Departamento de Ciência da Computação – IME-USP

MAC211 – Laboratório de Programação I
Quarto Exercício-Programa (EP4)

Canoagem – Fase 3 *†

Junho de 2013

Data de entrega: 28/06/2013

1 Introdução

O objetivo deste projeto é o desenvolvimento de um programa composto de várias partes (módulos). Os módulos serão construídos ao longo de três fases, distribuídas até o final do semestre. O projeto deve ser desenvolvido em linguagem C, por grupos de 2 a 3 alunos, para um ambiente PC/Linux.

Na avaliação do EP, 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 por meio de uma interface bem definida.

1.1 Simulação da canoagem

O projeto produzirá 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 quanto para frear o barco. O objetivo do jogo é percorrer a maior distância possível, ao mesmo tempo em que deve-se evitar que o barco atinja as margens do rio ou os obstáculos existentes no caminho.

O programa dependerá de diversas constantes (tamanho da grade, velocidade média da água, probabilidade de existência de obstáculos no rio, etc.). Essas constantes deverão ser definidas como tal no programa, de modo a ser fácil o ajuste para se obter um jogo mais realístico. Constantes podem ser definidas em C por meio da diretiva \`#define`.

As principais partes do programa são as seguintes:

*A primeira versão deste EP foi criada pelo Prof. Marco Gubitoso (o Gubi) e já foi adaptada por vários outros professores ao longo dos anos. Este texto é uma coletânea dessas adaptações. Fica registrado aqui o meu agradecimento à genialidade dos professores Gubi, Kon, Reverbel e Hirata.

†Este exercício-programa é uma continuação do anterior. As Seções de 1 a 3 deste texto são exatamente as mesmas do enunciado do EP3. A tarefa para o EP4 está definida na Seção 4. Não deixe de ler novamente a Seção 5, que traz observações importantes sobre o que se espera de sua implementação.

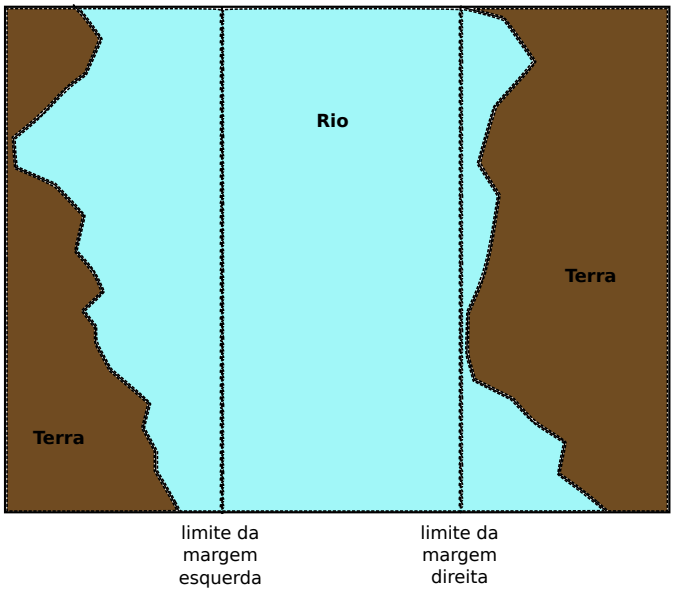

Figura 1: Limitações das margens do rio.

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

2 Primeira Fase: Simulação do Rio

O que o usuário verá na tela é a seção do rio onde seu barco se encontra. Essa 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 usar uma matriz (ou uma lista ligada de linhas, para quem preferir), em que 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 Figurã1 ilustra essa ideia.

A matriz (ou lista) deverá ser gerada de baixo para cima. A primeira linha a ser gerada é aquela correspondente à parte de baixo 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 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 relacionadas.

2.1 Geração da Primeira Linha

A primeira linha será gerada aleatoriamente 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 de uma linha é o fluxo do rio, que deverá ser mantido constante em todo o rio. O fluxo deverá ser passado como um argumento do programa. Se o usuário não fornecer esse 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 que

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

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

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

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

2.2 Geração das Linhas Subsequentes

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.
- **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).

