UNIVERSIDADE FEDERAL DE ITAJUBÁ *CAMPUS* ITABIRA

Documentação do Trabalho de Computação Gráfica - ECO034

Alunos: Marco Antonio
Filipe Pena
Douglas Venâncio

Sumário

1 Implementação do Jogo PAC-MAN para disciplina CG **1**

1.1 Tópicos de computação gráfica abordados no jogo. 1

1.1.1 Renderização de objetos na cena 1

1.1.2 Transformações 1

1.1.3 Renderização de objetos na cena 1

1.1.4 Iluminação 2

1.1.5 Textura 2

1.2 Outros Tópicos 3

1.2.1 Colisão 3

1.2.2 Inteligência do fantasma 3

2 Índice Hierárquico **5**

2.1 Hierarquia de Classes 5

3 Índice das Estruturas de Dados **7**

3.1 Estruturas de Dados 7

4 Índice dos Arquivos **9**

4.1 Lista de Arquivos 9

5 Estruturas **11**

5.1 Referência da Classe Fantasma . . . 11

5.1.1 Descrição detalhada . . . 12

5.1.2 Construtores e Destrutores . . . 12

5.1.2.1 Fantasma() . . . 12

5.1.3 Funções membros . . . 13

5.1.3.1 vertex() . . . 13

5.1.3.2 vertex2() . . . 13

5.1.3.3 move() . . . 14

5.1.3.4 desenha() . . . 14

5.1.3.5 animacao() . . . 17

5.1.3.6 stop() . . . 17

5.1.3.7 aleatorio() . . . 18

5.1.3.8 praonde() . . . 18

5.1.3.9 aumentanivel() . . . 20

5.1.3.10 colisao() . . . 20

5.1.3.11 conttempo() . . . 21

5.2 Referência da Classe Fruta . . . 21

5.2.1 Descrição detalhada . . . 22

5.2.2 Funções membros . . . 22

5.2.2.1 vertex() . . . 22

5.2.2.2 vertex2() . . . 23

5.2.2.3 colisao() . . . 23

5.3 Referência da Classe Jogo . . . 24

5.3.1 Descrição detalhada . . . 24

5.3.2 Construtores e Destrutores . . . 25

5.3.2.1 Jogo() . . . 25

5.3.3 Funções membros . . . 25

5.3.3.1 desenha() . . . 25

5.3.3.2 atualizatamanho() . . . 26

5.3.3.3 KeyboardHandle() . . . 26
5.3.3.4 SpecialKeyHandle() . . . 27
5.3.3.5 atualiza() . . . 27
5.3.3.6 newgame() . . . 29
5.3.3.7 gameover() . . . 30
5.4 Referência da Classe Menu . . . 30
5.4.1 Descrição detalhada . . . 31
5.4.2 Construtores e Destrutores . . . 31
5.4.2.1 Menu() . . . 31
5.4.3 Funções membros . . . 32
5.4.3.1 setasmenu() . . . 32
5.4.3.2 getpause() . . . 33
5.4.3.3 getstart() . . . 33
5.4.3.4 setstart() . . . 33
5.4.3.5 pausa() . . . 34
5.4.3.6 entermenu() . . . 34
5.4.3.7 setpontos() . . . 35
5.4.3.8 setptsgameover() . . . 35
5.4.3.9 setlife() . . . 35
5.4.3.10 desenha() . . . 36
5.4.3.11 texto() . . . 37
5.4.3.12 numero() . . . 38
5.5 Referência da Classe Objetos . . . 39
5.5.1 Descrição detalhada . . . 39
5.5.2 Funções membros . . . 39
5.5.2.1 desenha() . . . 39
5.5.2.2 colisao() . . . 39
5.6 Referência da Classe Pacman . . . 40
5.6.1 Descrição detalhada . . . 41
5.6.2 Construtores e Destrutores . . . 41

5.6.2.1 Pacman() . . . 41
5.6.3 Funções membros . . . 42
5.6.3.1 vertex() . . . 42
5.6.3.2 vertex2() . . . 42
5.6.3.3 direcao() . . . 43
5.6.3.4 stop() . . . 44
5.6.3.5 comeu() . . . 45
5.6.3.6 getx() . . . 45
5.6.3.7 gety() . . . 45
5.6.3.8 getz() . . . 46
5.6.3.9 getr() . . . 46
5.6.3.10 getangulo() . . . 46
5.6.3.11 vida() . . . 47
5.6.3.12 temvida() . . . 47
5.6.3.13 pontos() . . . 47
5.6.3.14 setpontos() . . . 47
5.7 Referência da Classe Parede . . . 48
5.7.1 Descrição detalhada . . . 49
5.7.2 Construtores e Destrutores . . . 49
5.7.2.1 Parede() . . . 49
5.7.3 Funções membros . . . 50
5.7.3.1 desenha() . . . 50
5.7.3.2 colisao() . . . 51

6 Arquivos **53**
6.1 Referência do Arquivo Fantasma.cpp . . . 53
6.1.1 Descrição detalhada . . . 53
6.2 Referência do Arquivo Fruta.cpp . . . 53
6.2.1 Descrição detalhada . . . 54
6.3 Referência do Arquivo jogo.cpp . . . 54
6.3.1 Descrição detalhada . . . 54
6.4 Referência do Arquivo main.cpp . . . 55
6.4.1 Descrição detalhada . . . 55
6.5 Referência do Arquivo Menu.cpp . . . 56
6.5.1 Descrição detalhada . . . 56
6.6 Referência do Arquivo Pacman.cpp . . . 56
6.6.1 Descrição detalhada . . . 57
6.7 Referência do Arquivo parede.cpp . . . 57
6.7.1 Descrição detalhada . . . 57
6.8 Referência do Arquivo tga.cpp . . . 57
6.8.1 Descrição detalhada . . . 58
6.8.2 Funções . . . 58
6.8.2.1 loadTGA() . . . 58

Capítulo 1

Implementação do Jogo PAC-MAN para disciplina CG

1.1 Tópicos de computação gráfica abordados no jogo.

1.1.1 Renderização de objetos na cena

Para desenhar os objetos no cenario do jogo, foram usadas as formas primitivas do Open-GL e destribuidas no espaço para criar formas geometricas 3D conforme suas esquações de modelagem. Para desenhar figuras mais complexas como uma esfera e um cilindro, foram aplicadas mudanças no sistema de referência usando coordenadas esféricas e cilíndricas.

1.1.2 Transformações

Para que um objeto possa sofrer alterações na cena, é necessário realizar operações no seu conjunto de pontos. Para esse Jogo as principais transformações usadas foram rotações e tranlações. Como o sistema de referência é ortogonal ao eixo y, a translação ocorre nos eixos x e z e a rotação m torno do eixo y.

1.1.3 Renderização de objetos na cena

Foram também aplicados conceitos referentes a configuração de câmeras. Foram implementadas três configurações de câmeras, a primeira toma uma visão completa do jogo, onde a camera fica estática e aponta para o centro do mapa e o VUP na diração de Y. A segunda configuração da zoom no mapa e câmera é apontada para posição em que esta o pacman. A terceira câmera segue o mesmo principio da anterior, porém em uma configuração que possibilita uma visão em terceira pessoa.

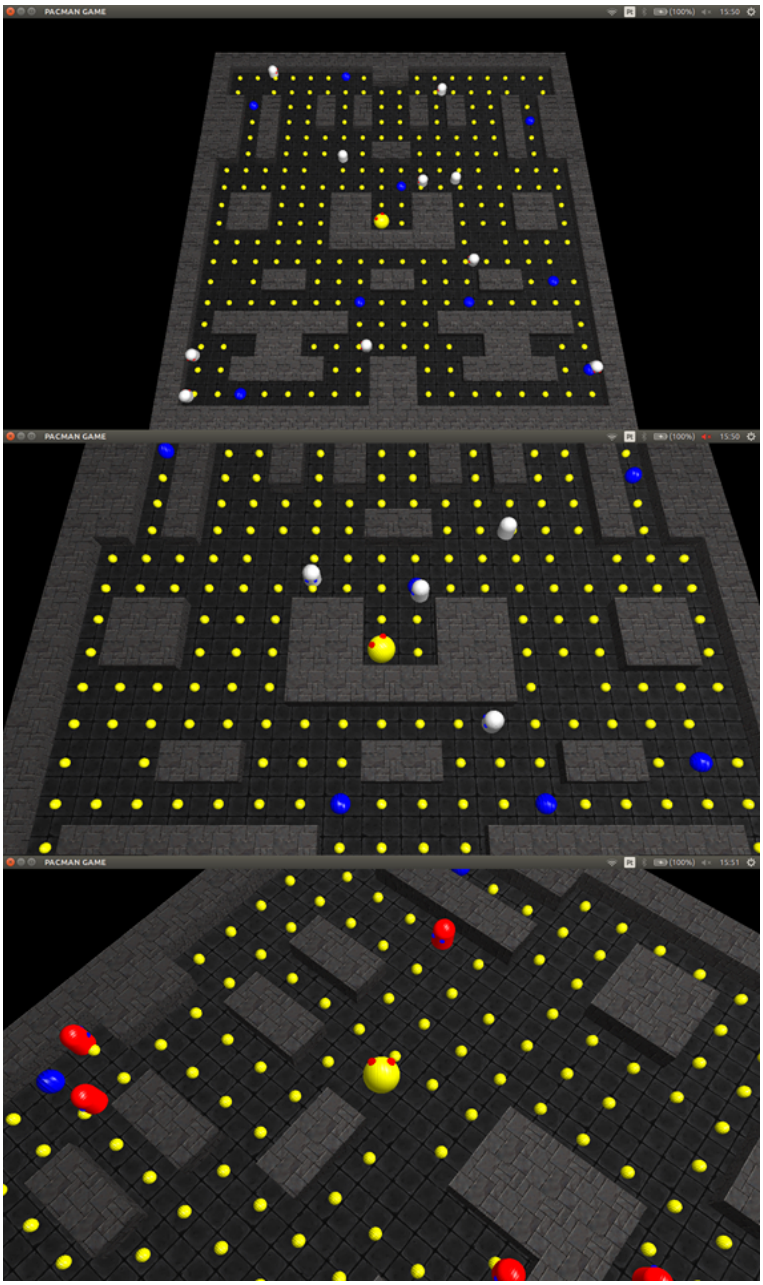


Figura 1.1 colisao

1.1.4 Iluminação

Para uma boa visualização dos objetos em 3D e além disso uma melhor representação de uma situação real, é necessario aplicar a iluminação. O Open-GL se baseia no modelo de iluminação de Phong para implementar a iluminação, então para configurar o a luz foram definidos parâmetros para as reflexões ambientais, difusa e especular.

1.1.5 Textura

Para criar formas mais realistas, é necessária a utilização de textura, visto que sem aplicar textura e conseguir o mesmo resultado demandaria muito processamento. O Open-GL disponibilisa funções que realizam manipulação como, cortar, escaçar e repitir ou um conjunto delas.

1.2 Outros Tópicos

1.2.1 Colisão

Para detectar a colisão foi implementado um código que verifica se há um interseção entre a área de dois objetos. A ação tomada quando ocorre a colisão entre o pacman e a parede, é impedir que o pacman continue na mesma direção, para isso quando é detectado que ouve interseção entre as áreas do pacman e a parede, o mesmo recua a uma distância insiguinificante na mesma direção em que estava.
A colisão com do pacman com o fantasma, as frutas, ocorre de maneira semelhante. Porém a ação tomada no momento da colisão é tratada de forma diferente.

