PLAYAPC

PLAYAPC

Biblioteca gráfica para programadores inexperientes

Sinayra Pascoal Cotts Moreira
Universidade de Brasília

Prof. Dr. José Carlos Loureiro Ralha
Universidade de Brasília

Prof. Dr. Alexandre Zaghetto,
Universidade de Brasília

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Survey Methodology / Robert M. Groves . . . [et al.].
p. cm.—(Wiley series in survey methodology)
"Wiley-Interscience."
Includes bibliographical references and index.
ISBN 0-471-48348-6 (pbk.)
1. Surveys—Methodology. 2. Social
sciences—Research—Statistical methods. I. Groves, Robert M. II. Series.

HA31.2.S873 2007
001.4'33—dc22 2004044064
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

À todos os alunos que
queiram fazer trabalhos
bonitinhos na primeira
matéria de computação
da UnB

CONTRIBUTORS

CONTENTS IN BRIEF

CONTEÚDO

LIST OF FIGURES

LIST OF TABLES

PREFACE

This is an example preface. This is an example preface. This is an example preface. This is an example preface.

R. K. Watts

Durham, North Carolina
September, 2007

ACKNOWLEDGMENTS

From Dr. Jay Young, consultant from Silver Spring, Maryland, I received the initial push to even consider writing this book. Jay was a constant "peer reader" and very welcome advisor durying this year-long process.

To all these wonderful people I owe a deep sense of gratitude especially now that this project has been completed.

G. T. S.

ACRONYMS

UnB	Universidade de Brasília
APC	Análise e Programação de Algoritmos

GLOSSARY

NormGibbs	Draw a sample from a posterior distribution of data with an unknown mean and variance using Gibbs sampling.
pNull	Test a one sided hypothesis from a numberically specified posterior CDF or from a sample from the posterior
sintegral	A numerical integration using Simpson's rule

SYMBOLS

A Amplitude
& Propositional logic symbol
a Filter Coefficient

$\mathcal{B}$ Number of Beats

INTRODUCTION

Sinayra Pascoal Cotts Moreira.

Departamento de Ciência da Computação - UnB
Brasília, DF, Brasil

O índice de reprovação nas matérias iniciais do curso de Ciência da Computação da UnB tem crescido a cada semestre, bem como o índice de evasão. Apesar das tentativas de criar mais horários de plantão de dúvidas e maior disponibilidade dos monitores para essas disciplinas, o desinteresse se mantém. Visando aumentar o interesse dos alunos pelo curso, está sendo desenvolvida uma biblioteca gráfica 2D simplificada denominada playAPC. Para o discente, a playAPC deve ser usada para consolidar os conceitos aprendidos em Análise e Programação de Algoritmos (APC) através de modelagem gráfica. Dessa forma, os alunos podem interagir com outras disciplinas do curso de modo lúdico.

A playAPC foi desenvolvida utilizando a linguagem C++, a API OpenGL e a biblioteca GLFW 2.7. A API OpenGL deve ser suportada pela placa de vídeo presente no computador, sendo exigido a versão 1.3 no mínimo. O tutorial para instalação tanto da GLFW quanto da própria playAPC está disponível em detalhes no site Guia de Referência da playAPC [1]. Apesar da playAPC ter sido desenvolvida em C++, o seu uso é focado primariamente para alunos que estejam a programar em C, ou seja, não é necessário conhec-

[1] http://pt-br.playcb.wikia.com/wiki/Categoria:Instala%C3%A7%C3%A3o

imento de C++ para utilizar a biblioteca, apenas utilizar a toolchain do g++ para compilar.

Neste livro, será disponibilizado uma série de exercícios usando da playAPC focando auxiliar os professores da Univerdade de Brasília (UnB) a desenvolverem novas práticas de laboratórios das turmas de APC.

REFERENCES

[1] OpenGL SuperBible. Pearson Education Inc, 6 edition, 2014.

[2] Marcus Geelnard and Camilla Berglund. GLFW - Reference guide, 2010. API version 2.7.

[3] Brian W. Kernighan and Dennis M. Ritchie. The C Programming Language. 1989.

[4] Stanley B. Lippman, Josés Lajoile, and Barbara Moo. C++ Primer. 2013.

PARTE I

ALGORITMOS SEQUENCIAIS, CONDICIONAIS E COM REPETIÇÕES

CAPÍTULO 1

ALGORITMOS SEQUENCIAIS

1.1 Resumo

Estrutura sequencial é um conjunto de instruções que serão executadas em sequência. A sequência de cada instrução deve ser seguida apara a realização de uma tarefa.

