Documentação EP4 - Canoagem

Manual do Desenvolvedor

Fellipe Souto Sampaio[1] Gervásio Protásio dos Santos Neto [2] Vinícius Jorge Vendramini [3]

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

[1]Número USP: 7990422 e-mail: fellipe.sampaio@usp.com
[2]Número USP: 7990996 e-mail: gervasio.neto@usp.br
[3]Número USP: 7991103 e-mail: vinicius.vendramini@usp.br

1 Introdução

Esta documentação apresenta uma completa descrição do funcionamento do exercício programa 2 parte C (vulgo ep 4) e seus algoritmos, divido através de sua implementação nos .c e .h . As implementações desenvolvidas nas fases anteriores foram mantidas, nesta terceira etapa as mudanças se concentraram nos seguintes arquivos:

- main.c
- util.c
- util.h
- PrintAllegro.c
- Output.h
- Vector_2D.c
- Vector_2D.h
- options.h
- config.l
- config.y
- makefile

As novas explanações serão feitas para as mudanças realizadas nos itens citados acima, mantendo-se as explicações anteriores para os arquivos que foram omitidos.

2 Pixel

Para modelar o funcionamento da grade decidiu-se implementar um matriz de tipo pixel. Nesta estrura existe dois campos, *velocidade* e *tipo* que guardam a velocidade e o tipo do ponto respectivamente. Para lidar com esta estrutura foi implemetado 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 atribuido 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 invervalo $[-0.5, 0.5]$ 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. Matemáticamente a operação relizada é a seguinte:

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

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:

$$\phi = \sum_{i=0}^{N} v_i \tag{2}$$

$$v_i \leftarrow v_i \cdot \frac{\Phi}{\phi} \tag{3}$$

onde Φ 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 interválo $[0.9, 1.1]$, 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 $(indice + alturaDaGrade - 1) \mod alturaDaGrade$. 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, *corrigeArgs*, 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 encerar 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 análisado passaram no teste do fluxo.

9 Implementação grafica

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 unica 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 funçcão /testeitSTinitAllegro 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 para o programa, desaloca a memória utilizada e encerra 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 triangulo 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. Essa metodologia é empregada para as duas margens; no caso de ilhas, 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 Implementações finais e dinâmica de jogo

10.1 main

Em nosso arquivo principal main , como nas etapas anteriores, acontece toda a dinâmica e processamento de jogo. Analisaremos linearmente as mudanças feitas para melhor interdimento. As bibliotecas Allegro aumentaram, sendo

agora um total de 8. Cada uma é responsável por algo implementado no código, temos as principais:

Allegro_image - Responsável pelo sprite do barco.

Allegro_audio - Responsável pela música de fundo e o sample.

Allegro_acodec - Responsável pelo codec de som.

Allegro_font - Responsável pelo gerenciamento da escrita na tela.

Allegro_ttf - Responsável pela fonte utilizada na escrita.

Além das novas diretivas Allegro nosso programa possui um analisador léxico e sintatico, implementados com Flex e Bison, uma maior descrição será feita na seção do analisador. Continuando no arquivo main temos o looping principal que cuida da análise de eventos vindos do teclado e de seu processamento, cada comando tem um comportamento bem definido influenciando de forma singular a movimentação do barco. O funcionamento geral do looping será explicado nas interfaces chamadas dentro dele e no anexo 5 que representa um diagrama do fluxo de processamento do looping. Por fim o main desaloca toda memória utilizada pelos processos do allegro e a memória alocada para manutenção da grade.