1.2.2 Inteligência do fantasma

Para realizar o sistema de perseguição do pacman por parte do fantasma, foi utilizado um algoritmo que faz com que ele ande aleatoriamente e comece a perserguir o pacman quando o mesmo entra em seu raio de alcance, quando pacman entrar neste raio os fantasmas começam a persegui-lo tomando a direção e sentido entre o eixos x e y em linha reta de que tem a menor distancia do pacman. No ato da perseguição caso haja alguma colisão o movimento na direção onde houve a colisão é bloqueado para movimentar por um determinado periodo.

Autor

Douglas Venâncio — RA- 28614
Filipe Pena — RA- 30213
Marco Antonio — RA- 30749

Capítulo 2

Índice Hierárquico

2.1 Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

Fantasma 11
Fruta 21
Jogo 24
Menu 30
Objetos 39
    Parede 48
Pacman 40

Capítulo 3

Índice das Estruturas de Dados

3.1 Estruturas de Dados

Aqui estão as estruturas de dados, uniões e suas respectivas descrições:

Fantasma
A Classe Fantasma que modela e implementa as funcionalidade do fantasma. 11

Fruta
A Classe Fruta implementa a modelagem da fruta e suas funcionalidades 21

Jogo
A classe Jogo é a principal classe do jogo, ela gerencia todas as outras classes, alem de possuir o loop principal do jogo 24

Menu
A classe Menu implementa a tela de inicial de controle do jogo 30

Objetos
A Classe Objetos que possui funções que são usadas em algumas outras classes 39

Pacman
A Classe Pacman implementa a modelagem visual e física e as funcionalidades do pacman no jogo 40

Parede
A classe Parede implementa a modelagem e as funcionalisdades dos blocos que formam as paredes do mapa 48

Capítulo 4

Índice dos Arquivos

4.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

Fantasma.cpp
Implementação da classe Fantasma . . . 53
Fantasma.hpp . . . **??**
Fruta.cpp
Implementação da classe Fruta . . . 53
Fruta.hpp . . . **??**
jogo.cpp
Implementação da classe jogo . . . 54
jogo.hpp . . . **??**
main.cpp
Jogo baseado no clássico arcade PAC-MAN, implementado em c++ com as blibliotecas do Open-GL . . . 55
Menu.cpp
Implementação da classe Menu . . . 56
Menu.hpp . . . **??**
Objeto.cpp
Implementação da classe Objeto . . . **??**
Objeto.hpp . . . **??**
Pacman.cpp
Implementação da classe Pacman . . . 56
Pacman.hpp . . . **??**
parede.cpp
Implementação da classe parede . . . 57
parede.hpp . . . **??**
tga.cpp
Implementação da classe tga . . . 57

Capítulo 5

Estruturas

5.1 Referência da Classe Fantasma

A Classe Fantasma que modela e implementa as funcionalidade do fantasma.

`#include <Fantasma.hpp>`

Membros Públicos

- Fantasma (float, float, float, float, float)

  Construtor da classe fantasma.
- void vertex (double, double, double, float, float, float)

  Função que define os vértices e as normais em coordenadas esféricas.
- void vertex2 (double, double, double, float, float, float)

  Função que define os vértices e as normais invertidas em coordenadas esféricas.
- void move ()

  Função que gerencia o movimento do fantasma.
- void desenha (bool)

  Função que desenha o fantasma no jogo.
- void animacao ()

  Função que gerencia as variáveis que realizão a animação dos olhos e da "saia" do fantasma.
- void stop (int, int)

  Função que informa ao fantasma que ocorreu uma colisão, e ele precisa parar o movimento.
- int aleatorio (int)

  A função aleatório gera valores aleatórios que são usados nas tomadas de decisão do fantasma.
- int praonde (float, float, bool)

  Função que calcula a direção que o fantasma deve seguir, buscando seguir o pacman.
- void aumentanivel ()

  Aumenta a vaiavel que é somada no passo do fantasma, fazendo com que o mesmo aumete a velocidade.
- int colisao (float, float, float)

  Função que verifica se o pacman colidiu com o fantasma.
- void conttempo ()

  Conta o tempo que o fantasma fica impossibilitado de escolhar uma direção.

Campos de Dados

- float **tempo** =0
- float **x**
- float **y**
- float **z**
- float **altura**
- float **r**
- int **travdir** =0
- int **travesq** =0
- int **travcima** =0
- int **travatras** =0

5.1.1 Descrição detalhada

A Classe Fantasma que modela e implementa as funcionalidade do fantasma.

Definição na linha 9 do arquivo Fantasma.hpp.

5.1.2 Construtores e Destrutores

5.1.2.1 Fantasma()

```
Fantasma::Fantasma (
              float x1,
              float y1,
              float z1,
              float raio,
              float alt )
```

Construtor da classe fantasma.

Parâmetros

x1	Posição na coordenada x do plano do jogo.
y2	Posição na coordenada y do plano do jogo.
z3	Posição na coordenada z do plano do jogo.
raio	Raio que indica o tamanho do corpo do corpo do Objeto .
alt	Altura do corpo do Fantasma.

Definição na linha 11 do arquivo Fantasma.cpp.

```
12 {
13     x = x1;
14     y = y1;
15     z = z1;
16     r = raio;
17     altura = alt;
18 }
```

5.1.3 Funções membros

5.1.3.1 vertex()

```
void Fantasma::vertex (
             double th2,
             double ph2,
             double raio,
             float xi,
             float yi,
             float zi )
```

Função que define os vértices e as normais em coordenadas esféricas.

Parâmetros

th2	Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
ph2	Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
raio	Raio da esfera que será desenhada.
xi	Posição da esfera na coordenada x.
yi	Posição da esfera na coordenada y.
zi	Posição da esfera na coordenada z.

Definição na linha 20 do arquivo Fantasma.cpp.

```
21 {
22     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
23     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
24     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
25     glVertex3d(xd, yd, zd);
26     glNormal3f(xd, yd, zd);
27 }
```

5.1.3.2 vertex2()

```
void Fantasma::vertex2 (
             double th2,
             double ph2,
             double raio,
             float xi,
             float yi,
             float zi )
```

Função que define os vértices e as normais invertidas em coordenadas esféricas.

Parâmetros

th2	Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
ph2	Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
raio	Raio da esfera que será desenhada.
xi	Posição da esfera na coordenada x.
yi	Posição da esfera na coordenada y.
zi	Posição da esfera na coordenada z.

Definição na linha 29 do arquivo Fantasma.cpp.

```
30 {
31     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
32     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
33     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
34     glVertex3d(xd, yd, zd);
35     glNormal3f(-xd, -yd, -zd);
36 }
```

5.1.3.3 move()

```
void Fantasma::move ( )
```

Função que gerencia o movimento do fantasma.

Soma nas coordenadas x, y, z o tamanho do passo mais um valor que representa o aumento da velocidade.

Definição na linha 352 do arquivo Fantasma.cpp.

```
353 {
354
355     //move o fantasma para a baixo
356
357     if (rnd == 2)
358     {
359         z = z - (velocidade + velnivel);
360     }
361
362     //move o fantasma para a cima
363
364     else if (rnd == 1)
365     {
366         z = z + (velocidade + velnivel);
367     }
368
369     //move o fantasma para a esquerda
370
371     else if (rnd == 4)
372     {
373         x = x - (velocidade + velnivel);
374     }
375
376     //move o fantasma para a direita
377
378     else if (rnd == 3)
379     {
380         x = x + (velocidade + velnivel);
381     }
382 }
```

5.1.3.4 desenha()

```
void Fantasma::desenha (
             bool aux )
```

Função que desenha o fantasma no jogo.

Essa função usa de conceitos de coordanas cilindricas e esféricas para desenhar o corpo e a cabeça do fantasma respectivamente. A animação da "saia" do fantasma é feita usando uma função seno com parâmetro variável.

Definição na linha 50 do arquivo Fantasma.cpp.

```
51 {
52     double h = altura;
53
54     glColor3f(0.9, 0.9, 0.9);
55
```

```
56      //Rotaciona a imagem do fantas para direção que ele esta andando
57
58     glPushMatrix();
59     if (rnd == 1)
60     {
61         glTranslatef(x, y, z);
62         glRotatef(-90, 0, 1, 0);
63         glTranslatef(-x, -y, -z);
64         direcimag = 1;
65     }
66     else if (rnd == 2)
67     {
68         glTranslatef(x, y, z);
69         glRotatef(-270, 0, 1, 0);
70         glTranslatef(-x, -y, -z);
71         direcimag = 2;
72     }
73
74     else if (rnd == 4)
75     {
76         glTranslatef(x, y, z);
77         glRotatef(-180, 0, 1, 0);
78         glTranslatef(-x, -y, -z);
79         direcimag = 4;
80     }
81
82     else if (rnd == 3)
83     {
84         direcimag = 3;
85     }
86
87     else
88     {
89         if (direcimag == 2)
90         {
91             glTranslatef(x, y, z);
92             glRotatef(-90, 0, 1, 0);
93             glTranslatef(-x, -y, -z);
94         }
95
96         else if (direcimag == 1)
97         {
98             glTranslatef(x, y, z);
99             glRotatef(-270, 0, 1, 0);
100             glTranslatef(-x, -y, -z);
101         }
102
103         else if (direcimag == 4)
104         {
105             glTranslatef(x, y, z);
106             glRotatef(-180, 0, 1, 0);
107             glTranslatef(-x, -y, -z);
108         }
109     }
110
111
112      //movimenta a imagem do fantasma pelo mapa
113
114     glTranslatef(x, y, z);
115
116
117
118      //Desenho da base do fantasma desenhando o modulo do seno em volta do circulo da base que se
       movimenta
119
120     glCullFace(GL_FRONT);
121     glBegin(GL_TRIANGLE_STRIP);
122     if (aux)
123                 glColor3f(1.2, 0, 0);
124             else
125                 glColor3f(0.9, 0.9, 0.9);
126     for (int i = 0; i <= 360; i = i + 10)
127     {
128         glVertex3d((r * cos(i * M_PI / 180)), 0 , (r * sin(i * M_PI / 180)));
129         glNormal3d((r * cos(i * M_PI / 180)), 0 , (r * sin(i * M_PI / 180)));
130         glVertex3d((r * cos(i * M_PI / 180)) , -abs(( h / 3 * sin(3* i * M_PI / 180 + animabase))) , (r
       * sin(i * M_PI / 180)));
131         glNormal3d((r * cos(i * M_PI / 180)) , 0 , (r * sin(i * M_PI / 180)) );
132     }
133     glEnd();
134     glCullFace(GL_BACK);
135
136
137      //Desenho de um cilindro que é corpo do fantasma
138
139     for (double j = 0; j < h; j = j + (h / 10))
140     {
```

```
141         glBegin(GL_TRIANGLE_STRIP);
142         for (double i = 0; i <= 360; i = i + 10)
143         {
144             //corpo
145             if (aux)
146                 glColor3f(1.2, 0, 0);
147             else
148                 glColor3f(0.9, 0.9, 0.9);
149             glVertex3d((r * cos(i * M_PI / 180)) , j , (r * sin(i * M_PI / 180)) );
150             glNormal3d((r * cos(i * M_PI / 180)) , 0 , (r * sin(i * M_PI / 180)) );
151             glVertex3d((r * cos(i * M_PI / 180)) , j + h / 10 , (r * sin(i * M_PI / 180)) );
152             glNormal3d((r * cos(i * M_PI / 180)) , 0 , (r * sin(i * M_PI / 180)) );
153         }
154         glEnd();
155     }
156 
157 
158      //Animação dos olhos troca de cor
159 
160 
161     if (animaolhos < 13)
162     {
163         glColor3f(1, 0, 0);
164     }
165     else
166     {
167         glColor3f(0, 0, 1);
168     }
169 
170 
171      //desenho dos olhos do fantasma utilizando esferas
172 
173     double heye = (8 * h / 10);
174     double angeye = 330;
175 
176     for (double ph2 = -90; ph2 < 90; ph2 += 10)
177     {
178         glBegin(GL_TRIANGLE_STRIP);
179         for (double th2 = 0; th2 <= 360; th2 += 10)
180         {
181             vertex(th2, ph2, r / 4, (r * 0.9 * cos(angeye * M_PI / 180)) ,  heye, (r * 0.9 * sin(angeye
    * M_PI / 180)) );
182             vertex(th2, ph2 + 10, r / 4, (r * 0.9 * cos(angeye * M_PI / 180)) ,  heye, (r * 0.9 *
    sin(angeye * M_PI / 180)) );
183         }
184         glEnd();
185     }
186 
187     angeye = 30;
188 
189     for (double ph2 = -90; ph2 < 90; ph2 += 10)
190     {
191         glBegin(GL_TRIANGLE_STRIP);
192         for (double th2 = 0; th2 <= 360; th2 += 10)
193         {
194             vertex(th2, ph2, r / 4, (r * 0.9 * cos(angeye * M_PI / 180)) ,  heye, (r * 0.9 * sin(angeye
    * M_PI / 180)) );
195             vertex(th2, ph2 + 10, r / 4, (r * 0.9 * cos(angeye * M_PI / 180)) ,  heye, (r * 0.9 *
    sin(angeye * M_PI / 180)) );
196         }
197         glEnd();
198     }
199 
200 
201      //desenho da cabeça do fantasma utilizando metade de uma esfera
202 
203     glPushMatrix();
204     glTranslatef(0, h, 0);
205     if (aux)
206                 glColor3f(1.2, 0, 0);
207             else
208                 glColor3f(0.9, 0.9, 0.9);
209     for (double ph2 = -90; ph2 < 90; ph2 += 10)
210     {
211         glBegin(GL_TRIANGLE_STRIP);
212         for (double th2 = -90; th2 <= 90; th2 += 10)
213         {
214             vertex(th2, ph2, r, 0, 0 , 0);
215             vertex(th2, ph2 + 10, r, 0, 0 , 0);
216         }
217         glEnd();
218     }
219     glPopMatrix();
220 
221     //
222     glPopMatrix();
223 }
```

