

# Introdução a Computação Gráfica

## Projeto final: aMaze Story

Luiz Fernando Gomes de Oliveira

Gustavo Jaruga Cruz

Guilherme Fay Vergara

Resumo— Abstract



## 1 INTRODUÇÃO

**D**ESCRIÇÃO das principais funcionalidades do programa:

- Objetivo do programa: Quais são os principais objetivos do presente programa dentro do contexto da disciplina. Listar os principais pontos a serem abordados no programa
- Entradas do Programa: como parametrizar tais entradas? Qual o formato dos parâmetros de entrada? Quais são os parâmetros internos do programa?(Como mostrado na tabela 1)
- Saídas Do Programa: Quais são as saídas do programa? Qual o formato dessas saídas?

**Tabela 1:** Principais Funções Implementadas.

Função	Descrição
<code>getTitle(self,url)</code>	Função que recebe uma url de um pdf de um capítulo do redbook e imprime o título
<code>indexToString(self)</code>	Função que retorna uma string com todos o índice criado
<code>salvar(self,path)</code>	Função que cria um arquivo com o índice criado
<code>createIndex(self)</code>	Função que cria o índice
<code>getIndex(self)</code>	Função que retorna os elementos do índice
<code>printIndex(self)</code>	Função que imprime os elementos do índice
<code>printUrl(self)</code>	Função que imprime as url que são usadas pela createIndex()

## 2 DESENVOLVIMENTO

Na seção desenvolvimento deve ser respondidas as seguintes perguntas:

- Estrutura do Programa: Qual a estruturação/arquitetura do Programa?
- Qual é o procedimento para a execução do programa?
- Quais artefatos são necessários para a execução do programa?
- Quais os problemas técnicos enfrentados no desenvolvimento do programa?

- Como os pontos relacionados à disciplina foram abordados no problema? Quais as lições aprendidas? Quais as principais dificuldades?
- Quais elementos teóricos abordado na disciplina foram implementados no programa?
- Quais adaptações, extensões, bibliotecas externas, foram necessários para a solução do problema?
- Caso use parte de códigos disponibilizados na Web, colocar referência <sup>1</sup>

As Figuras são simplesmente inseridas como mostrado na Fig. 1

**Figura 1:** Arquitetura do Programa.

## 2.1 Artefatos

Os artefatos entregues devem ser documentados no relatório:

- Arquivos contidos no programa. Lista dos nomes dos arquivos, assim como a extensão dos arquivo
- Arquivo README, com instruções de uso do software desenvolvido e necessidades técnicas para a execução do programa
- Arquivos de entrada/saída, caso necessário.

## 3 CASO DE TESTE

Nessa seção deve ser apresentado pelo menos um exemplo de caso de teste. Se não for especificado na descrição do problema, ela deve definida, explicada e ilustrada pelos autores.

## 4 CONCLUSÃO

Discutir os principais pontos relativos ao desenvolvimento do programa:

1. A home-page de onde tirei este material:<http://en.wikibooks.org/wiki/LaTeX>.Estou formatando para  $\LaTeX$  apenas para os estudantes irem se orientando de como e o quê escrever. Assim, me isento de responsabilidade sobre o conteúdo deste texto. Dúvidas: [carla\(rocha.carla@gmail.com\)](mailto:carla(rocha.carla@gmail.com))

- Dificuldades encontradas em atingir os objetivos propostos. Caso não tenha sido possível, concluir 100% da tarefa, listar razões para tal.
- Sugestões de melhorias do programa.
- Pontos teóricos mais relevantes abordados na prática e a relevância de tais conceitos (Exemplo de aplicações que tais conceitos seriam úteis). Com citações se necessário.



**Luiz Fernando Gomes de Oliveira**

Matricula: 10/46969

E-mail: ziuloliveira@gmail.com



**Gustavo Jaruga Cruz**

Matricula: 09/0066634

E-mail: darksshades@hotmail.com



**Guilherme Fay Vergara**

Matricula: 10/45547

E-mail: guifayvergara@hotmail.com

## APÊNDICE A

### CÓDIGOS FONTES

```

1 #ifndef _CAMERAS_H_
2 #define _CAMERAS_H_
3
4 #include "defines.h"
5 #include "entidade.h"
6
7
8 #define CAMERA_ANDA 20
9 #define CAMERA_CORRE 40
10
11 class Camera
12 {
13     private:
14         Entidade entidadeCamera;
15     public:
16         float lookX, lookY, lookZ;
17         float cameraX, cameraY, cameraZ;
18
19         float angleX, angleY;
20         float angleOffsetX, angleOffsetY;
21
22         float deltaAngleX, deltaAngleY;
23         float deltaMouseX, deltaMouseY;
24         float deltaMove, deltaMoveLado;
25
26         float velocidadeMove;
27         float velocidadeMoveAndar;
28         float velocidadeMoveCorre;
29         float velocidadeVira;
30         float velocidadeViraMouse;
31
32         int xOrigem, yOrigem;
33         unsigned int ticks;
34     public:
35         Camera();
36         static Camera CameraControl;
37
38         void ajustaCamera(); //seta posicao e direcao da camera
39         void reset();
40
41         void moveFrente(bool mover);
42         void moveTraz(bool mover);
43         void moveEsquerda(bool mover);
44         void moveDireita(bool mover);
45
46         void giraEsquerda(bool mover);
47         void giraDireita(bool mover);
48         void giraCima(bool mover);
49         void giraBaixo(bool mover);
50
51         void setMouse(int x, int y);
52         void moveMouse(int x, int y);
53         //temp como public
54         void calculaDirecao(void);
55
56         //Liga ou desliga correr
57         void setCorrer(void);
58         void setAndar(void);
59     private:
60
61         void calculaMovimento(float delta);
62         void calculaMovimentoLateral(float delta);
63
64 };
65 #endif

```

```

1 //=====
2 /*
3     Classe que representa uma entidade fisica, isto e, desenhada na tela.
4
5     Utilizada para representar objetos e possivelmente inimigos, tiros.
6
7     Notas sobre flags:
8
9     Utilizar flags = ENTIDADE_TIPO_FISICA;
10    para adiciona flags |= ENTIDADE_TIPO_GHOST;
11
12    testar com if (flags & ENTIDADE_TIPO_GHOST)
13    if (flags ^ ENTIDADE_TIPO_GHOST) checa se NAO tem a flag
14
15
16

```

```

17 */
18 //=====
19 #ifndef __ENTIDADE_H_
20 #define __ENTIDADE_H_
21
22 #include <vector>
23 #include "vetor3d.h"
24 #include "defines.h"
25 #include "map.h"
26
27 //Lista de flags
28 enum
29 {
30     ENTIDADE_FLAG_NENHUM = 0,
31     ENTIDADE_FLAG_GRAVIDADE = 0x00000001,
32     ENTIDADE_FLAG_GHOST = 0x00000002,
33     ENTIDADE_FLAG_GHOST_MAP = 0x00000004,
34     ENTIDADE_FLAG_TIRO = 0x00000008,
35     ENTIDADE_FLAG_PORTA = 0x00000016
36 };
37
38
39 class Entidade
40 {
41     public:
42         static std::vector<Entidade*> EntidadeList;
43         Entidade();
44         virtual ~Entidade();
45     protected:
46         bool isColisaoObjeto(Entidade* objeto);
47         bool isColisaoTile(Tile* bloco, float posY);
48         bool isColidido();
49         bool visible;
50         bool dead;
51         std::vector<Entidade*> entidadeColidida;
52
53     public:
54         bool isColisaoMapa(Vetor3D newPosicao);
55         void setColisao(Entidade* ent);
56         void setPosicao(float x, float y, float z);
57         //Ex: int delta = getTicks() - deltaTicks;
58         //Ex: posicao = posicao + (velocidade * (delta/1000.f) );
59         int deltaTicks; //ticks da ultima vez que calculou o movimento
60         Vetor3D posicao;
61         Vetor3D velocidade;
62         Vetor3D aceleracao;
63         Vetor3D maxVelocidade;
64         Vetor3D tamanho;
65         int flags;
66         bool showWired;
67     public:
68         bool isVisible();
69         void setTamanho(float newTamanho);
70     public:
71         void reset();
72         void removeFromEntidadeList();
73         void addToEntidadeList();
74
75         virtual bool carregaModelo(char* file);
76         virtual void loop();
77         virtual void render();
78         virtual void cleanup();
79         virtual void executaColisao();
80         virtual void testaColisao();
81
82
83 };
84
85
86 #endif