PROBLEMS

1.1 Exiba um plano cartesiano de -100 a 100 com espaçamento de 5 unidades.

1.2 Desenhe um boneco palito que utilize pelo menos uma vez as seguintes geometrias:

- Círculo
- Elipse
- Retângulo
- Triângulo

- Quadrado

1.3 Exiba a estrela de Davi.

1.2 Soluções

1.2.1 1.1Plano Cartesiano

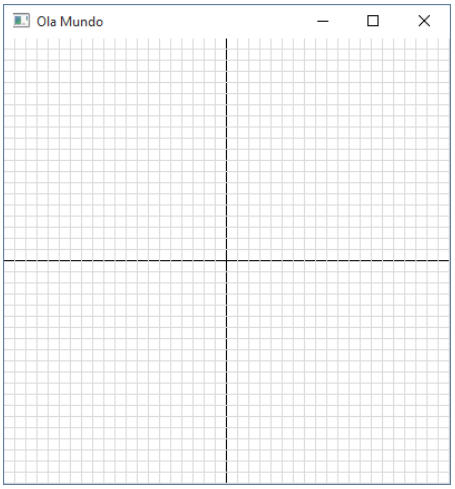

Figura 1.1 Plano Cartesiano de -100 à 100

Esta prática se refere a exibir um Plano Cartesiano na tela com espaçamento de 5 em 5 unidades, tanto no eixo x quanto no eixo y. Com ela, o aluno poderá notar a importância da ordem de chamada de funções da playCB e a necessidade das funções AbreJanela e Desenha, além de verificar, com um exemplo simples, se a playCB foi corretamente bem instalada.

Listagem 1.1 Código fonte de Plano Cartesiano

```
1  #include <playAPC/playapc.h>
2  int main(){
3
4      AbreJanela(400, 400, "Ola Mundo");
5
6      PintarFundo(255, 255, 255);
7      MostraPlanoCartesiano(5);
8
9      Desenha();
10 }
```

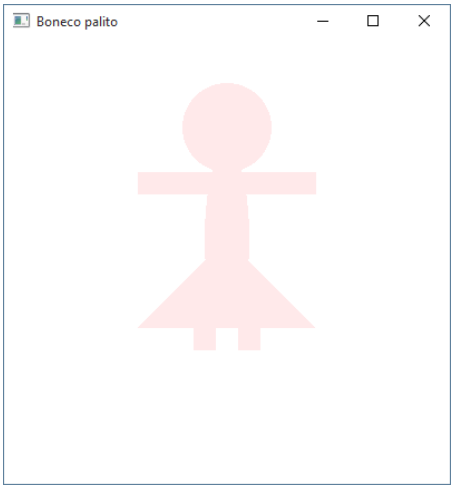

Figura 1.2 Boneco Palito

1.2.2 1.2Boneco Palito

Esta prática se refere a exibir um boneco palito e praticar a grande maioria das geometrias pré-definidas existentes na playCB. Os argumentos de cada função podem ser consultados no Guia de Referência da playCB [1]

Listagem 1.2 Código fonte do boneco palito

```
1  #include <playAPC/playapc.h>
2
3  int main(){
4      Ponto p;
5      AbreJanela(400, 400, "Boneco palito");
6      PintarFundo(255, 255, 255);
7
8      p.x = 0;
9      p.y = 60;
10     CriaCirculo(20, p); //(raio, ponto central)
11     Pintar(255, 233, 234);
12
13     p.y = 10;
14     CriaElipse(10, 40, p); //(metade do maior raio da elipse,
           ↪ metade do menor raio da elipse, ponto central)
15     Pintar(255, 233, 234);
16
17     p.x = -40;
18     p.y = 30;
19     CriaRetangulo(80, 10, p); //(base, altura, ponto esquerdo
           ↪ inferior)
20     Pintar(255, 233, 234);
```

[1] http://pt-br.playcb.wikia.com/wiki/Categoria:Geometrias

```
21
22      p.x = -40;
23      p.y = -30;
24      CriaTriangulo(80, 40, p); //(base, altura, ponto esquerdo
            ↪ inferior)
25      Pintar(255, 233, 234);
26
27      p.x = -15;
28      p.y = -40;
29      CriaQuadrado(10, p); //(lado, ponto esquerdo inferior)
30      Pintar(255, 233, 234);
31
32      p.x = 5;
33      p.y = -40;
34      CriaQuadrado(10, p);
35      Pintar(255, 233, 234);
36
37      Desenha();
38  }
```

1.2.3 1.3Estrela de Davi

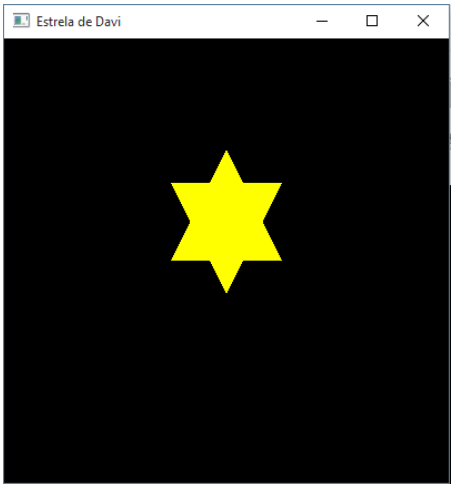

Figura 1.3 Estrela de Davi

Esta prática se refere a exibir a estrela de Davi, feita com dois triângulos. Um triângulo foi criado com a função CriaTriangulo e o outro com a função CriaPoligono. Verificamos nesta prática os argumentos de CriaTriangulo (base, altura e ponto esquerdo inferior) e, como não há como ter altura negativa, teve a necessidade de criar um polígono definido pelos três pontos p1, p2 e p3 para criar-se um triângulo de cabeça pra baixo.