5.1.3.5 animacao()

```
void Fantasma::animacao ( )
```

Função que gerencia as variáveis que realizão a animação dos olhos e da "saia" do fantasma.

Definição na linha 38 do arquivo Fantasma.cpp.

```
39 {
40
41     //valor segundo o tempo para animar a base do fantasma
42
43     animabase = int(animabase + (180* M_PI / 180))%360;
44
45     //valor segundo o tempo para animar os olhos do fantasma
46
47     animaolhos = (animaolhos + 1) % 26;
48 }
```

5.1.3.6 stop()

```
void Fantasma::stop (
            int bateu,
            int aux )
```

Função que informa ao fantasma que ocorreu uma colisão, e ele precisa parar o movimento.

Na colisão o fantasma trava a direção para onde colidiu por um tempo, e define uma direção aleatoria para o próximo movimento.

Definição na linha 390 do arquivo Fantasma.cpp.

```
391 {
392
393     //caso o fantasma colida com uma parede acima dele faz com que ele retorne uma
394     //quantia insignificante para na direção contraria da colisão, ativa a trava de movimentação para
      cima e calcula
395     //uma direção de movimentação aleatoria
396
397     if (bateu == 1)
398     {
399         travcima = 1;
400         x = x - 0.1;
401         aleatorio(aux);
402     }
403
404
405     //caso o fantasma colida com uma parede abaixo dele faz com que ele retorne uma
406     //quantia insignificante para na direção contraria da colisão, ativa a trava de movimentação para
      baixo e calcula
407     //uma direção de movimentação aleatoria
408
409     if (bateu == 2)
410     {
411         travatras = 1;
412         x = x + 0.1;
413         aleatorio(aux);
414     }
415
416
417     //caso o fantasma colida com uma parede a direita dele faz com que ele retorne uma
418     //quantia insignificante para na direção contraria da colisão, ativa a trava de movimentação para
      direita e calcula
419     //uma direção de movimentação aleatoria
420
421     if (bateu == 3)
422     {
423         travdir = 1;
424         z = z - 0.1;
```

```
425         aleatorio(aux);
426     }
427
428
429     //caso o fantasma colida com uma parede esquerda dele faz com que ele retorne uma
430     //quantia insignificante para na direção contraria da colisão, ativa a trava de movimentação para
        esquerda e calcula
431     //uma direção de movimentação aleatoria
432
433     if (bateu == 4)
434     {
435         travesq = 1;
436         z = z + 0.1;
437         aleatorio(aux);
438     }
439 }
```

5.1.3.7 aleatorio()

```
int Fantasma::aleatorio (
            int a )
```

A função aleatório gera valores aleatórios que são usados nas tomadas de decisão do fantasma.

As possíveis possibilidades que o fantasma são armezenadas em um vetor e os números aleátorios são gerados com base nesse vetor.

Parâmetros

a	Ajuda na composição da semente que gera o numero aleatório.

Definição na linha 224 do arquivo Fantasma.cpp.

```
225 {
226
227      //trava de direção movimentação do fantasma qunado há uma colisão
228
229     std::vector<int> vec;
230     if (travdir == 0)
231         vec.push_back(1);
232     if (travesq == 0)
233         vec.push_back(2);
234     if (travcima == 0)
235         vec.push_back(3);
236     if (travatras == 0)
237         vec.push_back(4);
238
239
240      //calculo de direção aleatoria da movimentação do fantasma
241
242     if(vec.size()!=0)
243     {
244     srand(time(NULL) + a);
245     rnd = vec[(rand() % vec.size())];
246     }
247     vec.clear();
248 }
```

5.1.3.8 praonde()

```
int Fantasma::praonde (
            float px,
```

```
            float pz,
            bool vida )
```

Função que calcula a direção que o fantasma deve seguir, buscando seguir o pacman.

Para seguir o pacman o fantasma busca encurtar a maior distância, em x ou em z, da posição em que se encontra o pacman. A lógica de perceguissão se inverte se o pacman possuir a função especial, assim a distância é almentada. A perceguissão só acontece em determinada proximidade entre os dois personagens.

Parâmetros

px	Posição do pacman na coordenada x.
pz	Posição do pacman na coordenada z.
vida	Informa se o pacman possui vidas.

Definição na linha 251 do arquivo Fantasma.cpp.

```
252 {
253
254     //calculo da IA do fantasma para movimentação no eixo x caso o pacman esta em uma distancia de 15
    no eixo x do fantasma
255
256
257     if (abs(px - x) > abs(pz - z) && abs(px - x) < 15)
258     {
259
260         //caso o pacman não tenha vida o fantasma se movimenta no eixo x seguindo o pacman
261
262         if (!vida)
263         {
264             velocidade = 0.05;
265             if ((px - x) > 0)
266             {
267                 if(travcima==0)
268                 rnd = 3;
269             }
270             else
271             {
272                 if(travatras==0)
273                 rnd = 4;
274             }
275         }
276
277         //caso o pacman tenha vida o fantasma se movimenta no eixo x fugindo do pacman
278
279         else
280         {
281             velocidade = 0.08;
282             if ((px - x) > 0)
283             {
284                 if(travatras==0)
285                 {
286                 travcima=1;
287                 rnd = 4;
288                 }
289             }
290             else
291             {
292                 if(travcima==0)
293                 {
294                 travatras=1;
295                 rnd = 3;
296                 }
297             }
298         }
299     }
300
301     //calculo da IA do fantasma para movimentação no eixo y caso o pacman esta em uma distancia de 15
    no eixo y do fantasma
302
303     else if (!(abs(px - x) >= abs(pz - z)) && abs(pz - z) < 15)
304     {
305
306         //caso o pacman não tenha vida o fantasma se movimenta no eixo y seguindo o pacman
307
308         if (!vida)
309         {
310
```

```
311             velocidade = 0.05;
312             if ((pz - z) > 0)
313
314             {
315
316                 if(travdir==0)
317                 rnd = 1;
318             }
319
320             else
321             {
322                 if(travesq==0)
323                 rnd = 2;
324             }
325         }
326
327         //caso o pacman tenha vida o fantasma se movimenta no eixo x fugindo do pacman
328
329         else
330         {
331             velocidade = 0.08;
332             if ((pz - z) > 0)
333             {
334                 if(travesq==0)
335                 {
336                 travdir=1;
337                 rnd = 2;
338                 }
339             }
340             else
341             {
342                 if(travdir==0)
343                 {
344                 travesq=1;
345                 rnd = 1;
346                 }
347             }
348         }
349     }
350 }
```

5.1.3.9 aumentanivel()

```
void Fantasma::aumentanivel ( )
```

Aumenta a vaiavel que é somada no passo do fantasma, fazendo com que o mesmo aumete a velocidade.

Definição na linha 384 do arquivo Fantasma.cpp.

```
385 {
386     velnivel = velnivel + 0.05;
387 }
```

5.1.3.10 colisao()

```
int Fantasma::colisao (
             float px,
             float pz,
             float pt )
```

Função que verifica se o pacman colidiu com o fantasma.

Se as coordenadas do fantasma mais o seu raio, e as coordenadas do pacman mais o raio, se os valores coincidirem colidiu.

Parâmetros

px	Coordenada x do pacman.
pz	Coordenada z do pacman.
pt	raio do pacman.

Retorna

0 se não colidiu.
5 se colidiu.

Definição na linha 456 do arquivo Fantasma.cpp.

```
457 {
458     if ((px + pt) >= x && (px - pt) <= x && (pz + pt) >= z && (pz - pt) <= z)
459     {
460         return 5;
461     }
462     else
463         return 0;
464 }
```

5.1.3.11 conttempo()

```
void Fantasma::conttempo ( )
```

Conta o tempo que o fantasma fica impossibilitado de escolhar uma direção.

Definição na linha 441 do arquivo Fantasma.cpp.

```
442 {
443
444
445     tempo=tempo+0.01;
446     if (tempo > 300)
447     {
448         tempo = 0;
449         travdir = 0;
450         travesq = 0;
451         travcima = 0;
452         travatras = 0;
453     }
454 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- Fantasma.hpp
- Fantasma.cpp

5.2 Referência da Classe Fruta

A Classe Fruta implementa a modelagem da fruta e suas funcionalidades.

```
#include <Fruta.hpp>
```

Membros Públicos

- **Fruta** (float x, float y, float z, float r, bool)
- void vertex (double th2, double ph2, double raio, float xi, float yi, float zi)

  Função que define os vértices e as normais em coordenadas esféricas.
- void vertex2 (double th2, double ph2, double raio, float xi, float yi, float zi)

  Função que define os vértices e as normais inversas em coordenadas esféricas.
- void desenha ()

  Função responsável pela parte visual da fruta ou seja seu desenho.
- bool colisao (float, float, float)

  Função por verificar se houve a colisão de algum objeto com a fruta.

5.2.1 Descrição detalhada

A Classe Fruta implementa a modelagem da fruta e suas funcionalidades.

Definição na linha 6 do arquivo Fruta.hpp.

5.2.2 Funções membros

5.2.2.1 vertex()

```
void Fruta::vertex (
            double th2,
            double ph2,
            double raio,
            float xi,
            float yi,
            float zi )
```

Função que define os vértices e as normais em coordenadas esféricas.

Parâmetros

th2	Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
ph2	Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
raio	Raio da esfera que será desenhada.
xi	Posição da esfera na coordenada x.
yi	Posição da esfera na coordenada y.
zi	Posição da esfera na coordenada z.

Definição na linha 19 do arquivo Fruta.cpp.