```

```

1 #ifndef __FRAMERATE_H_
2 #define __FRAMERATE_H_
3
4 #include "defines.h"
5
6
7 class FrameRate
8 {
9     private:
10         unsigned int ticks;
11         unsigned int ticksControl;
12         unsigned int frames;
13         float fps;
14     public:
15         void loop();

```

```

16
17     bool fpsCap;
18
19     void setFPSCap(bool cap);
20     bool isFPSCap();
21     float getFPS();
22     FrameRate();
23
24     void regulaFPS();
25
26     static FrameRate FPSControl;
27 };
28
29
30 #endif

1 #ifndef _MAPS_H_
2 #define _MAPS_H_
3
4 #include "defines.h"
5 #include "tile.h"
6 #include <vector>
7 #include <stdio.h>
8
9 class Map
10 {
11     private:
12         std::vector<Tile> listaTiles;
13         std::vector<Tile> listaRetangulos;
14         void geraQuadradosOtimizados();
15
16         int RENDER_MODE;
17
18
19         void renderTile(unsigned int i);
20         void renderTileOtimizado(unsigned int i);
21         void renderBloco(float width, float height, float flatness, bool left,
22                         bool right, bool front, bool back, bool top, int TYPE);
23         bool mostraWired;
24     public:
25
26         void reset();
27         Tile* getTile(int x, int y);
28         inline int getX(int i);
29         inline int getY(int i);
30         int MAP_HEIGHT;
31         int MAP_WIDTH;
32
33         float origemX; //Posicao aonde o mapa comeca a renderizar,
34         float origemZ; //Tile 0,0, aumenta pra direita-baixo
35
36         void setWired(int wired);
37         bool isWire();
38
39         Map();
40
41         void render();
42         void render(float cameraX, float cameraY, float cameraZ);
43         int load(char* filename);
44
45         void iniciaDisplayList();
46         GLuint dlMap;
47
48         //Usado pra outras classes obterem info sobre o mapa.
49         static Map MapControl;
50
51
52
53         //Operator overload
54         inline Tile* operator () (const int x, const int y)
55         {
56             return this->getTile(x,y);
57         }
58
59
60
61 };
62
63
64
65
66 #endif

1 #ifndef _TEXTURELOADER_H_
2 #define _TEXTURELOADER_H_
3

```

```

4 #include "defines.h"
5
6 //Represents an image
7 class Image {
8     public:
9         Image(char* ps, int w, int h);
10        ~Image();
11
12        /* An array of the form (R1, G1, B1, R2, G2, B2, ...) indicating the
13         * color of each pixel in image. Color components range from 0 to 255.
14         * The array starts the bottom-left pixel, then moves right to the end
15         * of the row, then moves up to the next column, and so on. This is the
16         * format in which OpenGL likes images.
17         */
18        //Array de pixels no formato R,G,B, R1,G1,B1
19        //Começa de baixo-esquerda, formato do OpenGL nativo
20        char* pixels;
21        int width;
22        int height;
23 };
24
25 #endif
26
27 namespace texture
28 {
29     //Le uma imagem BMP do arquivo
30     extern GLuint loadTextureBMP(const char* filename);
31     extern Image* loadBMP(const char* filename);
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

10 extern void moveMouse(int x, int y);
11
12 #endif

1 #ifndef __TEXTT_H_
2 #define __TEXTT_H_
3
4 #include "defines.h"
5 #include <stdio.h>
6
7 namespace txt
8 {
9     extern void renderBitmapString(
10         float x,
11         float y,
12         int spacing,
13         void *font,
14         char *string) ;
15
16
17
18     ///ARRUMA PROJECOES
19     extern void setProjecaoOrto();
20     extern void restauraProjecaoPerspectiva();
21
22     extern void renderText2dOrtho(float x, float y, int spacing, const char*pStr, ...);
23
24 }
25
26
27
28 #endif

1 #include "camera.h"
2
3 #include <math.h>
4 Camera Camera::CameraControl;
5 Camera::Camera()
6 {
7     angleX = 90.0f;
8     angleY = 0.0f;
9     angleOffsetX = angleOffsetY = 0;
10
11     lookX = 0.5f;
12     lookY = 0.0f;
13     lookZ = -1.0f;
14
15     cameraX = (TAMANHO_BLOCO*1) + TAMANHO_BLOCO/2;
16     cameraY = 5.0f;
17     cameraZ = (TAMANHO_BLOCO*1) + TAMANHO_BLOCO/2;
18     ///testes
19     entidadeCamera.reset();
20     entidadeCamera.addToEntidadeList();
21     entidadeCamera.setTamanho(2.5f);
22     entidadeCamera.posicao.x = ((TAMANHO_BLOCO*1) + TAMANHO_BLOCO/2) - (entidadeCamera.tamanho.x/2);
23     entidadeCamera.posicao.y = entidadeCamera.tamanho.y/2;
24     entidadeCamera.posicao.z = ((TAMANHO_BLOCO*1) + TAMANHO_BLOCO/2) - (entidadeCamera.tamanho.z/2);
25     entidadeCamera.showWired = true;
26     ///testes
27     deltaAngleX = deltaAngleY = 0.0f; //Angulo de rotacao da direcao horizontal e vertical
28
29     deltaMouseX = deltaMouseY = 0.0f;
30
31     deltaMove = deltaMoveLado = 0.0f;
32
33
34     velocidadeMoveAndar = CAMERA_ANDA;
35     velocidadeMoveCorre = CAMERA_CORRE;
36     velocidadeMove = velocidadeMoveAndar;
37     velocidadeVira = 45.f;
38     velocidadeViraMouse = 0.1f;
39
40     xOrigem = -1;
41     yOrigem = -1;
42     ticks = 0;
43
44     calculaDirecao();
45 }
46
47 void Camera::reset()
48 {
49     Camera();
50     ticks = glutGet(GLUT_ELAPSED_TIME);
51 }
52 void Camera::ajustaCamera()
53 {

```

```

54
55     if (deltaMove)
56     {
57         calculaMovimento(deltaMove);                                //Calcula posicao da camera
58         Vetor3D pos;
59         pos.x = cameraX-(entidadeCamera.tamanho.x/2);
60         pos.y = cameraY-(entidadeCamera.tamanho.y/2);
61         pos.z = cameraZ-(entidadeCamera.tamanho.z/2);
62         if( entidadeCamera.isColisaoMapa(pos) == false) //Verifica se colidiu
63             entidadeCamera.setPosicao(pos.x, pos.y, pos.z); // e setado para poder calcular colisoes com entidades no futuro
64         else
65             calculaMovimento(-deltaMove);                            //Recalcula para posicao anterior se colidiu
66     }
67
68     if (deltaMoveLado)
69     {
70         calculaMovimentoLateral(deltaMoveLado);
71         Vetor3D pos;
72         pos.x = cameraX-(entidadeCamera.tamanho.x/2);
73         pos.y = cameraY-(entidadeCamera.tamanho.y/2);
74         pos.z = cameraZ-(entidadeCamera.tamanho.z/2);
75         if (entidadeCamera.isColisaoMapa(pos) == false)
76             entidadeCamera.setPosicao(pos.x, pos.y, pos.z);
77         else
78             calculaMovimentoLateral(-deltaMoveLado);
79     }
80
81     if (deltaAngleX || deltaAngleY)
82         calculaDirecao();
83
84     gluLookAt( cameraX, cameraY, cameraZ,
85               cameraX+lookX, cameraY+lookY, cameraZ+lookZ,
86               0.0f, 1.0f, 0.0f);
87
88     entidadeCamera.render();
89
90     ticks = glutGet(GLUT_ELAPSED_TIME);
91 }
92
93 void Camera::calculaDirecao(void)
94 {
95     unsigned int deltaTicks = glutGet(GLUT_ELAPSED_TIME) - ticks;
96     float fator = deltaTicks/1000.f;
97     angleX += deltaAngleX*fator;
98     angleY += deltaAngleY*fator;
99
100     //corrige angulo
101     if ( angleX+angleOffsetX >= 360 )
102         angleX -= 360;
103     if ( angleX+angleOffsetX < 0)
104         angleX += 360;
105
106     //So permite rotacionar 180 graus em Y
107     if ( angleY+angleOffsetY >= 90 )
108         angleY = 90-angleOffsetY;
109     if ( angleY+angleOffsetY <= -90)
110         angleY = -(90+angleOffsetY);
111
112
113     lookX = sin( (angleX+angleOffsetX)*M_PI/180);
114     lookZ = cos( (angleX+angleOffsetX)*M_PI/180);
115
116     lookY = sin( (angleY+angleOffsetY)*M_PI/180);
117 }
118 void Camera::calculaMovimento(float delta)
119 {
120     //Adiciona ao movimento
121
122     unsigned int deltaTicks = glutGet(GLUT_ELAPSED_TIME) - ticks;
123     float fator = deltaTicks/1000.f;
124
125     //Fator delta vezes direcao. 0.1f para ajustar velocidade.
126     cameraX += (delta*fator) * lookX;
127     cameraZ += (delta*fator) * lookZ;
128 }
129 void Camera::calculaMovimentoLateral(float delta)
130 {
131     unsigned int deltaTicks = glutGet(GLUT_ELAPSED_TIME) - ticks;
132     float fator = deltaTicks/1000.f;
133
134     float lateralX = sin( (angleX-90)*M_PI/180);
135     float lateralZ = cos( (angleX-90)*M_PI/180);
136     //Adiciona ao movimento
137     //Fator delta vezes direcao. 0.1f para ajustar velocidade.
138     cameraX += (delta*fator) * (lateralX);
139     cameraZ += (delta*fator) * (lateralZ);