Listagem 1.3 Código fonte da Estrela de Davi

```
1  #include <playAPC/playapc.h>
2
3  int main(){
4      Ponto p1, p2, p3;
5      AbreJanela(400, 400, "Estrela de Davi");
6
7      p1.x = -25;
8      p1.y = 0;
9      CriaTriangulo(50, 50, p1);
10     Pintar(255, 255, 0);
11
12     p1.x = -25;
13     p1.y = 35;
14
15     p2.x = 25;
16     p2.y = 35;
17
18     p3.x = 0;
19     p3.y = -15;
20     CriaPoligono(3, p1, p2, p3);
21     Pintar(255, 255, 0);
22
23     Desenha();
24 }
```

CAPÍTULO 2

ALGORITMOS CONDICIONAIS

2.1 Resumo

Estrutura condicional expõe que a instrução ou bloco de instrução só seja executada se a condição for verdadeira.

PROBLEMS

2.1 Escreva um programa que receba do usuário um valor de ângulo em graus, converta para radianos, exiba uma reta com comprimento de 50 unidades e pinte-a de acordo com as seguintes regras:

- Se a reta pertencer ao primeiro quadrante, pinte-a de vermelho
- Se a reta pertencer ao segundo quadrante, pinte-a de verde
- Se a reta pertencer ao terceiro quadrante, pinte-a de azul
- Se a reta pertencer ao quarto quadrante, pinte-a de preto

.

2.2 Soluções

2.2.1 2.1Quadrante de uma reta

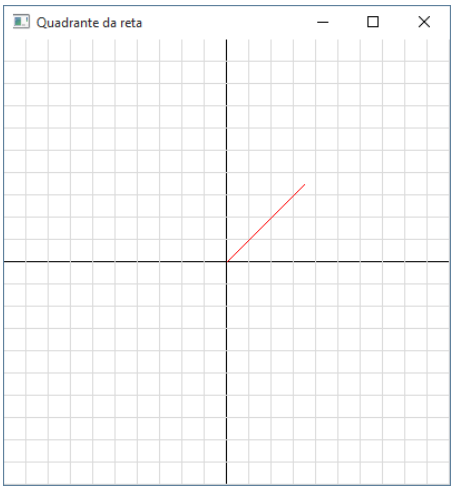

Figura 2.1 Determinação do quadrante de uma reta baseado no ângulo de inclinação dela

Esta prática exibe uma reta com cor variada de acordo com qual quadrante ela pertence. A função Pintar neste caso se refere a única geometria criada no programa, no caso, a reta.

Listagem 2.1 Código fonte do quadrante da reta

```
1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <math.h>
4
5  #define HIP 50
6  #define PI 3.14
7  int main(){
8      int angulo;
9      float anguloRad;
10     Ponto p1, p2;
11
12     printf("Digite um angulo de 0 a 360 graus:");
13     scanf("%d", &angulo);
14
15     p1.x = 0;
16     p2.x = 0;
17
18     anguloRad = (PI * angulo)/180;
19
20     p2.y = sin(anguloRad) * HIP;
21     p2.x = cos(anguloRad) * HIP;
22
```

```
23      AbreJanela(400, 400, "Quadrante da reta");
24      PintarFundo(255, 255, 255);
25      MostraPlanoCartesiano(10);
26
27      CriaReta(p1, p2);
28
29      if(p2.x > 0){
30          if(p2.y > 0)
31              Pintar(255, 0, 0); //vermelho: 1 quadrante
32          else
33              Pintar(0, 0, 0); //preto: 4 quadrante
34      }
35      else{
36          if(p2.y > 0)
37              Pintar(0, 255, 0); //verde: 2 quadrante
38          else
39              Pintar(0, 0, 255); //azul: 3 quadrante
40      }
41
42      Desenha();
43
44  }
```

CAPÍTULO 3

ALGORITMOS COM REPETIÇÃO

3.1 Resumo

Estruturas de repetição são criadas para que diversas instruções sejam executadas um determinado número de vezes, enquanto a condição se manter verdadeira.

PROBLEMS

3.1 Sabendo que a equação hiperbólica pode ser defina por

$$\begin{aligned} x &= a\cos(\theta) \\ y &= a\sin(\theta) \end{aligned}.$$

onde a é a assíntota para y e θ o ângulo equivalente ao ângulo em coordenadas polares, exiba duas espirais hiberbólicas, onde uma delas está invertida em relação a outra e coloque-as para girar.

3.2 Exiba um carrinho se movendo de -100 à 100.

3.3 Construa um moinho de vento e coloque apenas as hélices para girar.

3.2 Soluções

3.2.1 3.1Galáxia espiral

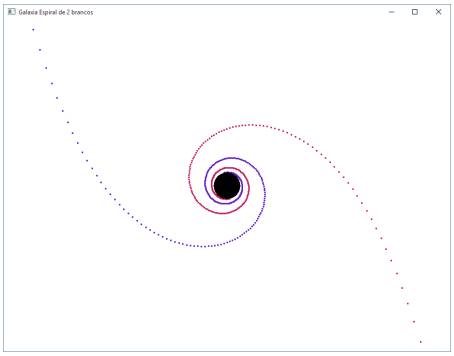

Figura 3.1 Duas espirais hiperbólicas girando

Esta prática ilustra como a função Desenha1Frame funciona. Na linha 23 até a linha 30, a cada frame são criados dois pontos, um de cada espiral.