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

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 (Figurã1).

Conhecendo-se as margens da linha anterior, as novas margens podem ser obtidas adicionando-se ou subtraindo-se um valor aleatório. Entretanto, você deve ficar atento para o fato de que os limites das margens sempre devem ser respeitados.

2.2.3 Geração de obstáculos

O rio deverá conter ilhas, que funcionarão como obstáculos para a navegação no rio.

As ilhas devem ser geradas aleatoriamente e podem ter diferentes larguras. Cada linha pode ter no máximo uma ilha. Além disso, as ilhas não podem aparecer em linhas muito próximas e nem podem ser muito largas, pois isso comprometeria a "jogabilidade" do seu programa.

Seu programa deverá ter parâmetros que determinam coisas como a probabilidade da existência de uma ilha, a distância mínima entre ilhas, etc.

2.2.4 Variação da velocidade da água

Estamos admitindo que a velocidade tem apenas a componente longitudinal, isto é, aponta sempre para cima.

As velocidades dos pontos de uma linha são calculadas em função dos pontos vizinhos e dos valores da linha anterior. Para cada ponto da linha, a velocidade é calculada da seguinte forma:

1. Se o ponto é terra, então sua velocidade é zero.

2. Se o ponto é rio e está colado à esquerda ou à direita de um ponto de terra, então sua velocidade é zero.

3. Se o ponto era terra e virou rio, então atribui-se um valor pequeno e aleatório para a sua velocidade. Alternativamente, pode-se pensar em alguma forma de interpolação da velocidade. Faça experiências.

4. Se o ponto era rio e continuou rio, então soma-se ou subtrai-se um pequeno valor aleatório na velocidade, tomando-se cuidado para não haver velocidade negativã[2]. Alternativamente, pode-se pensar em alguma forma de interpolação da velocidade. Faça experiências.

É importante ressaltar que um ponto é terra se ele está em uma das margens do rio ou se ele constitui um ilha.

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

Esse procedimento de cálculo de velocidades deve ser executado a cada vez que uma nova linha é gerada na grade.

2.2.5 Linhas adicionais

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

[2]Velocidade negativas poderiam ocorrer, mas isso traria uma complicação adicional para o simulador

As linhas novas irão substituir as linhas presentes na grade 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 número de linhas máximo da matriz menos um.

2.3 Visualização em modo texto

Nesta primeira fase, o rio será representado ao usuário através de uma saída baseada apenas em texto, com uma "resolução" de 100×30 "pixels" (na verdade, 100×30 caracteres). Ou seja, você irá imprimir a saída usando funções como, por exemplo, `printf`.

O seu programa deverá gerar várias "fotos" instantâneas do rio em intervalos de tempo fixos (por exemplo, uma foto a cada 100 milisegundos). Com isso, o resultado será uma animação que equivale à visão de uma pessoa que está sobrevoando o rio no sentido de sua correnteza.

Eis aqui um exemplo de foto do rio:

```
#################...........................................########################################
################.............................................#######################################
#################.............................................######################################
################.............................................#######################################
###############.............................................########################################
###############.............................................########################################
##############..............................................########################################
###############..............................................#######################################
################..............................................######################################
#################..............................................#####################################
################..............................................######################################
###############...............................................######################################
###############...................######......................######################################
##############.................................................#####################################
#############...................................................####################################
##############...................................................###################################
###############...................................................##################################
################.................................................###################################
#################...............................................####################################
#################................................###################################
#################.................................................##################################
##################.................................................#################################
###################..................................................################################
####################.......################..........................###############################
#####################.................................................##############################
#####################................................................###############################
#####################.................................................##############################
#####################..................................................#############################
####################..................................................##############################
#####################................................................###############################
```

No exemplo, usou-se o caractere '`.`' para representar a água do rio e o caractere '`#`' para representar terra. Note que as ilhas também são denotadas como pontos de terra no meio do rio.

O executável para Linux `canoagem_fase1`, disponibilizado no Paca junto com este enunciado, exemplifica o tipo de saída que é esperado nesta fase do projeto. Para encerrar a animação mostrada pelo programa, digite [CTRL-C].