```
20 {
21     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
22     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
23     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
24     glVertex3d(xd, yd, zd);
25     glNormal3f(xd, yd, zd);
```

```
26 }
```

5.2.2.2 vertex2()

```
void Fruta::vertex2 (
             double th2,
             double ph2,
             double raio,
             float xi,
             float yi,
             float zi )
```

Função que define os vértices e as normais inversas em coordenadas esféricas.

Parâmetros

th2	Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
ph2	Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
raio	Raio da esfera que será desenhada.
xi	Posição da esfera na coordenada x.
yi	Posição da esfera na coordenada y.
zi	Posição da esfera na coordenada z.

Definição na linha 28 do arquivo Fruta.cpp.

```
29 {
30     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
31     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
32     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
33     glVertex3d(xd, yd, zd);
34     glNormal3f(-xd, -yd, -zd);
35 }
```

5.2.2.3 colisao()

```
bool Fruta::colisao (
             float px,
             float pz,
             float pt )
```

Função por verificar se houve a colisão de algum objeto com a fruta.

Parâmetros

px	Posição x do objeto
pz	Posição z do objeto
pt	Posição distancia da aresta da hitbox do centro do objeto

Retorna

true se houve a colisão
false se não houve a colisão

Definição na linha 73 do arquivo Fruta.cpp.

```
74 {
75     //Detecta a colisao com a fruta considerando seu centro na cena
76     if ((px + pt) >= x && (px - pt) <= x && (pz + pt) >= z && (pz - pt) <= z)
77         return true;
78     else
79         return false;
80 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- Fruta.hpp
- Fruta.cpp

5.3 Referência da Classe Jogo

A classe Jogo é a principal classe do jogo, ela gerencia todas as outras classes, alem de possuir o loop principal do jogo.

`#include <jogo.hpp>`

Membros Públicos

- Jogo (int argc, char **argv)
  Construtor da classe Jogo.
- void desenha ()
  Função que chama todas as outras funções que desenham o objeto.
- void atualizatamanho (GLsizei w, GLsizei h)
  Atualiza o tamanho da janela do jogo.
- void KeyboardHandle (unsigned char key, int x, int y)
  Pega a ação do jogador ao precisionar uma tecla comum.
- void **MouseHandle** (int button, int state, int x, int y)
- void SpecialKeyHandle (int key, int x, int y)
  Pega a ação do jogador ao precisionar uma tecla especial.
- void atualiza (int value, void(*func_ptr)(int))
  Função que gerencia as ações que ocorrem durente o loop do jogo..
- void newgame ()
  Função que cria um novo jogo.
- void gameover ()
  Função que destroi os elementos do jogo.

5.3.1 Descrição detalhada

A classe Jogo é a principal classe do jogo, ela gerencia todas as outras classes, alem de possuir o loop principal do jogo.

Definição na linha 25 do arquivo jogo.hpp.

5.3.2 Construtores e Destrutores

5.3.2.1 Jogo()

```
Jogo::Jogo (
            int argc,
            char ** argv )
```

Construtor da classe Jogo.

Cria a janela do jogor, e chama uma função que cria um novo jogo.

Definição na linha 12 do arquivo jogo.cpp.

```
13 {
14     newgame();
15     glutInit(&argc, argv);
16     glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
17     glutInitWindowSize(900, 800);
18     glutCreateWindow("PACMAN GAME
);
19     Inicializa();
20 }
```

5.3.3 Funções membros

5.3.3.1 desenha()

```
void Jogo::desenha ( )
```

Função que chama todas as outras funções que desenham o objeto.

Esta função ainda gerencia o momento em que cada elemento do jogo deve ser desenhado. Desenha o piso do ambiente do jogo.

Definição na linha 128 do arquivo jogo.cpp.

```
129 {
130
131     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
132
133     if (menu->getstart() && !menu->getpause())
134     {
135
136         //chao
137         glEnable(GL_TEXTURE_2D);
138         glBindTexture(GL_TEXTURE_2D, 13);
139         glBegin(GL_POLYGON);
140         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
141         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
142         glColor3f(0.9, 0.9, 0.9);
143         glTexCoord2f(0.0f, 0.0f);
144         glVertex3d(38, -0.01, 38);
145         glTexCoord2f(10.0f, 0.0f);
146         glVertex3d(38, -0.01, -38);
147         glTexCoord2f(10.0f, 10.0f);
148         glVertex3d(-38, -0.01, -38);
149         glTexCoord2f(0.0f, 10.0f);
150         glVertex3d(-38, -0.01, 38);
151
152         glNormal3d(0, 1, 0);
153         glEnd();
```

```
154
155         glDisable(GL_TEXTURE_2D);
156
157         pacman->desenha();
158         for (int index = 0; index < valedassombras_.size(); ++index)
159             valedassombras_[index]->desenha(pacman->temvida());
160         for (int index = 0; index < rango_.size(); ++index)
161             rango_[index]->desenha();
162         for (int index = 0; index < especial_.size(); ++index)
163             especial_[index]->desenha();
164         for (list<Objetos *>::const_iterator it = mapa_.begin(); it != mapa_.end(); it++)
165         {
166             (*it)->desenha();
167         }
168     }
169     else
170     {
171         menu->setlife(pacman->vida());
172         menu->setpontos(pacman->pontos());
173         menu->desenha();
174     }
175     glutPostRedisplay();
176     glutSwapBuffers();
177 }
```

5.3.3.2 atualizatamanho()

```
void Jogo::atualizatamanho (
            GLsizei w,
            GLsizei h )
```

Atualiza o tamanho da janela do jogo.

Parâmetros

w	Largura da janela após a mesma ser expandida.
h	Altura da janela após a mesma ser expandida

Definição na linha 180 do arquivo jogo.cpp.

```
181 {
182     // Especifica as dimensões da Viewport
183     if (h == 0)
184         h = 1;
185     glViewport(0, 0, w, h);
186     fAspect = (GLfloat)w / (GLfloat)h;
187
188     glMatrixMode(GL_PROJECTION);
189     glLoadIdentity();
190     gluPerspective(angle, fAspect, 0.5, 500.0);
191     glMatrixMode(GL_MODELVIEW);
192     glLoadIdentity();
193     gluLookAt(40, 90, 0, 0, 0, 0, 0, 1, 0);
194     glutPostRedisplay();
195 }
```

5.3.3.3 KeyboardHandle()

```
void Jogo::KeyboardHandle (
            unsigned char key,
            int x,
            int y )
```

Pega a ação do jogador ao precisionar uma tecla comum.

Parâmetros

key	Valor da tecla precionada.

Definição na linha 198 do arquivo jogo.cpp.

```
199 {
200     //tecla enter menu
201     menu->entermenu(key);
202     //glutPostRedisplay();
203 }
```

5.3.3.4 SpecialKeyHandle()

```
void Jogo::SpecialKeyHandle (
            int key,
            int x,
            int y )
```

Pega a ação do jogador ao precisionar uma tecla especial.

Parâmetros

key	Valor da tecla precionada.
x	
y	

Definição na linha 220 do arquivo jogo.cpp.

```
221 {
222     //controle pacman
223     if (!menu->getpause())
224         pacman->direcao(key);
225
226     //teclas do menu
227     menu->setasmenu(key);
228     menu->pausa(key);
229
230     //glutPostRedisplay();
231 }
```

5.3.3.5 atualiza()

```
void Jogo::atualiza (
            int value,
            void(*)(int) func_ptr )
```

Função que gerencia as ações que ocorrem durente o loop do jogo..

Nessa função são realisados os teste de colisão em cada loop, para todos os componentes do jogo. Também gerencia o estado do jogo, como gameover, novo jogo, próxima fase. Controla o tempo de tudo que ocorre no jogo.

Parâmetros

value	Tempo em que ocorre as atualisações.
func_ptr	Ponteiro que referencia a própria função.

Definição na linha 233 do arquivo jogo.cpp.

```
234 {
235     if (!menu->getpause() && menu->getstart())
236     {
237         if (tempo != -1)
238         {
239             tempo++;
240         }
241         if (tempo == 500 || pacman->temvida() == false)
242         {
243             tempo = -1;
244             pacman->zeravida();
245         }
246         int aleat = 0, a = 0;
247         int bateu, bateu1, bateu2;
248         for (int index = 0; index < rango_.size(); ++index)
249         {
250             if (rango_[index]->colisao(pacman->getx(), pacman->getz(), pacman->       getr()))
251             {
252                 pacman->comeu(false);
253                 rango_[index] = rango_[rango_.size() - 1];
254                 rango_.pop_back();
255             }
256         }
257         for (int index = 0; index < especial_.size(); ++index)
258         {
259             if (especial_[index]->colisao(pacman->getx(), pacman->getz(), pacman->       getr()))
260             {
261                 especial_[index] = especial_[especial_.size() - 1];
262                 especial_.pop_back();
263                 pacman->comeu(true);
264                 tempo = 0;
265             }
266         }
267         for (list<Objetos *>::const_iterator it = mapa_.begin(); it != mapa_.end(); it++)
268         {
269             bateu = (*it)->colisao(pacman->getx(), pacman->getz(), pacman->       getr());
270             pacman->stop(bateu);
271         }
272         pacman->move();
273         for (int index = 0; index < valedassombras_.size(); ++index)
274         {
275             for (list<Objetos *>::const_iterator it = mapa_.begin(); it != mapa_.end(); it++)
276             {
277                 valedassombras_[index]->conttempo();
278                 bateu2 = (*it)->colisao(valedassombras_[index]->x, valedassombras_[index]->z,
    valedassombras_[index]->r);
279                 valedassombras_[index]->stop(bateu2, aleat++);
280             }
281             bateu1 = valedassombras_[index]->colisao(pacman->getx(), pacman->
        getz(), pacman->getr());
282             pacman->stop(bateu1);
283 
284             if (bateu1 == 5 && pacman->vida() > 0)
285             {
286                 nfantasma-;
287                 pacman->tiravida();
288                 valedassombras_[index] = valedassombras_[valedassombras_.size() - 1];
289                 valedassombras_.pop_back();
290             }
291 
292             //game over
293             else if (bateu1 == 5 && pacman->vida() < 1)
294             {
295                 int pont = pacman->pontos();
296                 gameover();
297                 newgame();
298                 menu->setptsgameover(pont);
299                 break;
300             }
301             valedassombras_[index]->move();
302             valedassombras_[index]->praonde(pacman->getx(), pacman->getz(), pacman->       temvida());
303             valedassombras_[index]->animacao();
304         }
305 
306         //passou de fase
307         if (pacman->pontos() == maxpontos && nivel < lastnivel)
308         {
```

```
309             nivel++;
310             int pont = pacman->pontos();
311             gameover();
312             newgame();
313             pacman->setpontos(pont);
314             for (int index = 0; index < valedassombras_.size(); ++index)
315                 valedassombras_[index]->aumentanivel();
316             menu->setstart(true);
317         }
318 
319         //venceu
320         else if (pacman->pontos() == maxpontos && nivel == lastnivel)
321         {
322             int pont = pacman->pontos();
323             gameover();
324             newgame();
325             menu->setptsgameover(pont);
326         }
327     }
328     else
329     {
330     }
331     glutPostRedisplay();
332     glutTimerFunc(20, func_ptr, time);
333 }
```

5.3.3.6 newgame()

```
void Jogo::newgame ( )
```

Função que cria um novo jogo.

Nessa função é criada uma matriz que contem informações para desenhar os elementos do jogo. Os valores da para matriz são carregadas de um arquivo, e para cada nivel do jogo existe um arquivo diferente. Chama a função que cria os objetos do jogo.