```



```
140 }
141
142
143 void Camera::moveFrente(bool mover)
144 {
145     if(mover)
146         deltaMove = velocidadeMove;
147     else
148         deltaMove = 0.0f;
149 }
150 void Camera::moveTraz(bool mover)
151 {
152     if(mover)
153         deltaMove = -velocidadeMove;
154     else
155         deltaMove = 0.0f;
156 }
157 }
158 void Camera::moveEsquerda(bool mover)
159 {
160     if(mover)
161         deltaMoveLado = -velocidadeMove;
162     else
163         deltaMoveLado = 0.0f;
164 }
165 void Camera::moveDireita(bool mover)
166 {
167     if(mover)
168         deltaMoveLado = velocidadeMove;
169     else
170         deltaMoveLado = 0.0f;
171 }
172
173 void Camera::giraEsquerda(bool mover)
174 {
175     if(mover)
176         deltaAngleX = velocidadeVira;
177     else
178         deltaAngleX = 0.0f;
179 }
180 void Camera::giraDireita(bool mover)
181 {
182     if(mover)
183         deltaAngleX = -velocidadeVira;
184     else
185         deltaAngleX = 0.0f;
186 }
187 void Camera::giraCima(bool mover)
188 {
189     if(mover)
190         deltaAngleY = velocidadeVira;
191     else
192         deltaAngleY = 0.0f;
193 }
194 void Camera::giraBaixo(bool mover)
195 {
196     if(mover)
197         deltaAngleY = -velocidadeVira;
198     else
199         deltaAngleY = 0.0f;
200 }
201
202 void Camera::setMouse(int x, int y)
203 {
204     xOrigem = x;
205     yOrigem = y;
206
207     if (xOrigem == -1) // Ambos serao -1 necessariamente
208     {
209         angleX +=angleOffsetX;
210         angleY +=angleOffsetY;
211         angleOffsetX = 0;
212         angleOffsetY = 0;
213     }
214 }
215 void Camera::moveMouse(int x, int y)
216 {
217     deltaMouseX = deltaMouseY = 0;
218     //Se houve deslocamento
219     if (xOrigem>0)
220     {
221         angleOffsetX = (xOrigem-x) * 0.1f;
222     }
223     if (yOrigem>0)
224     {
225         angleOffsetY = (yOrigem-y) * 0.1f;
```

```

226     }
227     calculaDirecao();
228 }
229
230 void Camera::setCorrer(void)
231 {
232     velocidadeMove = velocidadeMoveCorre;
233 }
234 void Camera::setAndar(void)
235 {
236     velocidadeMove = velocidadeMoveAndar;
237 }

1 #include "entidade.h"
2
3 #include <stdlib.h>
4
5
6
7
8 //=====
9 // Variaveis estaticas
10 //=====
11 std::vector<Entidade*> Entidade::EntidadeList;
12
13 //=====
14 // Construtores
15 //=====
16 Entidade::Entidade()
17 {
18     flags = ENTIDADE_FLAG_NENHUM;
19     entidadeColidida.clear();
20     deltaTicks = glutGet(GLUT_ELAPSED_TIME);
21     deltaTicks = 0;
22     tamanho.x = tamanho.y = tamanho.z = 10;
23     visible = true;
24     dead = false;
25     showWired = false;
26
27     maxVelocidade.x = maxVelocidade.y = maxVelocidade.z = 50.f;
28
29
30 }
31
32 void Entidade::reset()
33 {
34
35
36
37     Entidade();
38     deltaTicks = glutGet(GLUT_ELAPSED_TIME);
39     addToEntidadeList(); // Por algum motivo nao esta funcionando quando chamado no construtor no linux
40 }
41 Entidade::~Entidade()
42 {
43     cleanup();
44 }
45 void Entidade::cleanup()
46 {
47     removeFromEntidadeList();
48 }
49 bool Entidade::isColisaoObjeto(Entidade* objeto)
50 {
51     //Nota, o ponto posicao marca 0... ex: posicao 0 comeco do bloco final do bloco em x,y,z
52     //Tal que y mais abaixo = y e y mais alto = y+tamanhoY
53     int baixo1 = this->posicao.y;
54     int cima1 = this->posicao.y + this->tamanho.y;
55     int esquerdal = this->posicao.x;
56     int direital = this->posicao.x + this->tamanho.x;
57     int frentel = this->posicao.z;
58     int traz1 = this->posicao.z + this->tamanho.z;
59
60     int baixo2 = objeto->posicao.y;
61     int esquerda2 = objeto->posicao.x;
62     int frente2 = objeto->posicao.z;
63     int direita2 = objeto->posicao.x + objeto->tamanho.x;
64     int cima2 = objeto->posicao.y + objeto->tamanho.y;
65     int traz2 = objeto->posicao.z + objeto->tamanho.z;
66
67     if (
68         !(baixo1 > cima2) &&
69         !(cima1 < baixo2) &&
70         !(esquerdal > direita2) &&
71         !(direital < esquerda2) &&
72         !(frentel > traz2) &&
73         !(traz1 < frente2)

```

```

74     )
75     {
76         return true;
77     }
78
79     return false;
80
81 }
82 //=====
83 // Retorna true se estiver colidindo com o mapa
84 //=====
85 bool Entidade::isColisaoMapa(Vetor3D newPosicao)
86 {
87     //Calcula o Id do tile que deve ser testado
88     //Ex: X = 5 tal que startX = 0,41 = 0 endX = 1,3 = 1
89     int startX = (newPosicao.x) / TAMANHO_BLOCO;
90     int startZ = (newPosicao.z) / TAMANHO_BLOCO;
91     int endX = (newPosicao.x + (tamanho.x)) / TAMANHO_BLOCO;
92     int endZ = (newPosicao.z + (tamanho.z)) / TAMANHO_BLOCO;
93
94     //Checa colisoes com os tiles
95     for(int iZ = startZ; iZ <= endZ; iZ++) {
96         for(int iX = startX; iX <= endX; iX++) {
97             Tile* bloco = Map::MapControl(iX, iZ);
98
99             if(isColisaoTile(bloco, newPosicao.y))
100                 return true;
101         }
102     }
103     return false;
104 }
105 bool Entidade::isColisaoTile(Tile* bloco, float posY)
106 {
107     if ( //Se o bloco for uma parede e se posY for menor que a altura maxima Y do bloco, ou seja, esta abaixo do bloco
108         (bloco->typeId & TILE_TIPO_PAREDE) &&
109         (posY < (bloco->posY+bloco->tamanho) ) &&
110         ((posY+tamanho.y) > bloco->posY)
111     )
112         return true;
113     else
114         return false;
115 }
116
117 void Entidade::removeFromEntidadeList()
118 {
119     for(unsigned int i = 0; i < EntidadeList.size(); i++)
120     {
121         if (EntidadeList[i] == this)
122             EntidadeList.erase(EntidadeList.begin()+i);
123     }
124 }
125 void Entidade::addToEntidadeList()
126 {
127
128
129     for(unsigned int i = 0; i < EntidadeList.size(); i++)
130     {
131         if (EntidadeList[i] == this)
132             return; //Se ja estiver na lista, retorna
133     }
134
135     EntidadeList.push_back(this);
136 }
137
138 bool Entidade::carregaModelo(char* file){return true;}
139 //=====
140 // Executa acoes do loop, aceleracao, velocidade.
141 //=====
142 void Entidade::loop()
143 {
144     if(dead) return;
145     //deltaTicks reseta o render
146     int delta = glutGet(GLUT_ELAPSED_TIME) - deltaTicks;
147     float fator = delta/1000.f;
148
149     if (flags & ENTIDADE_FLAG_GRAVIDADE)
150         aceleracao.y = -15.f; // sistemas de coordenadas do openGL -y baixo
151
152     //Calcula aceleracoes
153     if ( velocidade.x + aceleracao.x <= maxVelocidade.x)
154         velocidade.x += (aceleracao.x * fator);
155     if ( velocidade.y + aceleracao.y <= maxVelocidade.y)
156         velocidade.y += (aceleracao.y * fator);
157     if ( velocidade.z + aceleracao.z <= maxVelocidade.z)
158         velocidade.z += (aceleracao.z * fator);
159