Observe que nesta primeira fase do projeto, as velocidades calculadas nos pontos do rio não são consideradas para a geração da visualização. Entretanto, você já deve implementar o cálculo das velocidades nesta fase.

2.4 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:

- **Robustez**. O programa sobrevive em condições especiais, como número de linhas muito grande ou muito pequeno, ou fluxo muito próximo de zero? O que acontece se os parâmetros que definem as margens permitirem que elas se toquem?
- **Correção**. O programa garante um fluxo constante em cada linha da grade?
- **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 ser implementadas em um módulo separado, mas podem ser acionadas a partir do mesmo programa que o jogo (por meio de parâmetros passados via linha de comando).

3 Segunda Fase: Visualização Gráfica

Agora que a geração do rio está funcionando, vamos apresentá-lo em uma janela gráfica. Você deverá adaptar o código que escreveu na fase anterior para que o mesmo utilize elementos gráficos (ao invés de usar caracteres em modo texto) na representação do rio, das margens, das ilhas, da canoa e do que mais você quiser incluir no seu jogo. Nesta fase, você não precisa se preocupar com o controle da canoa e nem com a interação com o usuário: esses aspectos serão desenvolvidos na próxima fase.

Nesta fase, cada elemento da grade implementada na fase anterior 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 5, mas pode ser alterado em função da resolução e da velocidade do programa final.

Para preencher os quadrados, você deverá se basear nos tipos dos nós das linhas superior e inferior da grade, interpolando a figura no meio (veja a Figurã2).

Assim como o que foi feito na visualização em modo texto, o que aparecerá na tela á a imagem correspondente a um pedaço do rio.

Num primeiro momento, você pode desenhar sua canoa como um retângulo ou uma elipse que aparecerá sempre no meio do rio. Não é preciso se preocupar com colisões com ilhas, já que não é possível controlar a direção da canoa agora.

O desenho de uma visualização gráfica é uma tarefa computacionalmente pesada. Por essa razão, para não comprometer o desempenho do seu jogo, evite desenhar pixel por pixel da visualização do rio. Uma melhor estratégia é desenhar retângulos, triângulos ou outros polígonos diretamente.

Os desenhos e a animação deverão ser feitos com o auxílio da biblioteca *Allegro*, uma biblioteca para a programação de jogos e aplicações multimídias. As versões 4 e 5 dessa biblioteca são multiplataformas (funcionam no Unix/Linux, Windows, MacOS, iPhone e Android). A biblioteca se ocupa de tarefas de "baixo nível", como a criação de janelas, a aceitação de dados de entrada de usuários, o carregamento de dados, o desenho de imagens, de sons, etc., abstraindo para o programador toda a complexidade subjacente a esse tipo de manipulação computacional.

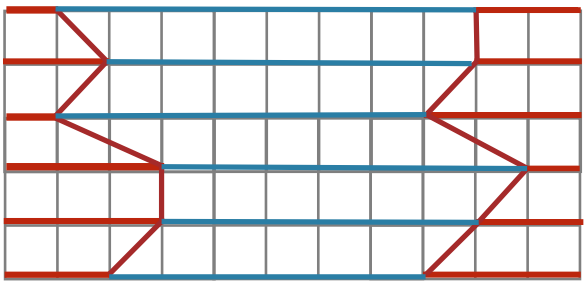

Figura 2: Interpolação dos nós da grade.

Coloque as funções de desenho e animação do rio em um (ou mais) novo(s) módulo(s), como uma extensão do seu código desenvolvido para a fase anterior.

3.1 Sobre a biblioteca *Allegro*

Para implementar a Fase 2 deste EP, usaremos a versão 5.0.9 da *Allegro*, porque essa é a versão estável mais recente da biblioteca. O site da biblioteca é o `http://alleg.sourceforge.net/`. O *download* da biblioteca pode ser feito a partir da página `http://alleg.sourceforge.net/download.html`. As instruções para a instalação em diferentes sistemas operacionais estão em: `http://wiki.allegro.cc/index.php?title=Allegro_5_Tutorial#Installing_Tutorials`. No Apêndice A deste enunciado, você encontra as instruções para instalação da biblioteca no sistema operacional Ubuntu.

