejado pelo usuário e na segunda realiza-se os outros cálculos e comparações, como o comprimento maior, menor e médio das margens, velocidades máxima, mínima e média. A função também retorna quantas linhas do frame analisado passaram no teste do fluxo.

## 9 Implementação gráfica

Na parte B do exercício programa 2 a nova tarefa a ser implementada é a interface gráfica do programa utilizando a biblioteca gráfica Allegro. Tomando mão da modularização dos métodos foi possível manter a mesma implementação da etapa anterior fazendo com que a única alteração substancial dentro do ep seja na dinâmica de impressão que agora é feita de forma gráfica e não mais textual. Descreveremos completamente as mudanças e melhorias realizadas para que o programa ganhasse o aspecto desejado.

### 9.1 main

Estruturalmente main continua como na fase anterior, como seu fluxo de funcionamento. Existe agora uma nova função *STinitAllegro* que inicializa as funcionalidades utilizadas pelo Allegro para simulação gráfica. No looping principal é executado os métodos de criação da grade, verificação de eventos vindos do teclado e impressão gráfica da grade. Decidimos por implementar uma discreta navegação do barco ao longo do rio para nos nortear quanto o desempenho da velocidade do barco em relação a animação do fundo. Para que a execução seja encerrada implemetamos a tecla **ESC** que aborta o programa, em seguida desaloca-se a memória utilizada e encerra-se o programa.

## 9.2 PrintAllegro

Em PrintAllegro temos a *OutputArray* que cuida de toda lógica e impressão da parte gráfica. Caso a linha anterior seja menor que a atual um triângulo é impresso para suavizar as diferenças das margens, caso a anterior seja maior e a atual menor o mesmo também é feito, só o triângulo que muda de orientação, no caso de margens com mesmo comprimento uma linha reta é desenhada. O complemento do triângulo, que seria água, é desenhado logo em seguida, fazendo que não haja pontos sem pixels. Essa metodologia é empregada para as duas margens, no caso de ilhas e água uma simples impressão é realizada. Uma elipse foi escolhida para representar a posição do barco, sua posição pode ser manipulada com as setas do teclado simulando quase que totalmente o funcionamento do jogo. Cada caracter textual tem como valor default 5x5 pixels, mas este valor pode ser alterado pelo usuário, entretando uma alta densidade de pixels torna a execução muito mais lenta.

## 10 Execução do programa

### 10.1 Makefile

No makefile do programa existem três receitas de compilação que geram três alvos diferentes:

- make - Gera o executável ep3 que é o programa de simulação do rio propriamente dito
- make teste - Gera o executável teste que realiza as simulações de teste do rio e a geração do relatório
- make clean - Limpa os arquivos objeto e executáveis gerados pela execução das outras receitas

Optamos por conservar os arquivos ligados ao executável teste e sua receita no makefile, a confiabilidade e robustez da execução ainda podem ser verificadas com o programa.

### 10.2 Parâmetros de execução

A simulação do rio pode ser feita sem que o usuário entre com qualquer parâmetro de definição, graças a valores de default implementados. Entretando, caso seja do desejo do usuário pode-se entrar com diversos parâmetros de definição para execução da simulação ou do teste, os quais são os seguintes:

- b → Velocidade do barco
- l → Largura do Rio

- s → Semente para o gerador aleatorio
- f → Fluxo da agua
- pI → Probabilidade de haver obstaculos
- dI → Distancia minima entre obstaculos
- lM → Limite das margens
- v → Verbose
- D → Tamanho do lado de cada quadrado de pixel