```

```

160 Vetor3D newPosicao = posicao + (velocidade * fator );
161
162 if (isColisaoMapa(newPosicao) == false)
163     posicao = newPosicao;
164 else
165 {
166     velocidade.x = 0;
167     velocidade.z = 0;
168     aceleracao.x = 0;
169     aceleracao.z = 0;
170     int pos = (int)(rand() % 4);
171     switch(pos)
172     {
173         case 0:
174             aceleracao.x = 20; break;
175         case 1:
176             aceleracao.x = -20; break;
177         case 2:
178             aceleracao.z = 20; break;
179         case 3:
180             aceleracao.z = -20; break;
181         default:;
182     }
183 }
184 }
185
186 deltaTicks = glutGet(GLUT_ELAPSED_TIME);
187 }
188 void Entidade::render()
189 {
190     int tamanhoCubo = tamanho.x; //Temp, enquanto utilizar glutCube
191     glPushMatrix();
192     //Centraliza devido ao GLUT
193     glColor3f(1.0f, 0.0f, 0.0f);
194     glTranslated(posicao.x+tamanho.x/2,
195                 posicao.y+tamanho.y/2,
196                 posicao.z+tamanho.z/2);
197     if (showWired)
198         glutWireCube(tamanhoCubo);
199     else
200         glutSolidCube(tamanhoCubo);
201     glPopMatrix();
202 }
203
204 }
205 void Entidade::testaColisao()
206 {
207     if(dead) return;
208
209     unsigned int thisID = -1;
210     for (unsigned int i = 0; i < EntidadeList.size(); i++)
211         if (EntidadeList[i] == this)
212         {
213             thisID = i;
214             break;
215         }
216     //Testa com todas as entidades desta para frente.
217     //Ex: lista: 1 2 3 4
218     // thisID =1, testa com 2, 3 , 4
219     // thisID = 2 testa com 3, 4      desta, forma, thisID = 2 nao testa colisoes com 1 pois ja foi testado anteriormente.
220     for (unsigned int i = thisID+1; i < EntidadeList.size(); i++)
221     {
222         if (EntidadeList[i] != this && !EntidadeList[i]->dead)
223         {
224             if(isColisaoObjeto(EntidadeList[i]) )
225             { //adiciona colisoes tanto neste elemento quanto no testado
226                 setColisao(EntidadeList[i]);
227                 EntidadeList[i]->setColisao(this);
228             }
229         }
230     }
231 }
232 //Seta colisao atraves de metodo publico
233 void Entidade::setColisao(Entidade* ent)
234 {
235     entidadeColidida.push_back(ent);
236 }
237 bool Entidade::isColidido()
238 {
239     if (entidadeColidida.size() == 0)
240         return false;
241     else
242         return true;
243 }
244 void Entidade::executaColisao()
245 {

```

```

246     if ( !isColidido() )
247         return; // sem colisoes
248     entidadeColidida.clear();
249 }
250
251 bool Entidade::isVisible()
252 {
253     return visible;
254 }
255 void Entidade::setTamanho(float newTamanho)
256 {
257     tamanho.x = tamanho.y = tamanho.z = newTamanho;
258 }
259 void Entidade::setPosicao(float x, float y, float z)
260 {
261     posicao.x = x;
262     posicao.y = y;
263     posicao.z = z;
264 }

1 #include "framerate.h"
2
3
4 FrameRate FrameRate::FPSControl;
5
6
7
8 float FrameRate::getFPS()
9 {
10     return fps;
11 }
12 void FrameRate::setFPSCap(bool cap)
13 {
14     fpsCap = cap;
15 }
16 bool FrameRate::isFPSCap()
17 {
18     return fpsCap;
19 }
20 FrameRate::FrameRate()
21 {
22     ticks = glutGet(GLUT_ELAPSED_TIME);
23     ticksControl = glutGet(GLUT_ELAPSED_TIME);
24     frames = 0;
25     fps = 0;
26     fpsCap = false;
27 }
28
29 void FrameRate::regulaFPS()
30 {
31     unsigned int step = 1000.0f/FRAMES_PER_SECOND;
32     unsigned int decorrido = glutGet(GLUT_ELAPSED_TIME) - ticksControl;
33     if(decorrido < step )
34         Sleep( step - decorrido);
35
36     ticksControl = glutGet(GLUT_ELAPSED_TIME);
37 }
38
39 void FrameRate::loop()
40 {
41     unsigned int decorrido = glutGet(GLUT_ELAPSED_TIME) - ticks;
42     frames++;
43     if (decorrido > 1000)
44     {
45         fps = ((float)frames*1000.0f/(float)decorrido);
46
47         frames = 0;
48         ticks = glutGet(GLUT_ELAPSED_TIME);
49     }
50
51     if (fpsCap)
52         regulaFPS();
53 }
54 }

1 #include "map.h"
2
3 //Usado pra outras classes obterem info sobre o mapa.
4 Map Map::MapControl;
5
6
7 //Pega o Tile na posicao x,y do mapa.
8 //Ex: Map 1 2 3   vector sera 1 2 3 4 5 6
9 //      4 5 6
10 Tile* Map::getTile(int x, int y)
11 {

```

```

12     unsigned int ID = 0;
13
14     ID = (y * MAP_WIDTH) + x;
15
16     return &listaTiles[ID];
17 }
18 inline int Map::getX(int i)
19 {
20     return i % MAP_WIDTH;
21 }
22 inline int Map::getY(int i)
23 {
24     return (int) i/MAP_WIDTH;
25 }
26
27 Map::Map()
28 {
29     origemX = -TAMANHO_BLOCO;
30     origemZ = -TAMANHO_BLOCO;
31     mostraWired = false;
32     RENDER_MODE = 0x0007; //GL_QUADS
33 }
34
35 void Map::renderBloco(float width, float height, float flatness, bool left,
36     bool right, bool front, bool back, bool top, int TYPE = GL_QUADS)
37 {
38     float w = width/2;
39     float h = height/2;
40     float f = flatness/2;
41
42     float xTexNumber = width/TAMANHO_BLOCO;
43
44     glEnable(GL_TEXTURE_2D);
45     glBindTexture(GL_TEXTURE_2D, wallTexture);
46     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
47     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
48
49
50     glBegin(TYPE);
51     //Front
52     if(front)
53     {
54         glNormal3f(0.0f, 0.0f, 1.0f);
55         //glNormal3f(-1.0f, 0.0f, 1.0f);
56         glTexCoord2f(0.0f, 0.0f);
57         glVertex3f(-w, -h, f);
58         //glNormal3f(1.0f, 0.0f, 1.0f);
59         glTexCoord2f(xTexNumber, 0.0f);
60         glVertex3f(w, -h, f);
61         //glNormal3f(1.0f, 0.0f, 1.0f);
62         glTexCoord2f(xTexNumber, 1.0f);
63         glVertex3f(w, h, f);
64         //glNormal3f(-1.0f, 0.0f, 1.0f);
65         glTexCoord2f(0.0f, 1.0f);
66         glVertex3f(-w, h, f);
67     }
68
69     //Right
70     if(right)
71     {
72         glNormal3f(1.0f, 0.0f, 0.0f);
73         //glNormal3f(1.0f, 0.0f, -1.0f);
74         glTexCoord2f(0.0f, 0.0f);
75         glVertex3f(w, -h, -f);
76         //glNormal3f(1.0f, 0.0f, -1.0f);
77         glTexCoord2f(0.0f, 1.0f);
78         glVertex3f(w, h, -f);
79         glTexCoord2f(1.0f, 1.0f);
80         //glNormal3f(1.0f, 0.0f, 1.0f);
81         glVertex3f(w, h, f);
82         glTexCoord2f(1.0f, 0.0f);
83         //glNormal3f(1.0f, 0.0f, 1.0f);
84         glVertex3f(w, -h, f);
85     }
86
87     //Back
88     if(back)
89     {
90         glNormal3f(0.0f, 0.0f, -1.0f);
91         //glNormal3f(-1.0f, 0.0f, -1.0f);
92         glTexCoord2f(0.0f, 0.0f);
93         glVertex3f(-w, -h, -f);
94         //glNormal3f(-1.0f, 0.0f, -1.0f);
95         glTexCoord2f(0.0f, 1.0f);
96         glVertex3f(-w, h, -f);
97         //glNormal3f(1.0f, 0.0f, -1.0f);