Boas referências para o aprendizado do funcionamento básico da *Allegro* 5 são:

- Um tutorial em português bem fácil de acompanhar e que ensina boa parto do que é necessário para implementar as fases 2 e 3 deste EP:
  `http://www.rafaeltoledo.net/tutoriais-allegro-5/`.
  Para a Fase 2, as funcionalidades básicas que você provavelmente usará estão descritas nas seguintes partes do tutorial:
  - Compilando e Instalando a *Allegro* 5 no Ubuntu
    [Atenção: se você já instalou a Allegro usando as instruções da Wiki ou do Apêndice A, pule diretamente para o final dessa parte do tutorial, onde aparece um código fonte para você testar se a instalação feita está funcionando corretamente.]
  - Criando uma Janela
  - Colocando uma Imagem na Tela
  - Eventos
  - Primitivas Gráficas
  - Animações

  Para a Fase 3, as seguintes partes do tutorial poderão ser úteis:
  - Mouse
  - Usando Fontes *True Type* (TTF)
  - Teclado
  - Áudio
  - Ajuste de FPS (Frames por Segundo)
- Manual de referência da biblioteca (versão 5.0.9): `http://alleg.sourceforge.net/a5docs/5.0.9/`

- Wiki da biblioteca (com artigos, tutoriais, etc.): `http://wiki.allegro.cc/`
- Fórum dos usuários da *Allegro*: `https://www.allegro.cc/`

Um exemplo de visualização gráfica feita com a *Allegro* (executável `canoagem_fase2`) para a simulação do rio encontra-se no Paca, juntamente com este enunciado.

4 Terceira Fase: Dinâmica do Jogo

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`) e/ou valores definidos pelo usuário, 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. Isso deverá ser feito por meio da biblioteca *Allegro* [3]. 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:

- $\vec{V_i}$ – velocidade inercial do barco. Essa é a velocidade do barco no momento do cálculo. $V_i = |\vec{V_i}|$.
- 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, essas variáveis deverão ser atualizadas e o barco reapresentado na posição nova. A velocidade inercial do barco $\vec{V_i}$ 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: $\vec{V_i}$ será multiplicada pela constante 0,8 [4].

A atualização do ângulo (θ) é mais complicada. O novo ângulo será influenciado pela diferença da velocidade de cada lado do barco em relação à agua. Essa velocidade pode ser aumentada por "remadas" do usuário. Um procedimento para achar a nova velocidade e orientação do barco é o mostrado a seguir, mas provavelmente não é a melhor aproximação (você pode tentar métodos alternativos que façam sentido):

- Sejam v_e e v_d as velocidades da água do lado esquerdo e direito do barco.
- Sejam N_e e N_d o número de remadas em cada lado. Esses números podem ser obtidos por meio da captura de eventos com a *Allegro*.
- Seja L a largura do barco.

[3] Para um exemplo sobre como capturar eventos de teclado com a *Allegro*, consulte o tutorial `http://www.rafaeltoledo.net/tutorial-allegro-5-6-utilizando-o-teclado/`.

[4] Ou por um outro valor mais adequado, dependendo do resultado.

- Seja $\vec{V_a}$ a velocidade que a água mais remadas adicionam ao barco. $V_a = |\vec{V_a}|$.
- $v_e = v_e + Ne \times \delta_v$
- $v_d = v_d + Nd \times \delta_v$
- $cot(\alpha) = \frac{v_d - v_e}{L}$
- $V_a = \frac{v_d + v_e}{2}$
- $\vec{V_a}$ tem módulo V_a e orientação α.
- A nova $\vec{V_i}$ é calculada como: $\vec{V_i} = 0.8\vec{V_i} + \vec{V_a}$. O novo ângulo θ corresponde à orientação da nova $\vec{V_i}$; essa orientação é influenciada tanto pelo valor anterior de θ quanto pelo ângulo α.