### 10.3 Fotos da execução

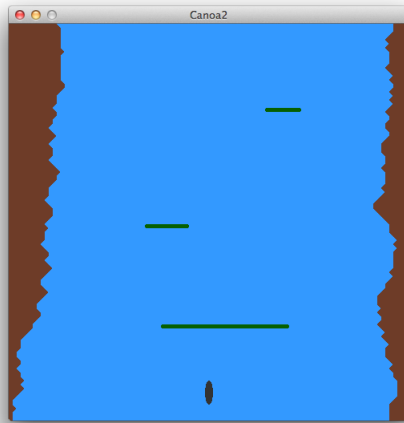


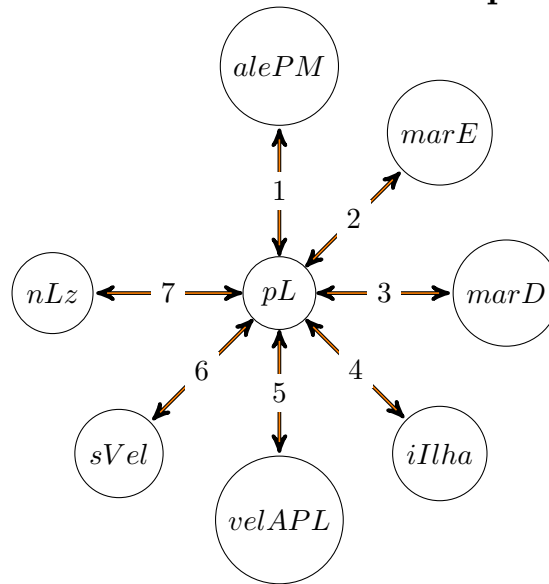
Figura 1: Execução 1



Figura 2: Execução 2



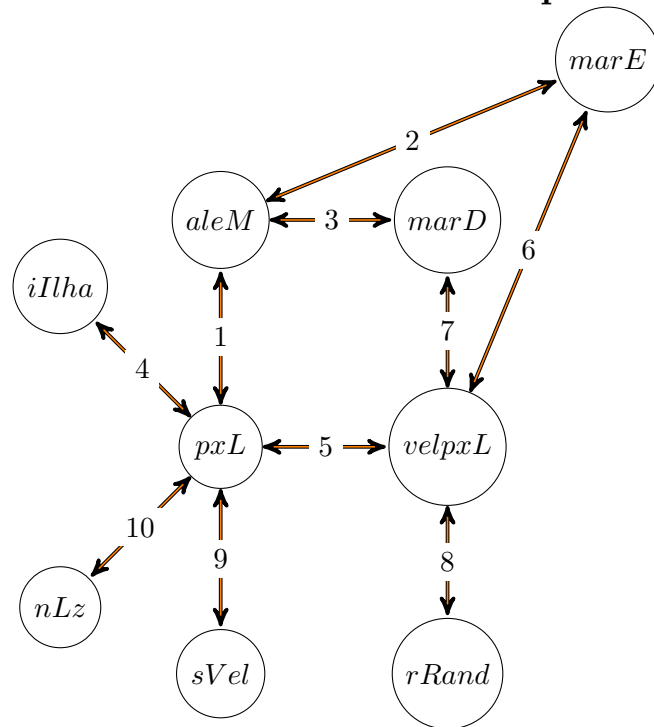
## Anexo 1 - Fluxo de funcionamento primeiraLinha



### primeimaLinha - Legenda

- *pL* - primeiraLinha
- *alePM* - aleatorizaPrimeiraMargem
- *marE* - margemEsquerda
- *marD* - margemDireita
- *iIlha* - Inserellha
- *velAPL* - velocidadeDaAguaPrimeiraLinha
- *sVel* - suavizaVelocidades
- *nLz* - normaliza

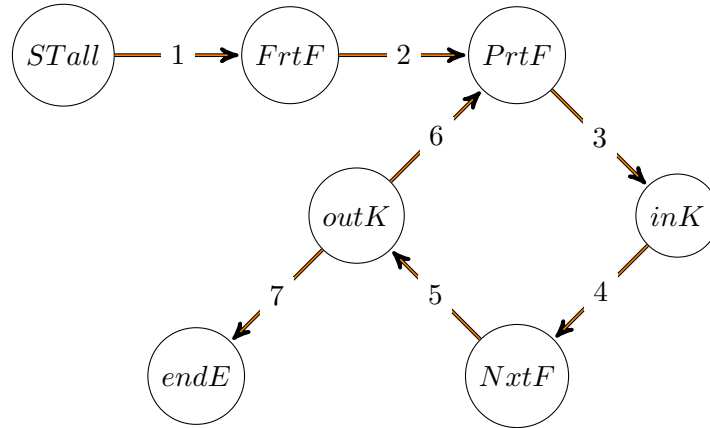
## Anexo 2 - Fluxo de funcionamento proximaLinha



### proximaLinha - Legenda

- pxL - proximaLinha
- aleM - aleatorizaMargem
- marE - margemEsquerda
- marD - margemDireita
- iIlha - InserirIlha
- velpxL - velocidadeProximaLinha
- rRand - realRandomico
- sVel - suavizaVelocidades
- nLz - normaliza

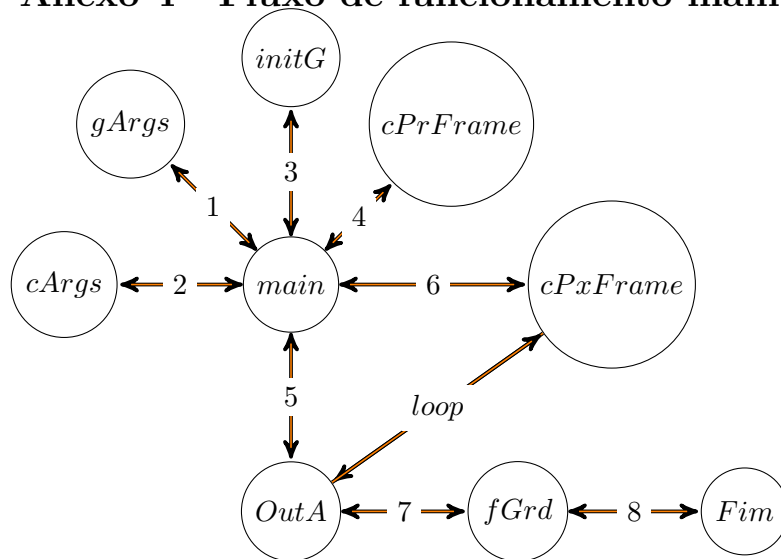
### Anexo 3 - Fluxo de funcionamento da interface gráfica



#### Fluxo de funcionamento da interface gráfica - Legenda

- *STall* - Inicialização do allegro
- *FrtF* - Criação do primeiro frame
- *PrtF* - Impressão do frame
- *inK* - Leitura de um evento do teclado
- *NxtF* - Criação dos novos frames
- *outK* - Execução de um evento do teclado
- *endE* - Fim da execução

## Anexo 4 - Fluxo de funcionamento main



### main - Legenda

- `gArgs` - `getArgs`
- `cArgs` - `corrigeArgs`
- `initG` - `initGrade`
- `cPrFrame` - `criaPrimeiroFrame`
- `OutA` - `OutputArray`
- `cPxFrame` - `criaProximoFrame`
- `fGrd` - `freeGrade`