```

```

98         glTexCoord2f(xTexNumber, 1.0f);
99         glVertex3f(w, h, -f);
100         //glNormal3f(1.0f, 0.0f, -1.0f);
101         glTexCoord2f(xTexNumber, 0.0f);
102         glVertex3f(w, -h, -f);
103     }
104
105
106     //Left
107     if(left)
108     {
109         glNormal3f(-1.0f, 0.0f, 0.0f);
110         //glNormal3f(-1.0f, 0.0f, -1.0f);
111         glTexCoord2f(0.0f, 0.0f);
112         glVertex3f(-w, -h, -f);
113         //glNormal3f(-1.0f, 0.0f, 1.0f);
114         glTexCoord2f(1.0f, 0.0f);
115         glVertex3f(-w, -h, f);
116         //glNormal3f(-1.0f, 0.0f, 1.0f);
117         glTexCoord2f(1.0f, 1.0f);
118         glVertex3f(-w, h, f);
119         //glNormal3f(-1.0f, 0.0f, -1.0f);
120         glTexCoord2f(0.0f, 1.0f);
121         glVertex3f(-w, h, -f);
122     }
123     glEnd();
124     glDisable(GL_TEXTURE_2D);
125     glBegin(TYPE);
126     //Top
127     if(top)
128     {
129         glNormal3f(0.0f, 1.0f, 0.0f);
130         //glNormal3f(-1.0f, 1.0f, -1.0f);
131         glVertex3f(-w, h, -f);
132         //glNormal3f(-1.0f, 1.0f, 1.0f);
133         glVertex3f(-w, h, f);
134         //glNormal3f(1.0f, 1.0f, 1.0f);
135         glVertex3f(w, h, f);
136         //glNormal3f(1.0f, 1.0f, -1.0f);
137         glVertex3f(w, h, -f);
138     }
139
140     ///Nao precisa imprimir fundo
141     /*
142     //Bottom
143     glNormal3f(0.0f, -1.0f, 0.0f);
144     //glNormal3f(-1.0f, -1.0f, -1.0f);
145     glVertex3f(-w, -h, -f);
146     //glNormal3f(-1.0f, -1.0f, 1.0f);
147     glVertex3f(-w, -h, f);
148     //glNormal3f(1.0f, -1.0f, 1.0f);
149     glVertex3f(w, -h, f);
150     //glNormal3f(1.0f, -1.0f, -1.0f);
151     glVertex3f(w, -h, -f);
152     */
153     glEnd();
154 }
155
156
157 void Map::render()
158 {
159     glPushMatrix();
160     float offset = (float)TAMANHO_BLOCO/2.0f;
161     glTranslated(offset, offset, offset); //Pois o glut imprime a partir do centro
162     glColor3f(COR_PAREDE);
163
164
165     for(unsigned int i = 0; i < listaTiles.size(); i++)
166     {
167         glPushMatrix();
168         renderTile(i);
169         glPopMatrix();
170     }
171
172     //Desenha chao
173     glPopMatrix();
174
175 }
176 void Map::render(float cameraX, float cameraY, float cameraZ)
177 {
178     glPushMatrix();
179     float offset = (float)TAMANHO_BLOCO/2.0f;
180     glTranslated(offset, offset, offset); //Pois o glut imprime a partir do centro
181     glColor3f(COR_PAREDE);
182
183     int indexX = (cameraX / TAMANHO_BLOCO);

```

```

184     int indexY = (cameraZ / TAMANHO_BLOCO);
185
186     int beginX = indexX - GAME_FOV;
187     int beginY = indexY - GAME_FOV;
188     int endX = indexX + GAME_FOV;
189     int endY = indexY + GAME_FOV;
190     if(endX > MAP_WIDTH)
191         endX = MAP_WIDTH;
192     if(endY > MAP_HEIGHT)
193         endY = MAP_HEIGHT;
194     if(beginX < 0)
195         beginX = 0;
196     if(beginY < 0)
197         beginY = 0;
198
199
200     for(int i = beginY; i < endY; i++)
201     {
202         for(int j = beginX; j < endX; j++)
203         {
204             glPushMatrix();
205             renderTileOptimizado(j+i*MAP_WIDTH);
206             glPopMatrix();
207         }
208     }
209
210     //Desenha chao
211     glPopMatrix();
212 }
213 void Map::renderTileOptimizado(unsigned int i)
214 {
215     //Camera no centro do quadrado 0,0,0
216     glTranslated(listaRetangulos[i].posX * TAMANHO_BLOCO,
217                 listaRetangulos[i].posY * TAMANHO_BLOCO,
218                 listaRetangulos[i].posZ * TAMANHO_BLOCO);
219
220
221     if(listaRetangulos[i].typeId == TILE_TIPO_PAREDE )
222     {
223         renderBloco(listaRetangulos[i].tamanho, listaRetangulos[i].tamanho, listaRetangulos[i].tamanho,
224                     listaRetangulos[i].left, listaRetangulos[i].right, listaRetangulos[i].front,
225                     listaRetangulos[i].back, listaRetangulos[i].top,
226                     RENDER_MODE);
227     }
228
229     else
230     if(listaRetangulos[i].typeId == TILE_TIPO_CHAO )
231     {
232         float offset = (float)TAMANHO_BLOCO/2.0f;
233         glColor3f(COR_CHAO);
234         glBegin(RENDER_MODE);
235             glNormal3f(0.0f, 1.0f, 0.0f);
236             glVertex3f(-offset, -offset, -offset);
237             glVertex3f(-offset, -offset, offset);
238             glVertex3f(offset, -offset, offset);
239             glVertex3f(offset, -offset, -offset);
240         glEnd();
241         glColor3f(COR_PAREDE);
242     }
243 }
244 }
245 void Map::renderTile(unsigned int i)
246 {
247     //Move ponto de referencia
248     if (i != 0) //No primeiro nao ha deslocamento
249     {
250         if (getY(i) > getY(i-1) ) //Se tiver pulado de linha, volta em X e avanca em Z
251             glTranslated(
252                 -(TAMANHO_BLOCO*(MAP_WIDTH-1) ),      0, TAMANHO_BLOCO);
253         else //Moveu em X
254             glTranslated(TAMANHO_BLOCO,0,0);
255     }
256     if(listaTiles[i].typeId == TILE_TIPO_PAREDE )
257     {
258         renderBloco(listaTiles[i].tamanho, listaTiles[i].tamanho,
259                     listaTiles[i].tamanho,true,true,true,true,true, RENDER_MODE);
260     }
261     else
262     if(listaTiles[i].typeId == TILE_TIPO_CHAO )
263     {
264         float offset = (float)TAMANHO_BLOCO/2.0f;
265         glColor3f(COR_CHAO);
266         glBegin(GL_QUADS);
267             glNormal3f(0.0f, 1.0f, 0.0f);
268             glVertex3f(-offset, -offset, -offset);
269             glVertex3f(-offset, -offset, offset);