Definição na linha 22 do arquivo jogo.cpp.

```
23 {
24     for (int i = 0; i < 20; i++)
25     {
26         vector<int> row;
27         for (int j = 0; j < 20; j++)
28         {
29             row.push_back(i * j);
30         }
31         mat_map.push_back(row);
32     }
33 
34     if (nivel == 1)
35         map_file.open("../src/maps/mapa2.map
);
36     if (map_file.is_open())
37     {
38         int aux;
39         int counti = 0;
40         int countj = 0;
41         while (map_file » aux)
42         {
43             mat_map[counti][countj] = aux;
44             countj++;
45             if (countj == 20)
46             {
47                 counti++;
48                 countj = 0;
49             }
50         }
51         map_file.close();
52     }
53 
54     //nivel2
55     if (nivel == 2)
56         map_file.open("../src/maps/mapa3.map
);
57     if (map_file.is_open())
58     {
59         int aux;
```

```
60          int counti = 0;
61          int countj = 0;
62          while (map_file » aux)
63          {
64              mat_map[counti][countj] = aux;
65              countj++;
66              if (countj == 20)
67              {
68                  counti++;
69                  countj = 0;
70              }
71          }
72          map_file.close();
73      }
74
75      pacman = new Pacman(0, 1.5, 0, 1.5);
76      menu = new Menu(20, 30);
77      insere_objetos();
78  }
```

5.3.3.7 gameover()

```
void Jogo::gameover ( )
```

Função que destroi os elementos do jogo.

Definição na linha 335 do arquivo jogo.cpp.

```
336 {
337     delete pacman;
338     especial_.clear();
339     rango_.clear();
340     mat_map.clear();
341     mapa_.clear();
342     valedassombras_.clear();
343     delete menu;
344 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- jogo.hpp
- jogo.cpp

5.4 Referência da Classe Menu

A classe Menu implementa a tela de inicial de controle do jogo.

```
#include <Menu.hpp>
```

Membros Públicos

- Menu (int, int)

  Construtor da classe menu.
- void setasmenu (int)

  Função que gerencia e realiza a comunicação entre o teclado e a tela de menu.
- bool getpause ()

  Retorna o parâmetro pause para comunicação com o jogo.
- bool getstart ()

  Retorna o parâmetro start para comunicação com o jogo.

- void setstart (bool)

  Avisa para o menu que o jogo foi estartado e desta forma desabilita o menu e abilita o jogo.
- void pausa (int)

  Realiza a comunicação do jogador com a tela de menu, na ação de pausar o jogo.
- void entermenu (unsigned char)

  Realiza a comunicação do jogador com a tela de menu, na ação confirmar a opção selecionada no menu.
- void setpontos (int)

  Atribui a variavel pts a pontuação do usuário no jodo, se o mesmo estiver iniciado.
- void setptsgameover (int)

  Atribui a variavel pts a pontuação do usuário no jodo, quando o jodador perde.
- void setlife (int)

  atribui a variavel life a quantidade de vidas do usuário.
- void desenha ()

  Função que desenha o menu na tela.
- void texto (char ∗, float, float, float, int)

  Função que responsavel pela parte visual dos textos.
- void numero (int, float, float, float)

  Função que responsavel pela parte visual dos números.

5.4.1 Descrição detalhada

A classe Menu implementa a tela de inicial de controle do jogo.

Além da tela inicial a classe gerencia o menu de pause. A pontuação do jogo é apresentada ao jogador pela tela de menu.

Definição na linha 11 do arquivo Menu.hpp.

5.4.2 Construtores e Destrutores

5.4.2.1 Menu()

```
Menu::Menu (
              int tx,
              int tz )
```

Construtor da classe menu.

Parâmetros

tx	define o tamanho da tela na direção x
tz	define o tamanho da tela na direção z

Definição na linha 5 do arquivo Menu.cpp.

```
6 {
7     this->tx = tx;
```

```
8     this->tz = tz;
9     pts=0;
10     life=0;
11 }
```

5.4.3 Funções membros

5.4.3.1 setasmenu()

```
void Menu::setasmenu (
              int key )
```

Função que gerencia e realiza a comunicação entre o teclado e a tela de menu.

Parâmetros

key	Inteiro que parametriza a tecla precionada.

Definição na linha 47 do arquivo Menu.cpp.

```
48 {
49     //habilita as setes de navegação do menu caso o jogo não tenha iniciado ou esteja pausado
50     if(getpause()==true||getstart()==false)
51     {
52
53     //muda o botão do jogo selecionado
54     if(key==GLUT_KEY_UP)
55     {
56         opseta-;
57         if(start==false)
58         {
59             if(opseta==-1)
60             {
61             opseta=2;
62             }
63             if(opseta==1)
64             {
65             opseta=0;
66             }
67         }
68         else
69         {
70             if(opseta==0)
71             {
72             opseta=2;
73             }
74         }
75     }
76
77     //muda o botão do jogo selecionado
78     if(key==GLUT_KEY_DOWN)
79     {
80         opseta++;
81         if(start==false)
82         {
83             opseta=opseta%3;
84             if(opseta==1)
85             {
86             opseta=2;
87             }
88         }
89         else
90         {
91             if(opseta==0||opseta==3)
92             opseta=1;
93         }
94     }
95
96     }
97 }
```

5.4.3.2 getpause()

```
bool Menu::getpause ( )
```

Retorna o parâmetro pause para comunicação com o jogo.

Retorna

true Se o jogo foi pausado.
false Se o jogo não esta pausado.

Definição na linha 145 do arquivo Menu.cpp.

```
146 {
147     return pause;
148 }
```

5.4.3.3 getstart()

```
bool Menu::getstart ( )
```

Retorna o parâmetro start para comunicação com o jogo.

Retorna

true Se o jogo foi iniciado.
false Se o jogo não esta iniciado.

Definição na linha 150 do arquivo Menu.cpp.

```
151 {
152     return start;
153 }
```

5.4.3.4 setstart()

```
void Menu::setstart (
            bool starta )
```

Avisa para o menu que o jogo foi estartado e desta forma desabilita o menu e abilita o jogo.

Parâmetros

starta	valor passado pela lógica do jogo, quendo o jogador vai passar de fase.

Definição na linha 155 do arquivo Menu.cpp.

```
156 {
```

```
157     start=starta;
158 }
```

5.4.3.5 pausa()

```
void Menu::pausa (
            int key )
```

Realiza a comunicação do jogador com a tela de menu, na ação de pausar o jogo.

Parâmetros

key	Passa a informação de que a tecla foi precionada.

Definição na linha 126 do arquivo Menu.cpp.

```
127 {
128     //caso o jogo esteja startado e não estja pausado habilida a tecla de pause(HOME)
129     if(getstart()==true&&getpause()==false)
130     {
131     //verifica se a tecla home foi pressionada para pausar o jogo
132      if(key==GLUT_KEY_HOME)
133     {
134         pause=!pause;
135         if(pause)
136         {
137             opseta=1;
138         }
139     }
140
141     }
142
143 }
```

5.4.3.6 entermenu()

```
void Menu::entermenu (
            unsigned char key )
```

Realiza a comunicação do jogador com a tela de menu, na ação confirmar a opção selecionada no menu.

Parâmetros

key	Passa a informação de que a telca enter foi precionada.

Definição na linha 100 do arquivo Menu.cpp.

```
101 {
102     //habilita a tecla de seleção de opção do menu(enter) se o jogo estiver em pause ou caso ele não
      tenha sido startado
103     if(getpause()==true||getstart()==false)
104     {
105
106     //caso esteja selecionado o botão iniciar e seja apertado a tecla enter inicia o jogo
107     if(key==(char)13&&opseta==0)
108     {
109         start=true;
110     }
111     //caso esteja selecionado o botão continuar e seja apertado a tecla enter continua o jogo
112     if(key==(char)13&&opseta==1)
```

```
113     {
114         pause=false;
115     }
116     //caso esteja selecionado o botão sair e seja apertado a tecla enter sai do jogo
117     if(key==(char)13&&opseta==2)
118     {
119         exit(0);
120     }
121 
122     }
123 }
```

5.4.3.7 setpontos()

```
void Menu::setpontos (
            int pontos )
```

Atribui a variavel pts a pontuação do usuário no jodo, se o mesmo estiver iniciado.

Parâmetros

pontos	Passa o valor de pontos somado pelo jogador.

Definição na linha 160 do arquivo Menu.cpp.

```
161 {
162     if(start)
163     {
164     pts=pontos;
165     }
166 }
```

5.4.3.8 setptsgameover()

```
void Menu::setptsgameover (
            int pontos )
```

Atribui a variavel pts a pontuação do usuário no jodo, quando o jodador perde.

Parâmetros

pontos	Passa o valor de pontos somado pelo jogador antes de perder.

Definição na linha 168 do arquivo Menu.cpp.

```
169 {
170     pts=pontos;
171 }
```

5.4.3.9 setlife()

```
void Menu::setlife (
            int vida )
```

atribui a variavel life a quantidade de vidas do usuário.

Parâmetros

vida	passa o número de vidas que o jogador possui no jogo.

Definição na linha 173 do arquivo Menu.cpp.

```
174 {
175     life=vida;
176 }
```

5.4.3.10 desenha()

```
void Menu::desenha ( )
```

Função que desenha o menu na tela.

Definição na linha 178 do arquivo Menu.cpp.

```
179 {
180     // configura a camera do menu
181     glLoadIdentity();
182     gluLookAt(0, 60, 0, 0, 0, 0, 1, 0, 0);
183 
184     // faz o plano de fundo do menu
185     glColor3f(0.3, .3, .3);
186     glBegin(GL_QUADS);
187     glNormal3d(0, 1, 0);
188     glVertex3d(-tx, 0, tz);
189     glVertex3d(tx, 0, tz);
190     glVertex3d(tx, 0, -tz);
191     glVertex3d(-tx, 0, -tz);
192     glEnd();
193 
194     //botão opção inciar
195     //caso selecionado colore de verde
196     if(opseta==0)
197     glColor3f(0, 1, 0);
198     //nao selecionado colore de vermelho
199     else
200     glColor3f(1, 0, 0);
201     //caso o jogo esteja em estado de pause colore de cinza desabilitando o botao
202     if(pause==true)
203     glColor3f(0.2, .2, .2);
204     //desenha o botão
205     glBegin(GL_QUADS);
206     glNormal3d(0, 1, 0);
207     glVertex3d(-lb / 2, 0.1, 8);
208     glVertex3d(lb / 2, 0.1, 8);
209     glVertex3d(lb / 2, 0.1, -8);
210     glVertex3d(-lb / 2, 0.1, -8);
211     glEnd();
212 
213     //botão opção continuar
214     //caso selecionado colore de verde
215     if(opseta==1)
216     glColor3f(0, 1, 0);
217     //nao selecionado colore de vermelho
218     else
219     glColor3f(1, 0, 0);
220     //caso o jogo esteja em estado de nao foi startado colore de cinza desabilitando o botao
221     if(start==false)
222     glColor3f(0.2, .2, .2);
223     //desenha o botão
224     glBegin(GL_QUADS);
225     glNormal3d(0, 1, 0);
226     glVertex3d(-lb / 2 - 6, 0.1, 8);
227     glVertex3d(lb / 2 - 6, 0.1, 8);
228     glVertex3d(lb / 2 - 6, 0.1, -8);
229     glVertex3d(-lb / 2 - 6, 0.1, -8);
230     glEnd();
231 
232     //botão opção sair
233     //caso selecionado colore de verde
```