Listagem 3.1 Código fonte da galáxia expiral

```
1  #include <playAPC/playapc.h>
2  #include <math.h>
3
4  int main (int argc, char * argv[]) {
5
6    AbreJanela (960,960, "Galaxia Espiral de 2 brancos");
7    /*
8      Espiral Hiperbólica: equação em coordenadas cartesianas
9         x = a*cos(t)/t
10        y = a*sin(t)/t
11
12        a é a assintota para y (reta paralela ao eixo x)
13        t equivalente ao angulo em coordenadas polares
14    */
15   Ponto p, q, r;
16
17   //  for (double t = 0; t < 4*PI; t += .01){
18
19     /* espiral hiperbolica, caminhando do "fim" pro centro (0,0)
20     p.x = 100*cos(t)/t;
21     p.y = 100*sin(t)/t;
22     */
23   for (double t = 4*PI; t > 0 ; t -= .05) {
24     p.x = 100*cos(t)/t;          q.x = -p.x;
25     p.y = 100*sin(t)/t;          q.y = -p.y;
26
27     CriaPonto (p);     Pintar (200, 30, 100);     Grafite(3);
28     CriaPonto (q);     Pintar (100, 30, 200);     Grafite(3);
```

```
29
30      Desenha1Frame();
31    }
32
33
34    //A massive Black Hole in the very centre
35    //If you want to see (the unseeable) black hole
36    //paint the background on a different colour
37    r.x =0;      r.y = 0;
38    CriaCirculo(8, r);
39    Pintar (0, 0,  0);
40
41    for (double t=0;  ; t += .5) {
42      Gira(t);
43      Desenha1Frame();
44
45      //Depois de um tempinho, pinta o fundo de branco pra mostrar
46      //o buraco negro
47      if ( t > 200 ) PintarFundo (255, 255, 255);
48
49      //quebra o loop e encerra o programa
50      if(ApertouTecla(GLFW_KEY_ENTER)) return 0;
51    }
52
53  }
```

3.2.2 3.2Carro andando

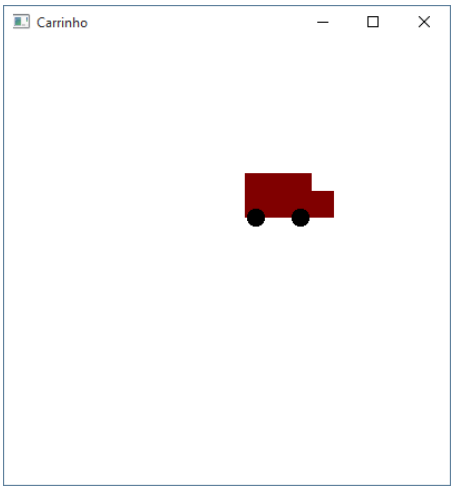

Figura 3.2 Carro se movendo da posição -100 até a posição 100

Esta prática exibe um carro construído com dois retângulos e dois círculos, agrupados com a função CriaGrupo, movendo-se da posição -100 até a posição

100. Nota-se que todas as geometrias que estão abaixo da função CriaGrupo pertencem a um único grupo, o grupo carro.

Listagem 3.2 Código fonte do carro andando

```
#include <playAPC/playapc.h>

int main(){
    Ponto p;
    int carro;

    AbreJanela(400, 400, "Carrinho");
    PintarFundo(255, 255, 255);

    carro = CriaGrupo();
        p.x = -100;
        p.y = 20;
        CriaRetangulo(30, 20, p);
        Pintar(128, 0, 0);

        p.x = -80;
        p.y = 20;
        CriaRetangulo(20, 12, p);
        Pintar(128, 0, 0);

        p.x = -95;
        p.y = 20;
        CriaCirculo(4, p);
        Pintar(0, 0, 0);

        p.x = -75;
        p.y = 20;
        CriaCirculo(4, p);
        Pintar(0, 0, 0);

    p.y = 20;
    for(p.x = -100; p.x < 100; p.x++){
        Move(p, carro);
        Desenha1Frame();
    }
    Desenha();
}
```

3.2.3 3.3Moinho de vento

Esta prática exibe um moinho de vento criado com um grupo composto por um triângulo e um retângulo, o grupo moinho, e outro grupo composto pelas hélices, o grupo grupo. Somente o grupo sofre a ação de girar. [1]

Listagem 3.3 Código fonte do moinho

```
/* *
```

[1]Exemplo criado pelo aluno Pedro Paulo de Pinho Matos, da turma de Computação Básica de 1/2014