```



```

270         glVertex3f(offset, -offset, offset);
271         glVertex3f(offset, -offset, -offset);
272     glEnd();
273     glColor3f(COR_PAREDE);
274 }
275 }
276 }
277 }
278
279 void Map::iniciaDisplayList()
280 {
281 }
282 }
283
284 int Map::load(char* filename)
285 {
286     listaTiles.clear();
287
288     FILE* file = fopen(filename, "r");
289
290     if(file == NULL)
291         return -1;
292
293     MAP_HEIGHT = MAP_WIDTH = 0;
294
295     //Pega o tamanho do mapa, quanto por quantos blocos
296     int error = fscanf(file, "%d-%d\n", &MAP_WIDTH, &MAP_HEIGHT);
297
298     for (int y = 0; y < MAP_HEIGHT; y++)
299     {
300         for (int x = 0; x < MAP_WIDTH; x++)
301         {
302             Tile tempTile;
303             error = fscanf(file, "[%d]_", &tempTile.typeId);
304
305             listaTiles.push_back(tempTile);
306         }
307         error = fscanf(file, "\n");
308     }
309     fclose(file);
310     ///TESTE
311     geraQuadradosOptimizados();
312     return error;
313 }
314
315 void Map::geraQuadradosOptimizados()
316 {
317     for(int iY = 0; iY < MAP_HEIGHT; iY++)
318     {
319         for(int iX = 0; iX < MAP_WIDTH; iX++) //Testa todos os blocos a depois do atual em X
320         {
321             Tile retangulo;
322             int index = iX + MAP_WIDTH*iY;
323             if (listaTiles[index].typeId == TILE_TIPO_CHAO)
324             {
325                 retangulo.typeId = TILE_TIPO_CHAO;
326                 retangulo.posX = iX;
327                 retangulo.posZ = iY;
328                 listaRetangulos.push_back(retangulo);
329                 continue;
330             }
331
332             retangulo.top = true;
333             //Se parede, verifica fora de bordas
334             if (index-1 < 0)
335                 retangulo.left = true;
336             else //Se for chao, entao tem parede naquela direcao
337                 if (listaTiles[index-1].typeId == TILE_TIPO_CHAO)
338                     retangulo.left = true;
339             if (index - MAP_WIDTH < 0)
340                 retangulo.back = true;
341             else //Se for chao, entao tem parede naquela direcao
342                 if (listaTiles[index - MAP_WIDTH].typeId == TILE_TIPO_CHAO)
343                     retangulo.back = true;
344             if (index + 1 >= (int)listaTiles.size())
345                 retangulo.right = true;
346             else //Se for chao, entao tem parede naquela direcao
347                 if (listaTiles[index + 1].typeId == TILE_TIPO_CHAO)
348                     retangulo.right = true;
349             if (index + MAP_WIDTH >= (int)listaTiles.size())
350                 retangulo.front = true;
351             else //Se for chao, entao tem parede naquela direcao
352                 if (listaTiles[index + MAP_WIDTH].typeId == TILE_TIPO_CHAO)
353                     retangulo.front = true;
354
355             retangulo.posX = iX;

```

```

356         retangulo.posZ = iY;
357         retangulo.typeId = listaTiles[index].typeId;
358
359         listaRetangulos.push_back(retangulo);
360
361     }
362 }
363 }
364
365
366
367 void Map::setWired(int wired)
368 {
369     if (wired)
370     {
371         mostraWired = true;
372         RENDER_MODE = GL_LINES;
373     }
374     else
375     {
376         mostraWired = false;
377         RENDER_MODE = GL_QUADS;
378     }
379 }
380
381 bool Map::isWire()
382 {
383     return mostraWired;
384 }
385
386 void Map::reset()
387 {
388     Map();
389 }

```

```

1 #include "textureloader.h"
2
3 #include <assert.h>
4 #include <fstream>
5
6 using namespace std;
7
8
9 Image::Image(char* ps, int w, int h) : pixels(ps), width(w), height(h) {
10
11 }
12
13 Image::~Image() {
14     delete[] pixels;
15 }
16
17 namespace {
18     //Converts a four-character array to an integer, using little-endian form
19     int toInt(const char* bytes) {
20         return (int)((unsigned char)bytes[3] << 24) |
21                ((unsigned char)bytes[2] << 16) |
22                ((unsigned char)bytes[1] << 8) |
23                (unsigned char)bytes[0]);
24     }
25
26     //Converts a two-character array to a short, using little-endian form
27     short toShort(const char* bytes) {
28         return (short)((unsigned char)bytes[1] << 8) |
29                (unsigned char)bytes[0]);
30     }
31
32     //Reads the next four bytes as an integer, using little-endian form
33     int readInt(ifstream &input) {
34         char buffer[4];
35         input.read(buffer, 4);
36         return toInt(buffer);
37     }
38
39     //Reads the next two bytes as a short, using little-endian form
40     short readShort(ifstream &input) {
41         char buffer[2];
42         input.read(buffer, 2);
43         return toShort(buffer);
44     }
45
46     //Just like auto_ptr, but for arrays
47     template<class T>
48     class auto_array {
49     private:
50         T* array;
51         mutable bool isReleased;

```

```

52     public:
53         explicit auto_array(T* array_ = NULL) :
54             array(array_), isReleased(false) {
55         }
56
57         auto_array(const auto_array<T> &aarray) {
58             array = aarray.array;
59             isReleased = aarray.isReleased;
60             aarray.isReleased = true;
61         }
62
63         ~auto_array() {
64             if (!isReleased && array != NULL) {
65                 delete[] array;
66             }
67         }
68
69         T* get() const {
70             return array;
71         }
72
73         T &operator*() const {
74             return *array;
75         }
76
77         void operator=(const auto_array<T> &aarray) {
78             if (!isReleased && array != NULL) {
79                 delete[] array;
80             }
81             array = aarray.array;
82             isReleased = aarray.isReleased;
83             aarray.isReleased = true;
84         }
85
86         T* operator->() const {
87             return array;
88         }
89
90         T* release() {
91             isReleased = true;
92             return array;
93         }
94
95         void reset(T* array_ = NULL) {
96             if (!isReleased && array != NULL) {
97                 delete[] array;
98             }
99             array = array_;
100         }
101
102         T* operator+(int i) {
103             return array + i;
104         }
105
106         T &operator[](int i) {
107             return array[i];
108         }
109     };
110 }
111
112 namespace texture {
113     GLuint loadTextureBMP(const char* filename)
114     {
115         Image* image = loadBMP(filename);
116
117         GLuint textureId;
118         glGenTextures(1, &textureId); //Make room for our texture
119         glBindTexture(GL_TEXTURE_2D, textureId); //Tell OpenGL which texture to edit
120         //Map the image to the texture
121         glTexImage2D(GL_TEXTURE_2D,
122                     0, //Always GL_TEXTURE_2D
123                     GL_RGB, //Format OpenGL uses for image
124                     image->width, image->height, //Width and height
125                     0, //The border of the image
126                     GL_RGB, //GL_RGB, because pixels are stored in RGB format
127                     GL_UNSIGNED_BYTE, //GL_UNSIGNED_BYTE, because pixels are stored
128                     //as unsigned numbers
129                     image->pixels); //The actual pixel data
130
131         delete image;
132         return textureId; //Retorna id da textura
133     }
134 }
135
136 Image* loadBMP(const char* filename) {
137     ifstream input;

```

```

138     input.open(filename, ifstream::binary);
139     assert(!input.fail() || !"Could_not_find_file");
140     char buffer[2];
141     input.read(buffer, 2);
142     assert( (buffer[0] == 'B' && buffer[1] == 'M' ) || !"Not_a_bitmap_file");
143     input.ignore(8);
144     int dataOffset = readInt(input);
145
146     //Read the header
147     int headerSize = readInt(input);
148     int width;
149     int height;
150     switch(headerSize) {
151         case 40:
152             //V3
153             width = readInt(input);
154             height = readInt(input);
155             input.ignore(2);
156             assert(readShort(input) == 24 || !"Image_is_not_24_bits_per_pixel");
157             assert(readShort(input) == 0 || !"Image_is_compressed");
158             break;
159         case 12:
160             //OS/2 V1
161             width = readShort(input);
162             height = readShort(input);
163             input.ignore(2);
164             assert(readShort(input) == 24 || !"Image_is_not_24_bits_per_pixel");
165             break;
166         case 64:
167             //OS/2 V2
168             assert(!"Can't_load_OS/2_V2_bitmaps");
169             break;
170         case 108:
171             //Windows V4
172             assert(!"Can't_load_Windows_V4_bitmaps");
173             break;
174         case 124:
175             //Windows V5
176             assert(!"Can't_load_Windows_V5_bitmaps");
177             break;
178         default:
179             assert(!"Unknown_bitmap_format");
180     }
181
182     //Read the data
183     int bytesPerRow = ((width * 3 + 3) / 4) * 4 - (width * 3 % 4);
184     int size = bytesPerRow * height;
185     auto_array<char> pixels(new char[size]);
186     input.seekg(dataOffset, ios_base::beg);
187     input.read(pixels.get(), size);
188
189     //Get the data into the right format
190     auto_array<char> pixels2(new char[width * height * 3]);
191     for(int y = 0; y < height; y++) {
192         for(int x = 0; x < width; x++) {
193             for(int c = 0; c < 3; c++) {
194                 pixels2[3 * (width * y + x) + c] =
195                     pixels[bytesPerRow * y + 3 * x + (2 - c)];
196             }
197         }
198     }
199
200     input.close();
201     return new Image(pixels2.release(), width, height);
202 }
203 }