```
234     if(opseta==2)
235     glColor3f(0, 1, 0);
236     //nao selecionado colore de vermelho
237     else
238     glColor3f(1, 0, 0);
239     //desenha o botão
240     glBegin(GL_QUADS);
241     glNormal3d(0, 1, 0);
242     glVertex3d(-lb / 2 - 12, 0.1, 8);
243     glVertex3d(lb / 2 - 12, 0.1, 8);
244     glVertex3d(lb / 2 - 12, 0.1, -8);
245     glVertex3d(-lb / 2 - 12, 0.1, -8);
246     glEnd();
247
248     //Desenha a caixa de texto dos Pontos
249     glColor3f(1.0, 0, 0);
250     glBegin(GL_QUADS);
251     glNormal3d(0, 1, 0);
252     glVertex3d(-lb / 2 - 16, 0.1, 6 + 22);
253     glVertex3d(lb / 2 - 16, 0.1, 6 + 22);
254     glVertex3d(lb / 2 - 16, 0.1, -6 + 22);
255     glVertex3d(-lb / 2 - 16, 0.1, -6 + 22);
256     glEnd();
257
258     //Desenha a caixa de texto das vidas
259     glColor3f(1.0, 0, 0);
260     glBegin(GL_QUADS);
261     glNormal3d(0, 1, 0);
262     glVertex3d(-lb / 2 - 10, 0.1, 6 + 22);
263     glVertex3d(lb / 2 - 10, 0.1, 6 + 22);
264     glVertex3d(lb / 2 - 10, 0.1, -6 + 22);
265     glVertex3d(-lb / 2 - 10, 0.1, -6 + 22);
266     glEnd();
267
268     //coloca os textos em seus respectivos botoes e caixas de texto
269     glColor3f(255, 255, 0);
270     texto("Pontos:
, -10.5, 0.2, 16, 1);
271     glColor3f(0, 0, 1);
272     texto("Vidas:
, -16.5, 0.2, 16, 1);
273     texto("Iniciar
, -1, 0.2, -3, 1);
274     texto("Continuar
, -7, 0.2, -5, 1);
275     texto("Sair
, -13, 0.2, -2, 1);
276
277     glColor3f(255, 255, 0);
278     texto("PAC-MAN
, 10, 0.2, -5, 2);
279
280     glColor3f(0, 0, 1);
281     numero(pts, -10.5, 0.2, 22);
282     glColor3f(0, 0, 1);
283     numero(life, -16.5, 0.2, 22);
284 }
```

5.4.3.11 texto()

```
void Menu::texto (
              char * str,
              float x,
              float y,
              float z,
              int t )
```

Função que responsavel pela parte visual dos textos.

Parâmetros

str	caracteres que armazenam o texto a ser exibido na tela
x	Posição na coordenada x da tela

Parâmetros

y	Posição na coordenada y da tela
z	Posição na coordenada z da tela
t	variavel que informa o tamanho do texto a ser desenhado.

Definição na linha 15 do arquivo Menu.cpp.

```
16 {
17     //escreve o texto de str em determinada coordenada do menu
18     glRasterPos3f(x, y, z);
19     if (t == 1)
20     {
21         for (int i = 0; i < strlen(str); i++)
22         {
23             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str[i]);
24         }
25     }
26     else if (t == 2)
27     {
28         for (int i = 0; i < strlen(str); i++)
29         {
30             glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str[i]);
31         }
32     }
33 }
```

5.4.3.12 numero()

```
void Menu::numero (
              int n,
              float x,
              float y,
              float z )
```

Função que responsavel pela parte visual dos números.

Parâmetros

str	caracteres que armazenam o texto a ser exibido na tela
x	Posição na coordenada x da tela
y	Posição na coordenada y da tela
z	Posição na coordenada z da tela

Definição na linha 34 do arquivo Menu.cpp.

```
35 {
36     //escreve o valor de n em determinada coordenada do menu
37     char str[20];
38     sprintf(str, "%d
, n);
39     glRasterPos3f(x, y, z);
40     for (int i = 0; i < strlen(str); i++)
41     {
42         glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str[i]);
43     }
44 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

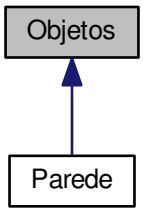
- Menu.hpp
- Menu.cpp

5.5 Referência da Classe Objetos

A Classe Objetos que possui funções que são usadas em algumas outras classes.

`#include <Objeto.hpp>`

Diagrama de hierarquia para Objetos:



Membros Públicos

- virtual void desenha ()=0

  Função que desenha um objeto no ambiente do jogo.
- virtual void **atualiza** (int value)=0
- virtual int colisao (float x, float z, float t)=0

  Função que implementa a colisão do objeto com um outro.

5.5.1 Descrição detalhada

A Classe Objetos que possui funções que são usadas em algumas outras classes.

Definição na linha 25 do arquivo Objeto.hpp.

5.5.2 Funções membros

5.5.2.1 desenha()

```
virtual void Objetos::desenha ( )  [pure virtual]
```

Função que desenha um objeto no ambiente do jogo.

Implementado por Parede.

5.5.2.2 colisao()

```
virtual int Objetos::colisao (
            float x,
            float z,
            float t )  [pure virtual]
```

Função que implementa a colisão do objeto com um outro.

Parâmetros

x	Posição em x do objeto comparado.
z	Posição em z do objeto comparado.
t	Raio do objeto comparado.

Retorna

value que contém a informação sobre a colisão.

Implementado por Parede.

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- Objeto.hpp

5.6 Referência da Classe Pacman

A Classe Pacman implementa a modelagem visual e física e as funcionalidades do pacman no jogo.

`#include <Pacman.hpp>`

Membros Públicos

- Pacman (float x, float y, float z, float r)
  Construtor do pacman.
- ~Pacman ()
  Destrutor do pacman.
- void vertex (double th2, double ph2, double raio, float xi, float yi, float zi)
  Função que define os vértices e as normais em coordenadas esféricas.
- void vertex2 (double th2, double ph2, double raio, float xi, float yi, float zi)
  Função que define os vértices e as normais inversas em coordenadas esféricas.
- void move ()
  Função responsável por fazer a movimentação do pacman mudando as coordenadas da sua posição.
- void direcao (int key)
  Função responsável por fazer a comunicação das teclas de movimentação com o pacman;.
- void desenha ()
  Função responsável pela parte visual do pacman ou seja seu desenho.
- void stop (int bateu)
  Função responsável por tratar a colisão no pacman quando ela ocorre.
- void comeu (bool)
  Verifica se o pacman comeu vida ou comida.
- void zeravida ()
  Zera o numero de vidas do pacman.
- float getx ()
  Retorna o valor da posicao onde o pacman esta em x.
- float gety ()
  Retorna o valor da posicao onde o pacman esta em y.

- float getz ()

  Retorna o valor da posicao onde o pacman esta em z.
- float getr ()

  Retorna o valor do raio do pacman.
- float getangulo ()

  Retorna o angulo para o qual a frente do pacman esta direcionado.
- int vida ()

  Retorna o numero de vidas que o pacman tem.
- bool temvida ()

  Retorna se o pacman tem alguma vida.
- void tiravida ()

  Tira uma vida do pacman.
- int pontos ()

  Retorna se o numero de comidas que o pacman comeu ou seja o numero de pontos do score.
- void setpontos (int)

  Muda o valor de pontos ou comidas que o pacman comeu.

5.6.1 Descrição detalhada

A Classe Pacman implementa a modelagem visual e física e as funcionalidades do pacman no jogo.

Definição na linha 5 do arquivo Pacman.hpp.

5.6.2 Construtores e Destrutores

5.6.2.1 Pacman()

```
Pacman::Pacman (
            float x,
            float y,
            float z,
            float r )
```

Construtor do pacman.

Parâmetros

x	posição do pacman em x
y	posição do pacman em y
z	posição do pacman em z
r	raio do pacman

Definição na linha 10 do arquivo Pacman.cpp.

```
11 {
12     x = x1;
13     y = y1;
14     z = z1;
```

```
15     r = raio;
16 }
```

5.6.3 Funções membros

5.6.3.1 vertex()

```
void Pacman::vertex (
             double th2,
             double ph2,
             double raio,
             float xi,
             float yi,
             float zi )
```

Função que define os vértices e as normais em coordenadas esféricas.

Parâmetros

th2	Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
ph2	Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
raio	Raio da esfera que será desenhada.
xi	Posição da esfera na coordenada x.
yi	Posição da esfera na coordenada y.
zi	Posição da esfera na coordenada z.

Definição na linha 18 do arquivo Pacman.cpp.

```
19 {
20     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
21     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
22     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
23     glVertex3d(xd, yd, zd);
24     glNormal3f(xd, yd, zd);
25 }
```

5.6.3.2 vertex2()

```
void Pacman::vertex2 (
             double th2,
             double ph2,
             double raio,
             float xi,
             float yi,
             float zi )
```

Função que define os vértices e as normais inversas em coordenadas esféricas.

```
@param th2 Ângulo Theta calculado para definir os vértices com base em coordenadas esféricas.
@param ph2 Ângulo Phi calculado para definir os vértices com base em coordenadas esféricas.
@param raio Raio da esfera que será desenhada.
@param xi Posição da esfera na coordenada x.
@param yi Posição da esfera na coordenada y.
@param zi Posição da esfera na coordenada z.
```

Definição na linha 27 do arquivo Pacman.cpp.

```
28 {
29     double xd = (raio * sin(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + xi;
30     double yd = (raio * cos(th2 * M_PI / 180) * cos(ph2 * M_PI / 180)) + yi;
31     double zd = (raio * sin(ph2 * M_PI / 180)) + zi;
32     glVertex3d(xd, yd, zd);
33     glNormal3f(-xd, -yd, -zd);
34 }
```

5.6.3.3 direcao()

```
void Pacman::direcao (
            int key )
```

Função responsável por fazer a comunicação das teclas de movimentação com o pacman;.

Parâmetros

key	Passa informação da tecla precionada pelo usuario

Definição na linha 213 do arquivo Pacman.cpp.

```
214 {
215
216      //adiciona 10 no angulo da direção em que o pacman ira andar caso seja pressionada a tecla seta da
     direita
217
218     if(key==GLUT_KEY_RIGHT)
219     {
220         angdir= angdir+10;
221         if(angdir>=360)
222         {
223             angdir=0;
224         }
225
226     }
227
228      //subtrai 10 no angulo da direção em que o pacman ira andar caso seja pressionada a tecla seta da
     esquerda
229
230     if(key==GLUT_KEY_LEFT)
231     {
232         angdir= angdir-10;
233         if(angdir<=0)
234         {
235             angdir=360;
236         }
237     }
238
239      //caso a tecla seta para cima seja pressionada a varivel direc para 4 que faz com que o
240      //pacman se movimente no sentido positivo em que sua direção esta posicionada
241
242     if(key==GLUT_KEY_UP)
243     {
244         direc = 4;
245     }
246
247      //caso a tecla seta para baixo seja pressionada a varivel direc para 4 que faz com que o
248      //pacman se movimente no sentido negativo em que sua direção esta posicionada
249
250     if(key==GLUT_KEY_DOWN)
251     {
252         direc = 3;
```

```
253     }
254
255      //caso a tecla F1 seja pressionada muda para a visão do jogo para visão do mapa inteiro
256
257     if(key==GLUT_KEY_F1)
258     {
259         changevision=0;
260     }
261
262      //caso a tecla F2 seja pressionada muda a visão para a visão em terceira pessoa seguindo o pacman
      por traz
263
264     if(key==GLUT_KEY_F2)
265     {
266         changevision=1;
267     }
268
269     //caso a tecla F3 seja pressionada muda a visão para a visão do jogo escolhida é a visão aproximada
      seguindo o pacman
270
271     if(key==GLUT_KEY_F3)
272     {
273         changevision=2;
274     }
275 }
```

5.6.3.4 stop()

```
void Pacman::stop (
            int bateu )
```

Função responsável por tratar a colisão no pacman quando ela ocorre.