10.2 Analisador léxico e sintatico

Nosso analisador léxico e sintatico foi escrito utilizando as ferramentas Flex e Bison respectivamente. Em *config.l* são definidos as expressões regulares que o analisador léxico deve interpretar e de que forma essas informações devem ser retornadas para o Bison. Em *config.y* a semântica dos itens léxicos é delimitada, atribuindo a cada variável externa vinda do header **options.h**, um valor lido do arquivo de texto processado. Essas variáveis externas estão definidas na header mencionada e são visíveis a main.c como parâmetros de configuração do programa. Na fase anterior a entrada era processada pelo via stream de dados, sendo substituido agora pelo nosso analisador, entretando o usuário ainda pode passar parâmetros via linha de comando, e estes possuem maior prioridade sobre o analisador. Além disso parâmetros que possam comprometer o funcionamento do programa são analisados em *util.c* e corrigidos para valores default mínimos.

10.3 PrintAllegro

A impressão da interface gráfica do programa é o principal elo de interatividade entre o jogador e o programa, e pensando nisso optamos por implementar recursos visuais de fácil intendimento de todos usuários. Em

comparação a a fase anterior podemos citar com grande diferença na visualização do jogo o sprite (objeto bidimensional que se move em um jogo) do barco e sua movimentação.

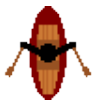


Figura 1: Barco

10.4 Movimentação do barco e controles

A movimentação do barco acontece utilizando as quatro setas do teclado. Os eventos são lidos no main.c e processados lá mesmo. A cada toque para esquerda ou direita o barco efetua uma pequena rotação. Inicialmente o barco possui uma posição neutra próximo ao meio inferior da tela. A cada remada para frente uma pequena velocidade é somada e a posição do barco no próximo frame será a frente da anterior. Caso a tecla baixo seja pressionada ou o barco fique estático sem receber nenhum comando o barco é lentamente movido para trás utilizando a velocidade que a água adquire na posição (x,y) na grade. Isso simula a força de arrasto que um barco pode ter ao navegar contra uma correnteza contrária. Caso o usuário queira sair da tela por qualquer um dos quatro lados a posição do barco é corrigida para que este sempre esteja a vista do jogador. Nas rotações o barco usa a variável angle, rotando uma quatidade |X| de radianos de acordo com o lado requisitado. Os cálculos trigonométricos necessários são feitos em **Vector_2D** e foram implementados de forma a serem chamados o mínimo de vezes possível, evitando assim muitas chamadas de funções matemáticas custosas.

10.5 Dinâmica de jogo

Para se jogar o simulador de canoagem deve-se desviar das ilhas o máximo de tempo possível. Existe um contador implementado no canto da tela que conta quantos minutos e segundos o usuário passou sem ser atingido por uma margem ou ilha. Cada vez que uma colisão acontece o contador de tempo é resetado e o barco adquire uma pequena invencibilidade de cerca de 3 segundos para que o usuário corrija sua posição na água. Durante todo o jogo uma pequena música de fundo é tocada e a cada remada um som é executado. Caso o usuário queira finalizar o jogo deve-se apertar a tecla **ESC**, que o leva para uma tela que exibe o maior tempo que o jogador conseguiu jogar sem colisão e um placar de quantos pontos ele conseguiu marcar.

11 Execução do programa

11.1 Makefile

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

- make - Gera o executável ep4 que é o jogo Canoagem
- 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.

11.2 Parâmetros de execução

A simulação é feita a partir de variáveis setadas pelo parser do analisador. Entretando, caso seja do desejo do usuário pode-se entrar com diversos parâme-tros de definição para execução da simulação ou do teste, os quais são os seguintes:

-b → FPS inicial

-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

No arquivo de entrada para o analisador os campos devem ser escritos da seguinte forma:

FPS = %d

Semente = %d

Largura = %d

Fluxo = %d

Pixel = %d

Verbose = %d

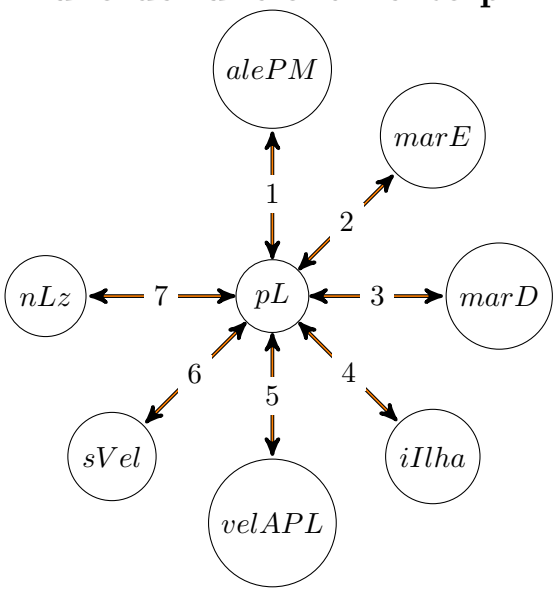
Probabilidade = %f

Distancia = %d

Limite = %f

onde %d é um número inteiro e %f um ponto flutuante. As posições podem ser permutadas ou omitidas, neste ultimo caso receberão valores default, porem a forma de escrita apresentada acima (letras maiúsculas e minúsculas) deve ser respeitada, afim de que o parâmetro delimitado seja lido corretamente. Por fim o arquivo de configuração deve ter o nome **entrada.txt** e estar localizado no mesmo diretório do executável.

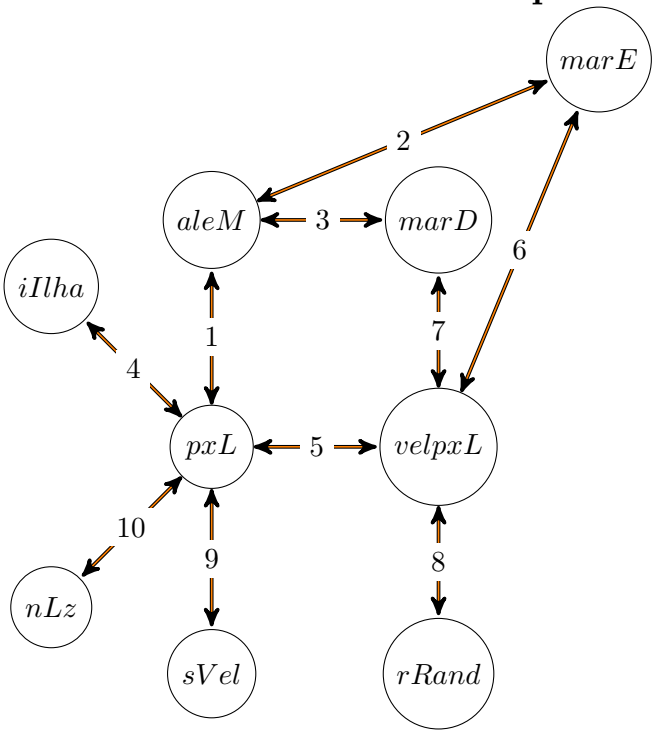
Anexo 1 - Fluxo de funcionamento primeiraLinha



primeimaLinha - Legenda

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

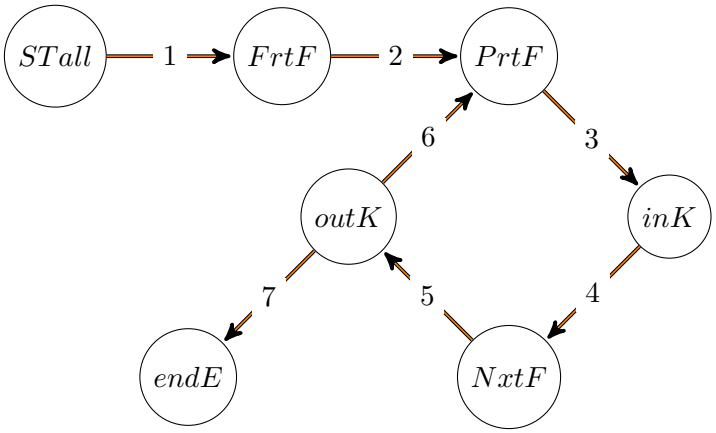
Anexo 2 - Fluxo de funcionamento proximaLinha