```

```
1 #include "defines.h"
```

```
2
3 float wScreen = SCREEN_WIDTH;
4 float hScreen = SCREEN_HEIGHT;
5 GLuint wallTexture;
6 GLuint floorTexture;
```

```
1 #include "eventos.h"
```

```
2
3 #include "gamemanager.h"
```

```
4
5 void teclasNormais(unsigned char key, int x, int y)
```

```
6 {
7
8     int mod = glutGetModifiers();
9     if (mod == GLUT_ACTIVE_SHIFT)
10         Camera::CameraControl.setCorrer();
11     else
12         Camera::CameraControl.setAndar();

```

```

13
14  switch(key)
15  {
16      case 27: //ESC
17          exit(0);
18          break;
19      case 'W':
20      case 'w':
21          {
22              Camera::CameraControl.moveFrente(true);
23              break;
24          }
25      case 'S':
26      case 's':
27          {
28
29              Camera::CameraControl.moveTraz(true);
30              break;
31          }
32
33      case 'A':
34      case 'a':
35          Camera::CameraControl.moveEsquerda(true);
36          break;
37      case 'D':
38      case 'd':
39          Camera::CameraControl.moveDireita(true);
40          break;
41      case 'Q':
42      case 'q':
43          Camera::CameraControl.giraEsquerda(true);
44          break;
45      case 'E':
46      case 'e':
47          Camera::CameraControl.giraDireita(true);
48          break;
49      case '2':
50          Camera::CameraControl.giraCima(true);
51          break;
52      case '3':
53          Camera::CameraControl.giraBaixo(true);
54          break;
55      case '1': // reseta angulo Y
56          Camera::CameraControl.angleY = 0;
57          Camera::CameraControl.calculaDirecao();
58          break;
59      case 'Z':
60      case 'z':
61          Camera::CameraControl.cameraY += 2;
62          break;
63      case 'X':
64      case 'x':
65          Camera::CameraControl.cameraY -= 2;
66          break;
67      case 'C':
68      case 'c':
69          Camera::CameraControl.cameraX = 6;
70          break;
71      case 'V':
72      case 'v':
73          Camera::CameraControl.cameraY = 3;
74          break;
75      case 'B':
76      case 'b':
77          Camera::CameraControl.cameraZ = 6;
78          break;
79      case 'F':
80      case 'f':
81          {
82              GLboolean isFog = false;
83              glGetBooleanv(GL_FOG, &isFog);
84              if (isFog)
85                  glDisable(GL_FOG);
86              else
87                  glEnable(GL_FOG);
88
89              break;
90          }
91
92      case 'R':
93      case 'r':
94          if (FrameRate::FPSControl.isFPSCap())
95              FrameRate::FPSControl.setFPSCap(false);
96          else
97              FrameRate::FPSControl.setFPSCap(true);
98          break;

```

```

99         default:break;
100     }
101 }
102 void teclasNormaisUp(unsigned char key, int x, int y)
103 {
104     switch(key)
105     {
106         case 'W':
107         case 'w':
108             Camera::CameraControl.moveFrente(false);
109             break;
110         case 'S':
111         case 's':
112             Camera::CameraControl.moveTraz(false);
113             break;
114         case 'A':
115         case 'a':
116             Camera::CameraControl.moveEsquerda(false);
117             break;
118         case 'D':
119         case 'd':
120             Camera::CameraControl.moveDireita(false);
121             break;
122         case 'Q': case 'q':
123             Camera::CameraControl.giraEsquerda(false);
124             break;
125         case 'E': case 'e':
126             Camera::CameraControl.giraDireita(false);
127             break;
128         case '2':
129             Camera::CameraControl.giraCima(false);
130             break;
131         case '3':
132             Camera::CameraControl.giraBaixo(false);
133             break;
134         default:break;
135     }
136 }
137 }
138
139 void teclasEspeciais(int key, int x, int y )
140 {
141     switch(key)
142     {
143         case GLUT_KEY_UP: Camera::CameraControl.moveFrente(true); break;
144         case GLUT_KEY_DOWN: Camera::CameraControl.moveTraz(true); break;
145         case GLUT_KEY_LEFT: Camera::CameraControl.giraEsquerda(true); break;
146         case GLUT_KEY_RIGHT: Camera::CameraControl.giraDireita(true); break;
147         default: break;
148     }
149 }
150
151
152 }
153
154 void teclasEspeciaisSoltar(int key, int x, int y)
155 {
156     switch(key)
157     {
158         case GLUT_KEY_UP: Camera::CameraControl.moveFrente(false); break;
159         case GLUT_KEY_DOWN: Camera::CameraControl.moveTraz(false); break;
160         case GLUT_KEY_LEFT: Camera::CameraControl.giraEsquerda(false); break;
161         case GLUT_KEY_RIGHT: Camera::CameraControl.giraDireita(false); break;
162         default: break;
163     }
164 }
165
166 void mouseButton(int button, int state, int x, int y)
167 {
168     if (button == GLUT_LEFT_BUTTON)
169     {
170         if (state == GLUT_UP) //Reseta posicoes e ajusta deslocamento
171         {
172             Camera::CameraControl.setMouse(-1,-1);
173         }
174         else
175         {
176             Camera::CameraControl.setMouse(x,y);
177         }
178     }
179 }
180
181 void moveMouse(int x, int y)
182 {
183     Camera::CameraControl.moveMouse(x,y);
184 }

```

```

1 #include "gamemanager.h"
2 #include "eventos.h"
3
4 GameManager game;
5
6 void changeSize(int w, int h)
7 {
8     //Previne divisao por zero
9     if ( h == 0)
10         h = 1;
11
12     float ratio = w*1.0 / h;
13
14     //Usa matriz de projecao
15     glMatrixMode(GL_PROJECTION);
16     //Reseta matriz
17     glLoadIdentity();
18
19     //Arruma viewport para janela inteira
20     glViewport(0,0,w,h);
21
22     //Arruma a perspectiva correta
23     gluPerspective(45.0f, ratio, 1, GAME_FOV*TAMANHO_BLOCO);
24
25     //Volta para o modelView
26     glMatrixMode(GL_MODELVIEW);
27
28     wScreen = w;
29     hScreen = h;
30 }
31 void GameManager::inicializaRender(void)
32 {
33
34
35     glEnable(GL_LIGHTING); //Habilita luz
36     glEnable(GL_LIGHT0); //Habilita luz #0
37     glEnable(GL_LIGHT1); //Habilita luz #0
38     glEnable(GL_NORMALIZE); //Automatically normalize normals
39     glEnable(GL_COLOR_MATERIAL);
40     //glEnable(GL_LIGHT1); //Habilita luz #1
41
42     glEnable(GL_DEPTH_TEST);
43     glShadeModel(GL_SMOOTH); //Shading
44
45     glEnable(GL_CULL_FACE); //Reduz quantidade de triangulos desenhados.
46     glCullFace(GL_CW);
47
48     wallTexture = texture::loadTextureBMP("data/wall.bmp");
49     floorTexture = texture::loadTextureBMP("data/floor.bmp");
50
51
52 }
53 void GameManager::inicializa(void)
54 {
55     inicializaRender();
56     //-----
57     //Especifica a cor de fundo
58     glClearColor(0.1f,0.1f,0.5f,1.0f);
59
60
61
62
63
64     GLfloat fog_color[4] = {0.1f,0.1f,0.5f,1.0};
65     glFogfv(GL_FOG_COLOR, fog_color);
66     glFogf(GL_FOG_DENSITY, 0.35f);
67
68     glFogi(GL_FOG_MODE, GL_LINEAR);
69     glHint(GL_FOG_HINT, GL_DONT_CARE);
70     glFogf(GL_FOG_START, 1.0f);
71     glFogf(GL_FOG_END, 50.0f);
72     glEnable(GL_FOG);
73     /*
74     glFogfv(GL_FOG_COLOR, fog_color);
75     glFogf(GL_FOG_START, 10.0f );
76     glFogf(GL_FOG_END, 70.0f );
77     glFogi(GL_FOG_MODE, GL_LINEAR);
78     glEnable(GL_FOG);
79     */
80
81     Map::MapControl.reset();
82     Map::MapControl.load((char*) "test.txt");
83     Map::MapControl.iniciaDisplayList();
84
85
86     Entidade* player2 = new Entidade();