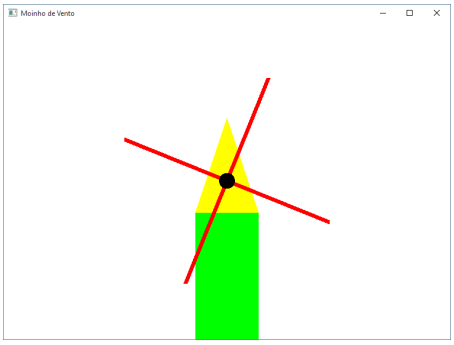

Figura 3.3 Moinho de vento

```
2  UnB - Professor : Ralha - Computação Básica
3  Aluno : Pedro Paulo de Pinho Matos
4  Matrícula : 14/0070354
5  */
6
7 #include <stdlib.h>
8 #include <stdio.h>
9 #include <playAPC/playapc.h>
10
11
12 int main (int argc, char* argv[]){
13  int angulo =1;
14
15  AbreJanela(1024, 768, "Moinho de Vento" );
16  PintarFundo(255,255,255);
17  Ponto p,q, r;
18
19  int moinho = CriaGrupo();
20  Ponto x, y;
21  x.x = -20; x.y = -20; CriaTriangulo(40,60,x); Pintar(255,255,0);
22  y.x = -20; y.y = -100; CriaRetangulo(40,80,y); Pintar(0,255,0);
23
24  int grupo = CriaGrupo(); // Hélice 1
25  q.x = 0; q.y = 0; r.x = 0; r.y = 70;
26  CriaReta(q,r); Pintar(255,0,0); Grafite(8);
27
28  // Hélice 2
29  q.x = 0; q.y = 0; r.x = 70; r.y = 0;
30  CriaReta(q,r); Pintar(255,0,0); Grafite(8);
31
32  // Helice 3
33  q.x = 0; q.y = 0; r.x = 0; r.y = -70;
34  CriaReta(q,r); Pintar(255,0,0); Grafite(8);
35
36  // Helice 4
37  q.x = 0; q.y = 0; r.x = -70; r.y = 0;
38  CriaReta(q,r); Pintar(255,0,0); Grafite(8);
39
40  p.x = 0; p.y = 0;
```

```
41 ||  CriaCirculo(5,p); Pintar(0,0,0); Grafite(25);
42 ||
43 ||  while( angulo > 0){
44 ||       Desenha1Frame();
45 ||       Gira(angulo,grupo);
46 ||       angulo ++;
47 ||  }
48 ||
49 ||  return 0;
50 || }
```

PARTE II

ESTRUTURA DE DADOS N-DIMENSIONAIS HOMOGÊNEAS

CAPÍTULO 4

VETORES

4.1 Resumo

Vetores são um tipo de estrutura que podem armazenar um tamanho fixo de elementos do mesmo tamanho e mesmo tipo, alocados em memória contígua. Utiliza-se vetores como um tipo de lista unidimensional, acessada através de índices.

PROBLEMS

4.1 Exiba o gráfico do polinômio $-x^3$ para $-50 \leq x \leq 50$.

4.2 Soluções

4.2.1 5.1Criando um gráfico

Esta prática mostra como construir um gráfico a partir de um vetor de Pontos. Cada posição em y de cada ponto é calculada dentro do loop. Por padrão, os limites da janela de exibição da playCB vão de -100 à 100, entretanto, os

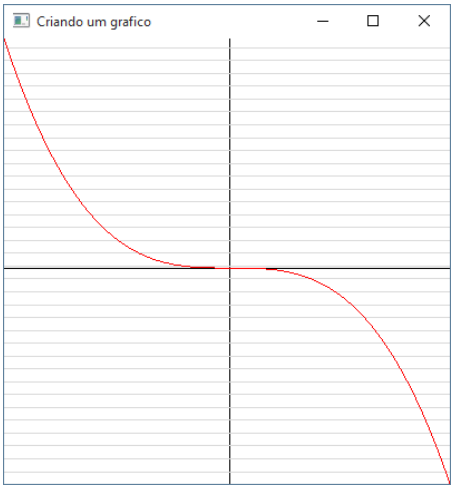

Figura 4.1 Gráfico do polinômio $-x^3$

valores em y nesta função variam de −125.000 até 125.000, tendo a necessidade de mudar o limite de exibição com a função MudaLimitesJanela(125000).

Listagem 4.1 Código fonte de polinômio

```
#include <playAPC/playapc.h>
#include <math.h>

int main(){
    Ponto p[100];
    int i, j;

    MudaLimitesJanela(125000);

    AbreJanela(400, 400, "Criando um grafico");
    PintarFundo(255, 255, 255);
    MostraPlanoCartesiano(7000);

    j = -50;
    for(i = 0; i < 100; i++, j++){
        p[i].x = j;
        p[i].y = -pow(p[i].x, 3);
    }

    CriaGrafico(100, p, 1);
    Pintar(255, 0, 0);

    Desenha();

}
```

CAPÍTULO 5

MATRIZES

5.1 Resumo

Assim como vetores, matrizes são um tipo de estrutura que armazena dados de mesmo tamanho e mesmo tipo, mas são utilizadas de maneira n-dimensional. O modo mais comum de utilizar matriz é usando-a na forma bidimensional, onde os dados são tratados como se estivessem numa tabela, com linhas e colunas.