Parâmetros

bateu	Identifica se houve a colisão e o lado em que esta o objeto que colidiu

Definição na linha 304 do arquivo Pacman.cpp.

```
305 {
306
307     //caso o pacman colida com uma parede acima dele faz com que ele retorne uma
308     //quantia insignificante para na direção contraria da colisão
309
310     if (bateu == 1)
311     {
312         x = x - 0.1;
313     }
314
315     //caso o pacman colida com uma parede abaixo dele faz com que ele retorne uma
316     //quantia insignificante para na direção contraria da colisão
317
318     if (bateu == 2)
319     {
320         x = x + 0.1;
321     }
322
323     //caso o pacman colida com uma parede a direita dele faz com que ele retorne uma
324     //quantia insignificante para na direção contraria da colisão
325
326     if (bateu == 3)
327     {
328         z = z - 0.1;
329     }
330
331     //caso o pacman colida com uma parede a esquerda dele faz com que ele retorne uma
332     //quantia insignificante para na direção contraria da colisão
333
334     if (bateu == 4)
335     {
336         z = z + 0.1;
337     }
338
339     //caso o pacman colida com fantsma faz com que ele pare
```

```
340
341     if (bateu == 5)
342     {
343         direc = 0;
344
345     }
346 }
```

5.6.3.5 comeu()

```
void Pacman::comeu (
             bool tip )
```

Verifica se o pacman comeu vida ou comida.

Parâmetros

tip	Passa o tipo do objeto que o pacman comeu

Definição na linha 348 do arquivo Pacman.cpp.

```
349 {
350     if (tip)
351         ++vidas;
352     else
353         ++cont;
354 }
```

5.6.3.6 getx()

```
float Pacman::getx ( )
```

Retorna o valor da posicao onde o pacman esta em x.

Retorna

x Valor da posicao onde o pacman esta em x

Definição na linha 379 do arquivo Pacman.cpp.

```
380 {
381     return x;
382 }
```

5.6.3.7 gety()

```
float Pacman::gety ( )
```

Retorna o valor da posicao onde o pacman esta em y.

Retorna

y Valor da posicao onde o pacman esta em y

Definição na linha 384 do arquivo Pacman.cpp.

```
385 {
386     return y;
387 }
```

5.6.3.8 getz()

```
float Pacman::getz ( )
```

Retorna o valor da posicao onde o pacman esta em z.

Retorna

z Valor da posicao onde o pacman esta em z

Definição na linha 389 do arquivo Pacman.cpp.

```
390 {
391     return z;
392 }
```

5.6.3.9 getr()

```
float Pacman::getr ( )
```

Retorna o valor do raio do pacman.

Retorna

r Valor da raio do pacman

Definição na linha 394 do arquivo Pacman.cpp.

```
395 {
396     return r;
397 }
```

5.6.3.10 getangulo()

```
float Pacman::getangulo ( )
```

Retorna o angulo para o qual a frente do pacman esta direcionado.

Retorna

angdir Angulo para o qual a frente do pcman esta direcionado

Definição na linha 404 do arquivo Pacman.cpp.

```
405 {
406     return angdir;
407 }
```

5.6.3.11 vida()

```
int Pacman::vida ( )
```

Retorna o numero de vidas que o pacman tem.

Retorna

vidas Número de vidas do pacman

Definição na linha 360 do arquivo Pacman.cpp.

```
361 {
362     return vidas;
363 }
```

5.6.3.12 temvida()

```
bool Pacman::temvida ( )
```

Retorna se o pacman tem alguma vida.

Retorna

true Caso tenha vida
false Caso não tenha vida

Definição na linha 372 do arquivo Pacman.cpp.

```
373 {
374     if (vidas > 0)
375         return true;
376     return false;
377 }
```

5.6.3.13 pontos()

```
int Pacman::pontos ( )
```

Retorna se o numero de comidas que o pacman comeu ou seja o numero de pontos do score.

Retorna

cont Número de comidas que o pacman comeu ou seja o numero de pontos do score

Definição na linha 356 do arquivo Pacman.cpp.

```
357 {
358     return cont;
359 }
```

5.6.3.14 setpontos()

```
void Pacman::setpontos (
            int pontos )
```

Muda o valor de pontos ou comidas que o pacman comeu.

Parâmetros

pontos	Número de pontos ou comidas que ira modificar no pacman

Definição na linha 399 do arquivo Pacman.cpp.

```
400 {
401     cont=pontos;
402 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- Pacman.hpp
- Pacman.cpp

5.7 Referência da Classe Parede

A classe Parede implementa a modelagem e as funcionalisdades dos blocos que formam as paredes do mapa.

`#include <parede.hpp>`

Diagrama de hierarquia para Parede:

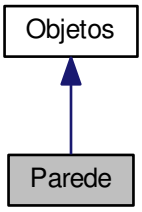
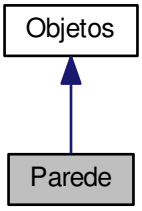


Diagrama de colaboração para Parede:



Membros Públicos

- Parede (float x, float y, float z, float c, float l, float h)

  Construtor da Parede.
- void desenha ()

  Função que desenha os blocos da parede no mapa do jogo.
- void **atualiza** (int value)
- int colisao (float x, float z, float t)

  Função que verifica se um objeto, com a posição e tamanho passados por parâmetro, colide com o bloco da parede.

5.7.1 Descrição detalhada

A classe Parede implementa a modelagem e as funcionalisdades dos blocos que formam as paredes do mapa.

Definição na linha 13 do arquivo parede.hpp.

5.7.2 Construtores e Destrutores

5.7.2.1 Parede()

```
Parede::Parede (
             float x,
             float y,
             float z,
             float c,
             float l,
             float h )
```

Construtor da Parede.

Parâmetros

x	Posição da parede na coordenada x no plano do jogo.
y	Posição da parede na coordenada y no plano do jogo.
z	Posição da parede na coordenada z no plano do jogo.
c	Comprimento da parede.
l	Largura da parede.
h	Autura da parede.

Definição na linha 9 do arquivo parede.cpp.

```
10 {
11     this->x=x;
12     this->y=y;
13     this->z=z;
14     this->c=c;
15     this->l=l;
16     this->h=h;
17 }
```

5.7.3 Funções membros

5.7.3.1 desenha()

```
void Parede::desenha ( )  [virtual]
```

Função que desenha os blocos da parede no mapa do jogo.

Os vértices são desenhados com base nos parametros recebidos no construtor. além dos vértices são definidas as normais de cada face dos blocos.

Implementa Objetos.

Definição na linha 68 do arquivo parede.cpp.

```
69 {
70
71     glPushMatrix();
72     glTranslatef(x,0,z);
73     glLineWidth(3.0f);
74     glColor3f(1.0f, 0.0f, 0.0f);
75     glBegin(GL_QUADS);
76     glNormal3f(0.0, -1, 0);
77     glVertex3f( c / 2, 0,  l / 2);
78     glVertex3f( c / 2, 0,  -l / 2);
79     glVertex3f( -c / 2, 0,  -l / 2);
80     glVertex3f( -c / 2, 0,  l / 2);
81     glEnd();
82
83     glEnable (GL_TEXTURE_2D);
84     glBindTexture (GL_TEXTURE_2D, 12);
85     glBegin (GL_POLYGON);
86     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
87     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
88     glColor3f(0.8, 0.8, 0.8);
89     glBegin(GL_QUADS);
90     glNormal3f(0.0, 1.0, 0);
91     glTexCoord2f (0.0f,0.0f);glVertex3f( c / 2, h,  l / 2);
92     glTexCoord2f (1.0f, 0.0f);glVertex3f( c / 2, h,  -l / 2);
93     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  -l / 2);
94     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, h,  l / 2);
95     glEnd();
96
97
98     glColor3f(0.8, 0.8, 0.8);
99     glBegin(GL_QUADS);
100     glNormal3f(1.0, 0.0, 0.0);
101     glTexCoord2f (0.0f,0.0f);glVertex3f( c / 2, 0,  l / 2);
102     glTexCoord2f (1.0f, 0.0f);glVertex3f( c / 2, 0,  -l / 2);
103     glTexCoord2f (1.0f, 1.0f);glVertex3f( c / 2, h,  -l / 2);
104     glTexCoord2f (0.0f, 1.0f);glVertex3f( c / 2, h,  l / 2);
105     glEnd();
106
107     glColor3f(0.8, 0.8, 0.8);
108     glBegin(GL_QUADS);
109     glNormal3f(-1.0, 0.0, 0.0);
110     glTexCoord2f (0.0f,0.0f);glVertex3f( -c / 2, 0,  l / 2);
111     glTexCoord2f (1.0f, 0.0f);glVertex3f( -c / 2, 0,  -l / 2);
112     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  -l / 2);
113     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, h,  l / 2);
114     glEnd();
115     glCullFace(GL_FRONT);
116     glColor3f(0.8, 0.8, 0.8);
117     glBegin(GL_QUADS);
118     glNormal3f(-1.0, 0.0, 0.0);
119     glTexCoord2f (0.0f,0.0f);glVertex3f( -c / 2, 0,  l / 2);
120     glTexCoord2f (1.0f, 0.0f);glVertex3f( -c / 2, 0,  -l / 2);
121     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  -l / 2);
122     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, h,  l / 2);
123     glEnd();
124     glCullFace(GL_BACK);
125
126
127     glColor3f(0.8, 0.8, 0.8);
128     glBegin(GL_QUADS);
```

```
129     glNormal3f(0.0, 0.0, -1);
130     glTexCoord2f (0.0f,0.0f);glVertex3f( c / 2, 0,  -l / 2);
131     glTexCoord2f (1.0f, 0.0f);glVertex3f( c / 2, h,  -l / 2);
132     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  -l / 2);
133     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, 0,  -l / 2);
134     glEnd();
135     glCullFace(GL_FRONT);
136     glColor3f(0.8, 0.8, 0.8);
137     glBegin(GL_QUADS);
138     glNormal3f(0.0, 0.0, -1);
139     glTexCoord2f (0.0f,0.0f);glVertex3f( c / 2, 0,  -l / 2);
140     glTexCoord2f (1.0f, 0.0f);glVertex3f( c / 2, h,  -l / 2);
141     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  -l / 2);
142     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, 0,  -l / 2);
143     glEnd();
144     glCullFace(GL_BACK);
145
146     glColor3f(0.8, 0.8, 0.8);
147     glBegin(GL_QUADS);
148     glNormal3f(0.0, 0.0, 1);
149     glTexCoord2f (0.0f,0.0f);glVertex3f( c / 2, 0,  l / 2);
150     glTexCoord2f (1.0f, 0.0f);glVertex3f( c / 2, h,  l / 2);
151     glTexCoord2f (1.0f, 1.0f);glVertex3f( -c / 2, h,  l / 2);
152     glTexCoord2f (0.0f, 1.0f);glVertex3f( -c / 2, 0,  l / 2);
153     glEnd();
154
155     glDisable (GL_TEXTURE_2D);
156     glPopMatrix();
157 }
```

5.7.3.2 colisao()

```
int Parede::colisao (
            float x,
            float z,
            float t )  [virtual]
```

Função que verifica se um objeto, com a posição e tamanho passados por parâmetro, colide com o bloco da parede.

Cada direção que a colisão a colisão pode ocorrer, retorna um valor diferente.

Parâmetros

x	Posição em x do objeto comparado.
z	Posição em z do objeto comparado.
t	Raio do objeto que se deseja verificar a colisão

Retorna

Valores interios de 1 a 4, que carrega a informação referente a direção que ocorreu a colisão.

Implementa Objetos.

Definição na linha 32 do arquivo parede.cpp.