Com $\vec{V_i}$ atualizada, é possível calcular a nova posição do barco. O deslocamento horizontal é dado pela componente horizontal de $\vec{V_i}\Delta t$, 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 $\vec{V_i}\Delta t$, onde Δt é o intervalo de tempo correspondente a uma iteração.

Um cuidado adicional que deve ser tomado é verificar se o barco não se chocou com as margens ou com alguma ilha. Você deve aplicar penalidades ao usário em caso de choques (como, por exemplo, o desconto de pontos do placar ou a perda de vidas).

4.2 O arquivo de configurações

A fim de exercitar conceitos apreendidos em aula, nesta fase do EP você deve permitir que os usuários passem seus parâmetros de configuração do jogo por meio de um arquivo texto fornecido como entrada para o programa.

Você deve criar uma "mini-linguagem", que descreva a sintaxe que um usuário deve usar para definir os parâmetros que desejar e seus respectivos valores. O *parser* que extrairá desse arquivo de configuração os valores necessários para o funcionamento do programa deverá ser gerado com o auxílio das ferramentas *Flex* e *Bison*.

Não se esqueça de:

- explicar no manual do usuário o formato esperado para o arquivo de configurações;
- entregar, junto com o código-fonte do seu programa, os arquivos `.l` e `.y` (entrada para o *Flex* e o *Bison*, respectivamente) criados.

5 Considerações Importantes

- Critérios de avaliação do EP:

  \+ Documentação de desenvolvedor
  \+ Documentação de usuário
  \+ Organização e clareza do código
  \+ Uso correto de estruturas de dados
  \+ Uso correto de técnicas de programação

    + A independência dos módulos/bibliotecas
    + O interfaceamento para comunicação entre os módulos
    + Funcionamento e robustez do código
    + Apoio à compilação e à geração do executável por meio do utilitário `make` e um arquivo `Makefile`
    + Uso do Latex para a criação da documentação
    + **Uso do Flex e do Bison para a geração de um *parser* para o arquivo de configurações do jogo (Novo!)**

- Dúvidas sobre o EP podem ser discutidas no fórum da disciplina no Paca, para o aproveitamento de todos.
- O Departamento de Ciência da Computação considera o plágio (ou cola) uma infração disciplinar inadmissível e, na ocorrência de tais casos, recomenda a reprovação do aluno na disciplina e o relato da ocorrência na Comissão de Graduação para as devidas providências. Neste oferecimento de MAC211, poderemos usar métodos computacionais para a detecção de plágio.

Apêndice A
Instruções para a instalação da *Allegro* no Ubuntu

Os passos para a instalação da *Allegro* versão 5.0.9 no Ubuntu são:

1. Abra um terminal e execute os seguintes comandos, para a instalação dos pacotes dos quais a biblioteca depende:

   ```
   $ sudo apt-get update

   $ sudo apt-get install libgl1-mesa-dev libglu1-mesa-dev cmake build-essential
     make libxcursor-dev

   $ sudo apt-get install -y cmake g++ freeglut3-dev libxcursor-dev libpng12-dev
     libjpeg-dev libfreetype6-dev libgtk2.0-dev libasound2-dev libpulse-dev
     libopenal-dev libflac-dev libdumb1-dev libvorbis-dev libphysfs-dev
   ```

2. Baixe o código fonte da biblioteca disponível em:
   `http://alleg.sourceforge.net/download.html`
3. Descompacte o código fonte (aqui neste exemplo, o arquivo `allegro-5.0.9.tar.gz`). Um diretório chamado `allegro-5.0.9` (ou só `allegro`) será criado.
   ```
   $ tar -zxvf allegro-5.0.9.tar.gz
   ```
4. Crie um diretório `build` no diretório `allegro-5.0.9` e entre nele:

   ```
   $ mkdir build
   $ cd build
   ```

5. Execute o comando `cmake` no diretório `build` da forma mostrada abaixo:
   ```
   $ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
   ```
6. Execute o comando `make` no diretório `build`:
   ```
   $ make
   ```
7. Para terminar, execute o `make install` no diretório `build`:
   ```
   $ sudo make install
   ```