PROBLEMS

5.1 Mostre graficamente o jogo da vida para uma matriz com 17 linhas e 17 colunas com a seguinte população inicial onde a população inicial estará VIVA para as seguintes posições na matriz:

$(1,5),(2,5),(3,5),(3,6),(5,1),(5,2),(5,3),(5,6),(5,7),(6,3),(6,5),(6,7),(7,5),(7,6)$

5.2 Soluções

5.2.1 5.1Jogo da Vida

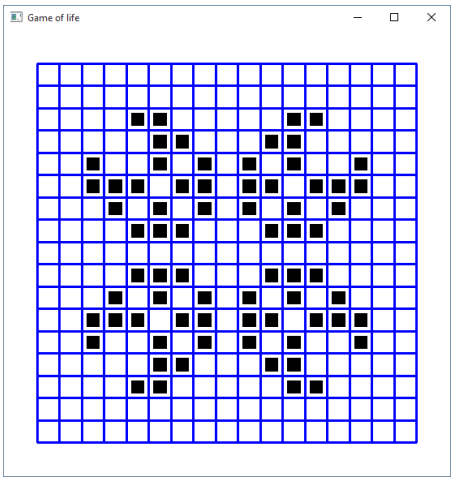

Figura 5.1 Pulsar

Esta prática mostra como utilizar o retorno das função CriaQuadrado, que esta retorna o índice da geometria criada. O seu índice é utilizado na função Pintar, que recebe, além do índice, o tipo da geometria. Como foi utilizado a função CriaQuadrado, o tipo de geometria é QUADRADO. Se fosse utilizado CriaCirculo, seria utilizado o tipo CIRCULO e assim sucessivamente.

Listagem 5.1 Código fonte do jogo da vida

```
1  #include <playAPC/playapc.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include<time.h>
5  #define LINHAS 17
6  #define COLUNAS 17
7  #define VIVO  1
8  #define MORTO 0
9
10 void waitFor (unsigned int );
11
12 int main(){
13
14     int geracaoAtual[LINHAS][COLUNAS], proximaGeracao[LINHAS][
       ↪ COLUNAS], geoIndex[LINHAS][COLUNAS];
15     int l, c, i, j, soma, r, s, n, ESTADO, deltaQuad=10, largQuad =
       ↪  6, marg = 2;
16     Ponto p1, p2, p3;
17
18     ////////////////////////////////
```

```
19      // Define o Janela            //
20      ////////////////////////////////
21      AbreJanela(550, 550, "Game of life");
22      PintarFundo(255, 255, 255);
23
24      ////////////////
25      // Desehna Grid //
26      ////////////////
27
28      // Desenha as retas horizontais
29      p1.x = -85;
30      p1.y =  85;
31      p2.x =  85;
32      p2.y =  85;
33      for(i = 0; i < LINHAS+1; i++){
34      CriaReta(p1, p2); Pintar(0, 0, 255); Grafite(3); //cima
35      p1.y -=  deltaQuad;
36      p2.y -=  deltaQuad;
37      }
38      // Desenha as retas verticais
39      p1.x = -85;
40      p1.y =  85;
41      p2.x =  -85;
42      p2.y =  -85;
43      for(i = 0; i < COLUNAS+1; i++){
44      CriaReta(p1, p2); Pintar(0, 0, 255); Grafite(3); //cima
45      p1.x +=  deltaQuad;
46      p2.x +=  deltaQuad;
47      }
48
49      // Desenha celulas
50      p1.y =  85-deltaQuad+marg;
51      for(i = 0; i < LINHAS; i++){
52      p1.x = -85+marg;
53          for(j = 0; j < COLUNAS; j++){
54              geoIndex[i][j] = CriaQuadrado(largQuad, p1);
55              Pintar(0, 0, 0, QUADRADO, geoIndex[i][j]);
56              p1.x += deltaQuad;
57          }
58      p1.y -=deltaQuad;
59      }
60
61      ////////////////////
62      // Joga o jogo    //
63      ////////////////////
64      // Preenche o fundo com o valor MORTO
65      for(l=0; l<LINHAS; l++){
66          for(c=0; c<COLUNAS; c++){
67              geracaoAtual[l][c]=MORTO;
68          }
69      }
70
71      // Define as celulas vivas no no canto superior esquerdo
72      geracaoAtual[1][5] = VIVO;
73      geracaoAtual[2][5] = VIVO;
74      geracaoAtual[3][5] = VIVO; geracaoAtual[3][6] = VIVO;
```

```
75      geracaoAtual[5][1] = VIVO; geracaoAtual[5][2] = VIVO;
            ↪ geracaoAtual[5][3] = VIVO; geracaoAtual[5][6] = VIVO;
            ↪ geracaoAtual[5][7] = VIVO;
76      geracaoAtual[6][3] = VIVO; geracaoAtual[6][5] = VIVO;
            ↪ geracaoAtual[6][7] = VIVO;
77      geracaoAtual[7][5] = VIVO; geracaoAtual[7][6] = VIVO;
78
79      // Realiza a reflexao do padrao
80      for(l = 0; l < LINHAS/2; l++){
81          for(c = 0; c < COLUNAS/2; c++){
82              geracaoAtual[(LINHAS-1)-l][c] = geracaoAtual[l][c];//
                    ↪ Inferior esquerdo
83              geracaoAtual[l][(COLUNAS-1)-c] = geracaoAtual[l][c];//
                    ↪ Superior direito
84              geracaoAtual[(LINHAS-1)-l][(COLUNAS-1)-c] =
                    ↪ geracaoAtual[l][c]; // Inferior direito
85          }
86      }
87
88      while(1){
89
90          // Faz a copia da geracao atual para a geracao anterior
91          for(l = 0; l < LINHAS; l++){
92              for(c = 0; c < COLUNAS; c++){
93                  proximaGeracao[l][c] = geracaoAtual[l][c];
94              }
95          }
96
97          for(l=0; l<LINHAS; l++){
98              for(c=0; c<COLUNAS; c++){
99                  if(proximaGeracao[l][c]==1){
100                     printf("%c", 219);
101                     Pintar(0, 0, 0, QUADRADO, geoIndex[l][c]);
102                 } else{
103                     printf("%c", proximaGeracao[l][c], ' ');
104                     Pintar(255, 255, 255, QUADRADO, geoIndex[l][c])
                            ↪ ;
105                 }
106             }
107             printf("\n");
108         }
109          Desenha1Frame();
110
111         // Conta a quantidade de vizinhos de uma celula
112         for (l = 1; l < LINHAS-1; l++){
113             for(c = 1; c < COLUNAS-1; c++){
114                 ESTADO = proximaGeracao[l][c];
115                 soma = 0;
116                 for(r = l-1; r < l+2; r++)
117                     for(s = c-1; s < c+2; s++)
118                             soma+=proximaGeracao[r][s];
119
120                 if(proximaGeracao[l][c] == VIVO) soma--;
121
122                  //Define se uma celula deve morrer, permanecer
                        ↪ viva ou nascer
```

```
123                 if ((ESTADO == VIVO && soma < 2) || (ESTADO == VIVO
                        ↪ && soma > 3)){
124                     geracaoAtual[l][c] = MORTO;
125                 }else if ((ESTADO == VIVO && (soma > 2 || soma <=
                        ↪ 3)) || (ESTADO == MORTO && soma == 3)){
126                     geracaoAtual[l][c] = VIVO;
127                 }
128             }
129         }
130         waitFor (1);
131         system("cls");
132     }
133 
134     Desenha();
135     return 0;
136 }
137 
138 void waitFor (unsigned int secs) {
139     unsigned int retTime;
140     retTime = time(0) + secs;      // Get finishing time.
141     while (time(0) < retTime);     // Loop until it arrives.
142 }
```