```
33 {
34
35
36     if ( (xc+tc<=(x+c/2))&&(xc+tc>=(x-c/2))&&(zc<=(z+l/2))&&(zc>=(z-l/2)))
37     {
38         return 1;
39     }
40     else if((xc-tc<=(x+c/2))&&(xc-tc>=(x-c/2))&&(zc<=(z+l/2))&&(zc>=(z-l/2)))
```

```
41      {
42          return 2;
43      }
44      else if((xc<=(x+c/2))&&(xc>=(x-c/2))&&(zc+tc<=(z+l/2))&&(zc+tc>=(z-l/2)))
45      {
46          return 3;
47      }
48      else if((xc<=(x+c/2))&&(xc>=(x-c/2))&&(zc-tc<=(z+l/2))&&(zc-tc>=(z-l/2)))
49      {
50          return 4;
51      }
52      return 0;
53
54  }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- parede.hpp
- parede.cpp

Capítulo 6

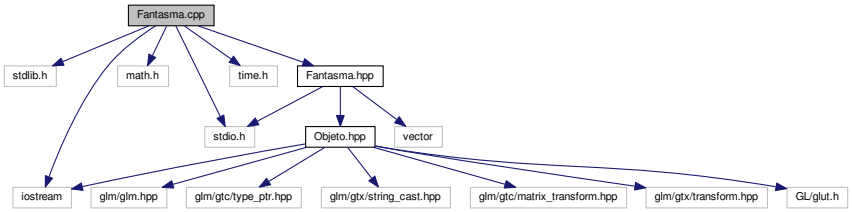
Arquivos

6.1 Referência do Arquivo Fantasma.cpp

Implementação da classe Fantasma.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <time.h>
#include "Fantasma.hpp"
```

Gráfico de dependência de inclusões para Fantasma.cpp:



6.1.1 Descrição detalhada

Implementação da classe Fantasma.

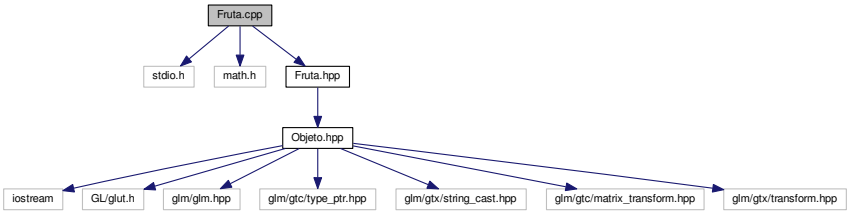
6.2 Referência do Arquivo Fruta.cpp

Implementação da classe Fruta.

```
#include <stdio.h>
#include <math.h>
```

```
#include "Fruta.hpp"
```

Gráfico de dependência de inclusões para Fruta.cpp:



6.2.1 Descrição detalhada

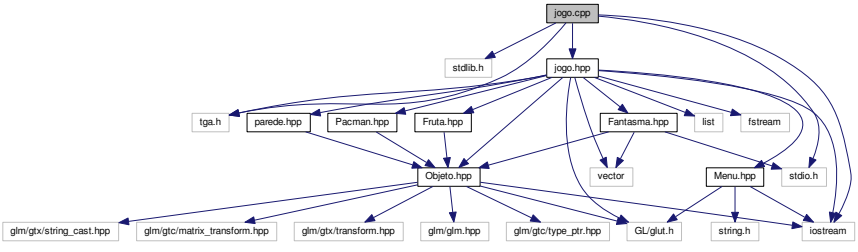
Implementação da classe Fruta.

6.3 Referência do Arquivo jogo.cpp

Implementação da classe jogo.

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "jogo.hpp"
#include "tga.h"
```

Gráfico de dependência de inclusões para jogo.cpp:



6.3.1 Descrição detalhada

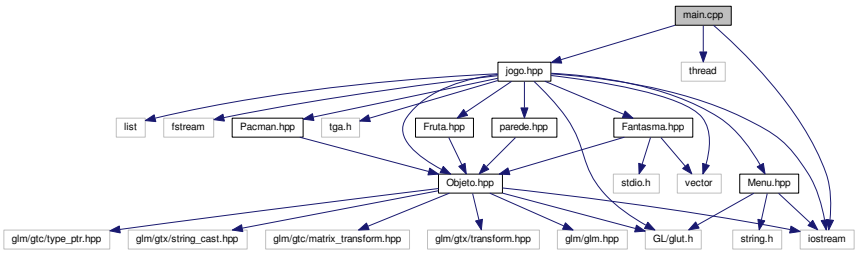
Implementação da classe jogo.

6.4 Referência do Arquivo main.cpp

Jogo baseado no clássico arcade PAC-MAN, implementado em c++ com as blibliotecas do Open-GL.

```
#include <iostream>
#include <thread>
#include "jogo.hpp"
```

Gráfico de dependência de inclusões para main.cpp:



Funções

- void **desenha** ()
- void **atualizatamanho** (GLsizei w, GLsizei h)
- void **KeyboardHandle** (unsigned char key, int x, int y)
- void **MouseHandle** (int button, int state, int x, int y)
- void **SpecialKeyHandle** (int key, int x, int y)
- void **atualiza** (int value)
- void **glInit** (void)
- int **main** (int argc, char ∗∗argv)

Variáveis

- Jogo ∗ **ptr_**

6.4.1 Descrição detalhada

Jogo baseado no clássico arcade PAC-MAN, implementado em c++ com as blibliotecas do Open-GL.

Autor

Douglas Venâncio — RA- 28614
Filipe Pena — RA- 30213
Marco Antonio — RA- 30749 . Giovani Bernardes Vitor

ECO034 - 2018

Data

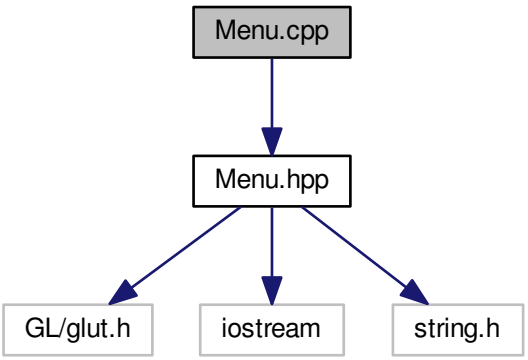
11-05-2018

6.5 Referência do Arquivo Menu.cpp

Implementação da classe Menu.

```
#include "Menu.hpp"
```

Gráfico de dependência de inclusões para Menu.cpp:



6.5.1 Descrição detalhada

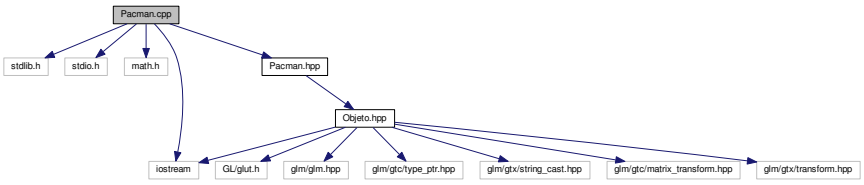
Implementação da classe Menu.

6.6 Referência do Arquivo Pacman.cpp

Implementação da classe Pacman.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include "Pacman.hpp"
```

Gráfico de dependência de inclusões para Pacman.cpp:



6.6.1 Descrição detalhada

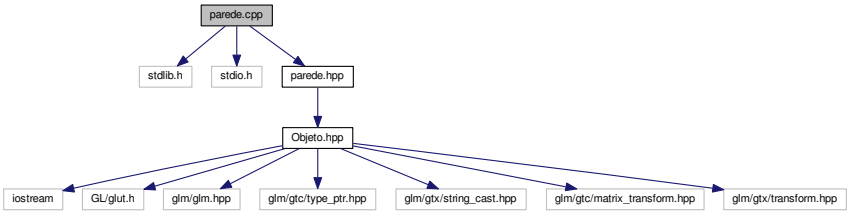
Implementação da classe Pacman.

6.7 Referência do Arquivo parede.cpp

Implementação da classe parede.

```
#include <stdlib.h>
#include <stdio.h>
#include "parede.hpp"
```

Gráfico de dependência de inclusões para parede.cpp:



6.7.1 Descrição detalhada

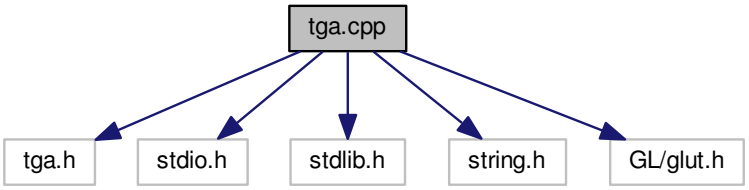
Implementação da classe parede.

6.8 Referência do Arquivo tga.cpp

Implementação da classe tga.

```
#include "tga.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <GL/glut.h>
```

Gráfico de dependência de inclusões para tga.cpp:



Funções

- int **checkSize** (int x)
- unsigned char * **getRGBA** (FILE *s, int size)
- unsigned char * **getRGB** (FILE *s, int size)
- unsigned char * **getGray** (FILE *s, int size)
- unsigned char * **getData** (FILE *s, int sz, int iBits)
- int **returnError** (FILE *s, int error)
- int loadTGA (char *name, int id)

  Função referente ao caregamento da imagem na textura.

Variáveis

- GLenum **texFormat**

6.8.1 Descrição detalhada

Implementação da classe tga.

6.8.2 Funções

6.8.2.1 loadTGA()

```
int loadTGA (
            char * name,
            int id )
```

Função referente ao caregamento da imagem na textura.

Parâmetros

*name	nome do arquivo a ser carregado
id	id da textura apos ela ser carregada para ligação da um objeto com ela posteriormente

Definição na linha 192 do arquivo tga.cpp.

```
193 {
194     unsigned char type[4];
195     unsigned char info[7];
196     unsigned char *imageData = NULL;
197     int imageWidth, imageHeight;
198     int imageBits, size;
199     FILE *s;
200
201     if (!(s = fopen (name, "r+bt
))))
202         return TGA_FILE_NOT_FOUND;
203
204     fread (&type, sizeof (char), 3, s); // read in colormap info and image type, byte 0 ignored
205     fseek (s, 12, SEEK_SET);            // seek past the header and useless info
206     fread (&info, sizeof (char), 6, s);
207
208     if (type[1] != 0 || (type[2] != 2 && type[2] != 3))
```

```
209         returnError (s, TGA_BAD_IMAGE_TYPE);
210
211     imageWidth = info[0] + info[1] * 256;
212     imageHeight = info[2] + info[3] * 256;
213     imageBits = info[4];
214
215     size = imageWidth * imageHeight;
216
217     /* make sure dimension is a power of 2 */
218     if (!checkSize (imageWidth) || !checkSize (imageHeight))
219         returnError (s, TGA_BAD_DIMENSION);
220
221     /* make sure we are loading a supported type */
222     if (imageBits != 32 && imageBits != 24 && imageBits != 8)
223         returnError (s, TGA_BAD_BITS);
224
225     imageData = getData (s, size, imageBits);
226
227     /* no image data */
228     if (imageData == NULL)
229         returnError (s, TGA_BAD_DATA);
230
231     fclose (s);
232
233     glBindTexture (GL_TEXTURE_2D, id);
234     glPixelStorei (GL_UNPACK_ALIGNMENT, 1);
235     glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
236     glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
237     /* glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST); */
238     glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
239     /* glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST); */
240     glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
241     glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
242     glTexImage2D (GL_TEXTURE_2D, 0, texFormat, imageWidth, imageHeight, 0, texFormat, GL_UNSIGNED_BYTE,
    imageData);
243
244     /* release data, its been uploaded */
245     free (imageData);
246
247     return 1;
248 }
```