```

```

87     player2->reset();
88     player2->addToEntidadeList();
89     player2->posicao.x = 12*2;
90     player2->posicao.y = 0;
91     player2->posicao.z = 12;
92
93     player2->aceleracao.x = 15.f;
94     player2->aceleracao.z = 4.2f;
95
96     player2->setTamanho(5);
97     //testes
98     player.reset();
99     player.addToEntidadeList();
100    player.posicao.x = 12*2;
101    player.posicao.y = 0;
102    player.posicao.z = 12;
103
104    player.aceleracao.x = 10.f;
105    player.aceleracao.z = 0.2f;
106
107    player.setTamanho(5);
108
109    Map::MapControl.reset();
110
111 }
112 void desenhaTela(void)
113 {
114     game.render();
115 }
116
117 void GameManager::loop(void)
118 {
119     FrameRate::FPSControl.loop();
120     for(unsigned int i = 0; i < Entidade::EntidadeList.size(); i++)
121     {
122         Entidade::EntidadeList[i]->loop();
123     }
124 }
125
126 void GameManager::render(void)
127 {
128     //Calcula iteracoes
129     this->loop();
130
131     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
132
133     glMatrixMode(GL_MODELVIEW);
134     glLoadIdentity();
135     //Iluminacao
136     GLfloat ambientLight[] = {0.1f, 0.1f, 0.1f, 1.0f};
137     glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);
138
139     GLfloat directedLight[] = {0.7f, 0.7f, 0.7f, 0.0f};
140     GLfloat directedLightPos[] = {0.0f, 20.0f, -100.0f, 1.0f};
141
142     GLfloat light[] = {0.9f, 0.9f, 0.9f, 1.0f};
143     GLfloat lightPos[] = {100.0f, 30.0f, -10.0f, 1.0f};
144
145     //Fim Iluminacao
146
147     Camera::CameraControl.ajustaCamera();
148
149     Map::MapControl.render(Camera::CameraControl.cameraX, Camera::CameraControl.cameraY, Camera::CameraControl.cameraZ);
150     //unsigned int temp = Entidade::EntidadeList.size();
151     for(unsigned int i = 0; i < Entidade::EntidadeList.size(); i++)
152     {
153         if (Entidade::EntidadeList[i]->isVisible())
154             Entidade::EntidadeList[i]->render();
155     }
156
157     txt::renderText2dOrtho(10,10,0,"FPS: %.2f",FrameRate::FPSControl.getFPS());
158
159     glPushMatrix();
160     glColor3f(1.0f, 1.0f, 1.0f);
161     glTranslatef(directedLightPos[0],directedLightPos[1],directedLightPos[2]);
162     glutSolidSphere(10.0f, 18.0f, 18.0f);
163     glPopMatrix();
164
165     glPushMatrix();
166     glColor3f(1.0f, 0.0f, 0.0f);
167     glTranslatef(lightPos[0],lightPos[1],lightPos[2]);
168     glutSolidSphere(10.0f, 18.0f, 18.0f);
169     glPopMatrix();
170
171     glLightfv(GL_LIGHT0, GL_DIFFUSE, directedLight);

```



```

173     glLightfv(GL_LIGHT0, GL_POSITION, directedLightPos);
174
175     glLightfv(GL_LIGHT1, GL_DIFFUSE, light);
176     glLightfv(GL_LIGHT1, GL_POSITION, lightPos);
177     glutSwapBuffers();
178 }
179 void GameManager::cleanup(void)
180 {
181     for(unsigned int i = 0; i < Entidade::EntidadeList.size(); i++)
182     {
183         delete Entidade::EntidadeList[i];
184     }
185 }
186
187
188 int main(int argc, char* args[])
189 {
190     game.executa(argc, args);
191     return 0;
192 }
193 void GameManager::executa(int argc, char* args[])
194 {
195     glutInit(&argc, args);
196     glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
197     glutInitWindowPosition(100,100);
198     glutInitWindowSize(SCREEN_WIDTH, SCREEN_HEIGHT);
199     glutCreateWindow("Labirinth");
200
201     inicializa();
202
203     glutDisplayFunc(desenhaTela);
204     glutReshapeFunc(changeSize);
205     glutIdleFunc(desenhaTela);
206
207     glutKeyboardFunc(teclasNormais);
208     glutKeyboardUpFunc(teclasNormaisUp);
209     glutSpecialFunc(teclasEspeciais);
210     glutSpecialUpFunc(teclasEspeciaisSoltar);
211     glutMotionFunc(moveMouse);
212     glutMouseFunc(mouseButton);
213
214     glutIgnoreKeyRepeat(0);
215     //Entra no loop de processamento de eventos
216     glutMainLoop();
217 }

```

```

1 #include "text.h"
2
3 namespace txt
4 {
5     void renderBitmapString(
6         float x,
7         float y,
8         int spacing,
9         void *font,
10        char *string) {
11
12        char *c;
13        int x1 = x; //Guarda posicao rasterizada para computar espaco
14
15        for (c=string; *c != '\0'; c++) {
16            glRasterPos2d(x1,y);
17            glutBitmapCharacter(font, *c);
18            x1 = x1 + glutBitmapWidth(font, *c) + spacing;
19        }
20    }
21
22    void* font_glut = GLUT_BITMAP_8_BY_13;
23
24    //ARRUMA PROJECOES
25    extern void setProjecaoOrto()
26    {
27        glMatrixMode(GL_PROJECTION);
28        glPushMatrix(); //nao fecha
29        glLoadIdentity();
30
31        // coloca projecao ortografica 2d
32        gluOrtho2D(0, wScreen, hScreen, 0);
33        glMatrixMode(GL_MODELVIEW);
34    }
35    extern void restauraProjecaoPerspectiva()
36    {
37        glMatrixMode(GL_PROJECTION);
38        glPopMatrix(); // fecha o pushMatrix do projecaoOrtho
39        glMatrixMode(GL_MODELVIEW);
40    }

```

```
41
42 extern void renderText2dOrtho(float x, float y, int spacing, const char*pStr, ...)
43 {
44     char string[128];
45     va_list valist; //info das variaveis
46     va_start(valist, pStr); //inicia lista de argumentos das variaveis
47     vsprintf(string, pStr, valist); // joga string formatado para string
48     va_end(valist); // realiza operacoes de fato
49
50     glDisable(GL_LIGHTING);
51     setProjecaoOrto();
52     glPushMatrix();
53     glLoadIdentity();
54     renderBitmapString(x,y, spacing, font_glut, string);
55     glPopMatrix();
56     restauraProjecaoPerspectiva();
57     glEnable(GL_LIGHTING);
58
59 }
60 }

1 #include "tile.h"
2
3 Tile::Tile()
4 {
5     tamanho = TAMANHO_BLOCO;
6     posY = 0;
7
8     left = right = front = back = top = bottom = false;
9 }
```

## APÊNDICE B

### ANEXOS

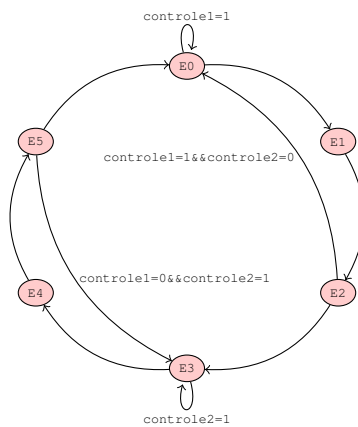


Diagrama de Estados