PARTE III

FUNÇÕES

CAPÍTULO 6

FUNÇÕES

6.1 Resumo

Uma função é um conjunto de instruções que, ao final da função, executa uma tarefa. Todo programa C possui pelo menos uma função, a main.

PROBLEMS

6.1 Crie o jogo Snake com as seguintes configurações

- A cabeça não pode estar na mesma posição que o corpo

- A cabeça não pode estar na mesma posição que a parede

SUGESTÃO: Utilize a lógica do exercício 4.1

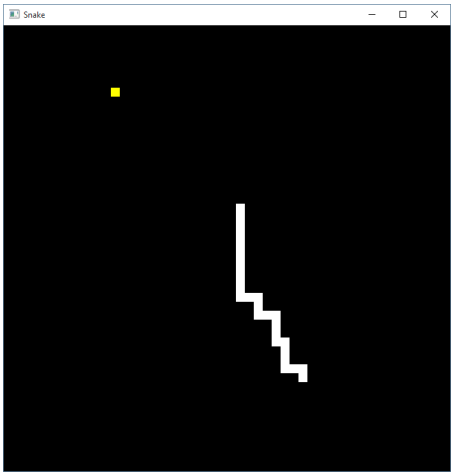

Figura 6.1 Jogo Snake

6.2 Soluções

6.2.1 6.1Snake

Esta prática ilustra como a função ApertaTecla e MudaLimitesJanela podem ser utilizadas: a primeira para lidar com input de teclado [1] e a segunda para ajustar o plano onde as geometrias serão desenhadas.

Listagem 6.1 Código fonte de Snake

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <playAPC/playapc.h>
4
5  #define ESQ GLFW_KEY_LEFT
6  #define DIR GLFW_KEY_RIGHT
7  #define CIMA GLFW_KEY_UP
8  #define BAIXO GLFW_KEY_DOWN
9
10 #define TAM 50
11
12 typedef enum{
13     CABECA,
14     CORPO,
15     COMIDA,
16     VAZIO
17 }tipoCobra;
18
19 typedef struct{
20     int direcao; //se tipo nao for vazio, indica direcao
```

[1] http://pt-br.playcb.wikia.com/wiki/Aperta_Tecla