proximaLinha - Legenda

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

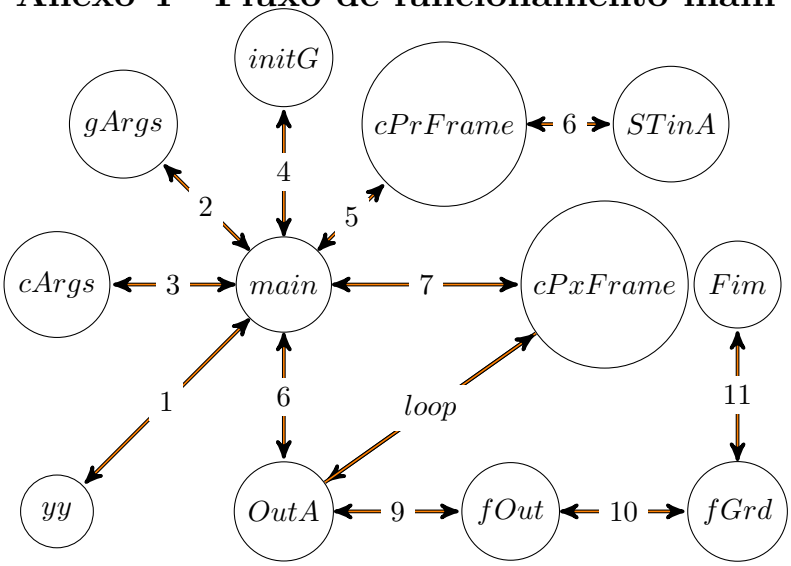
Anexo 3 - Fluxo de funcionamento da interface gráfica



Fluxo de funcionamento da interface gráfica - Legenda

- STall - Inicialização do allegro
- FrtF - Criação do primero 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

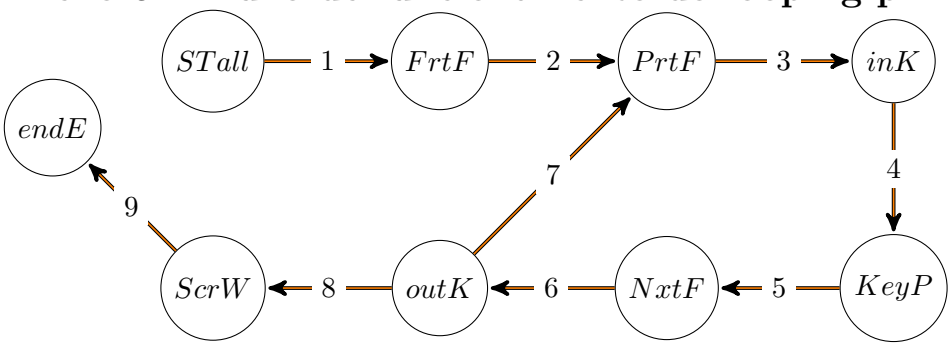
Anexo 4 - Fluxo de funcionamento main



main - Legenda

- yy - yyparse
- gArgs - getArgs
- cArgs - corrigeArgs
- initG - initGrade
- cPrFrame - criaPrimeiroFrame
- STinA - STinitAllegro
- OutA - OutputArray
- cPxFrame - criaProximoFrame
- fOut - freeOutput
- fGrd - freeGrade
- Fim - Fim da execução do programa

Anexo 5 - Fluxo de funcionamento do looping principal



Fluxo de funcionamento do looping principal - Legenda

- STall - Inicialização do allegro
- FrtF - Criação do primero frame
- PrtF - Impressão do frame
- inK - Leitura de um evento do teclado
- KeyP - Processamento do comando (métodos Vector_2D, correção da posição do barco fora da tela)
- NxtF - Criação dos novos frames
- outK - Execução de um evento do teclado
- ScrW - Mostra a tela de Score do jogador
- endE - Fim da execução

Obs: O processamento entre os passos 3-7 é um looping até o user teclar **ESC**