```
21      tipoCobra tipo; //como sera pintado
22      int index; //indice do quadrado a ser pintado
23  }tipoCelula;
24
25  void inicializaMatriz(tipoCelula m[TAM][TAM], int pos_i, int pos_j)
        ↪ {
26      Ponto p;
27
28      printf("Aguarde enquanto o jogo inicializa\n");
29
30      p.y = TAM/2 - 1;
31      for(int i = 0; i < TAM; i++){
32          p.x = -(TAM/2);
33          for(int j = 0; j < TAM; j++){
34              m[j][i].index = CriaQuadrado(1, p);
35              m[j][i].tipo = VAZIO;
36              //printf("m[%d][%d] = %d - P(%f,%f)\n", i, j, m[i][j].
                    ↪ index, p.x, p.y);
37              Pintar(0, 0, 0);
38              p.x++;
39          }
40          p.y--;
41      }
42      Pintar(255, 255, 255, QUADRADO, m[pos_i][pos_j].index);
43      m[pos_i][pos_j].direcao = CIMA;
44      m[pos_i][pos_j].tipo = CABECA;
45  }
46
47  int bateu(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
48      if(m[pos_i][pos_j].tipo == CORPO || m[pos_i][pos_j].tipo ==
            ↪ CABECA){
49          printf("CORPO\n");
50          return 1;
51      }
52      else if(pos_i >= TAM || pos_j >= TAM || pos_i < 0 || pos_j < 0)
            ↪ {
53          printf("LIMITE DA TELA\n");
54          return 1;
55      }
56
57      return 0;
58  }
59  void atualizaPosicao(int direcao, int *npos_i, int *npos_j){
60       switch(direcao){
61          case ESQ:
62              (*npos_i)--;
63          break;
64          case DIR:
65              (*npos_i)++;
66          break;
67          case CIMA:
68              (*npos_j)--;
69          break;
70          case BAIXO:
71              (*npos_j)++;
72          break;
```

```
73      }
74  }
75
76  void updateTeclado(int *direcao, int *npos_i, int *npos_j){
77      if(ApertouTecla(ESQ) && *direcao != DIR)
78          (*direcao) = ESQ;
79
80      if(ApertouTecla(DIR) && *direcao != ESQ)
81          (*direcao) = DIR;
82
83      if(ApertouTecla(CIMA) && *direcao != BAIXO)
84          (*direcao) = CIMA;
85
86      if(ApertouTecla(BAIXO) && *direcao != CIMA)
87          (*direcao) = BAIXO;
88
89      atualizaPosicao((*direcao), npos_i, npos_j);
90  }
91
92  //retorna se comeu comida
93  int updateCabeca(tipoCelula m[TAM][TAM], int pos_i, int pos_j, int
        ↪ direcao){
94      int comeu = 0;
95
96      Pintar(255, 255, 255, QUADRADO, m[pos_i][pos_j].index);
97      m[pos_i][pos_j].direcao = direcao;
98
99      if(m[pos_i][pos_j].tipo == COMIDA)
100         comeu = 1;
101
102     m[pos_i][pos_j].tipo = CABECA;
103
104     return comeu;
105 }
106
107 void updateRastro(tipoCelula m[TAM][TAM], int pos_i, int pos_j){
108     Pintar(0, 0, 0, QUADRADO, m[pos_i][pos_j].index);
109     m[pos_i][pos_j].tipo = VAZIO;
110 }
111
112 void sorteiaComida(tipoCelula m[TAM][TAM]){
113
114     int pos_i, pos_j;
115     do{
116         pos_i = rand()%TAM;
117         pos_j = rand()%TAM;
118     }while(m[pos_i][pos_j].tipo != VAZIO);
119
120     Pintar(255, 255, 0, QUADRADO, m[pos_i][pos_j].index);
121
122     m[pos_i][pos_j].tipo = COMIDA;
123 }
124
125 int main(){
126     tipoCelula m[TAM][TAM]; //-100 a 100 pra cima e pra baixo, cada
        ↪  quadrado
```

```
127     int pos_i = TAM/2; //posicao i da cabeca
128     int pos_j = TAM/2; //posicao j da cabeca
129
130     int rpos_i = pos_i; //posicao i do rabo
131     int rpos_j = pos_j; //posicao j do rabo
132
133     int direcao = CIMA;
134     MudaLimitesJanela(TAM/2);
135     AbreJanela(650, 650, "Snake");
136     PintarFundo(255, 0, 0);
137     //MostraPlanoCartesiano(5);
138
139     inicializaMatriz(m, pos_i, pos_j);
140
141     sorteiaComida(m);
142
143     while(1){
144         int npos_i = pos_i, npos_j = pos_j;
145
146         updateTeclado(&direcao, &npos_i, &npos_j);
147         m[pos_i][pos_j].direcao = direcao; //ultima posicao da
                ↪ cabeca recebe direcao que cabeca foi
148         if(!bateu(m, npos_i, npos_j)){
149             int comeu;
150
151             comeu = updateCabeca(m, npos_i, npos_j, direcao);
152             if(comeu)
153                 sorteiaComida(m);
154             else{
155                 updateRastro(m, rpos_i, rpos_j);
156                 atualizaPosicao(m[rpos_i][rpos_j].direcao, &rpos_i,
                        ↪ &rpos_j);
157                 m[rpos_i][rpos_j].tipo = CORPO;
158             }
159
160             pos_i = npos_i;
161             pos_j = npos_j;
162
163             printf("C(%d, %d)\t R(%d, %d)\n", pos_i, pos_j, rpos_i,
                    ↪ rpos_j);
164         }
165         else
166             break;
167
168         Desenha1Frame();
169     }
170
171     printf("O jogo acabou!\n");
172
173     Desenha();
174     return 0;
175 }
```