## Bibliotecas de Jogos

Uma biblioteca de jogos ou game engine pode ser vista como uma caixa de recursos e ferramentas para a construção de um jogo. Você pode criar um jogo sem uma biblioteca básica, assim como você pode criar uma mesa de madeira sem pregos, martelos, parafusos, chaves de fenda e serras, mas as vantagens que as ferramentas proporcionam justificam chamá-las de necessárias.

O nível dessas ferramentas varia: algumas engines se limitam a códigos, ou seja, constantes, variáveis, funções e classes relacionadas, mas outras contam com interfaces gráficas que possibilitam o desenvolvimento de um jogo sem codificação alguma. De qualquer forma, uma game engine precisa proporcionar, no mínimo, ferramentas para manipular sons, imagens (texto, imagens, etc), memória (dados) e controle (teclado, mouse, etc).

## Saga Game Library

Há duas finalidades que justificam o desenvolvimento de uma tecnologia que já existe: 1. O estudo da tecnologia, que por sua vez possibilita 2. O aperfeiçoamento da tecnologia. É certo que já existem muitas game engines, inclusive em C++, mas o estudo é o piso de todas as descobertas científicas, o que justifica e motiva o desenvolvimento de uma biblioteca de jogos didática. Esta é a nossa proposta: uma camada de orientação a objetos envolvendo a Allegro de uma forma simples e didática.

A engine, também conhecida com SAGA, ou simplesmente SGL, está estruturada em 7 camadas ou pacotes. São eles:

**1 - Sgl**

É o pacote mais geral, e que engloba todos os outros. Quando a funcionalidade de uma classe não é específica ou é usada como ferramenta auxiliar em outras classes, ela é colocada nesse pacote.

* 1. **-**  **Sgl:** é o arquivo que contém todos os includes da Allegro que serão usados e as definições dos pacotes. Além disso, contém o tipo String, como definido abaixo.

**typedef std::string String;**

* 1. **– AllegroStarter:** classe responsável por inicializar a Allegro e seus componentes, e também por desalocar os recursos quando o programa é fechado. Uma exceção é lançada caso algum dispositivo apresente problemas durante a inicialização. Também contém informações sobre a atual versão da Allegro. É baseada no padrão Singleton, e portanto existe somente uma instância da classe.
  2. **– Sgl\_exception:** é a classe gerenciadora das exceções que possam ocorrer durante a execução do programa. É uma especialização de std::exception. Abaixo um exemplo não muito prático mas que demonstra o seu uso.

**bool teste = true;**

**if( !teste ) throw sgl::Exception( "ERROR!" ); // nada acontece**

**teste = false;**

**if( !teste ) throw sgl::Exception( "ERROR!" ); // saida = "terminate called after throwing**

**// an instance of 'sgl::Exception'**

**// what(): ERROR!"**

* 1. **– Color:** possui recursos de cores, que podem ser usados tanto para colorir a tela como para alterar a cor de uma determinada fonte de texto. Ela aceita dois construtores. Com o primeiro deles é possível definir cores no formato RGB. Para isso, o construtor recebe três parâmetros que variam de 0 a 255, um para a cor vermelha, outro para verde e outro para azul, respectivamente. O segundo construtor aceita strings no formato html, e também strings com o nome em inglês de uma cor, desde que ele já esteja pré-definido. Strings mal formatadas podem levar a resultados inesperados. Material de consulta: <http://www.w3.org/TR/2010/PR-css3-color-20101028/#svg-color>

A classe tem, ainda, métodos para conversão de formatos e para mudar a cor de um objeto. Exemplos:

Definição das cores vermelho e azul escuro usando o primeiro construtor.

**Color vermelho(255,0,0);**

**Color azulEsc(0,0,139);**

Definição das cores rosa passando um nome pré-definido e das cores verde escuro e coral no formato html. Note que o símbolo ‘#’ é opcional.

**Color rosa("pink");**

**Color verdeEsc("#006400");**

**Color coral("FF7F50");**

Convertendo formatos.

**String aux;**

**aux = vermelho.getName(); // saída: aux = red**

**aux = azulEsc.toHTML(); // saída: aux = #00008b**

Alterando a cor vermelho para vermelho escuro.

**int r = 139, g = 0, b = 0;**

**vermelho.toRGB(r,g,b);**

* 1. **– Util:** classe de métodos utilitários. Exemplos:

O trecho de código abaixo converte uma string para lower case e outra para upper case, respectivamente.

**String b;**

**b = Util::toLower("AlFa"); // saida = alfa**

**b = Util::toUpper("beta"); // saida = BETA**

Função para decodificar dados que estejam em codificação Base64.

**Util::decodeBase64("aGVsbG8gd29ybGQK",b); // saida = hello world**

Funções para arredondamento para menor e para maior inteiros, respectivamente.

**float a;**

**a = Util::floorFloat(8.99); // saída = 8**

**a = Util::ceilFloat(8.99); // saída = 9**

A classe conta também com funções para descompactar strings nos formatos ZLIB e GZIP.

* 1. **– Vector2D:** tal qual o nome diz esta classe define um vetor bidimensional, iniciando-se no ponto (0,0) da tela e indo até o ponto (x,y) definido pelo construtor. A classe possui sobrecarga de operadores para o correto tratamento das operações vetoriais, além de outros métodos para a manipulação de vetores.

**static Vector2D generateVector2D( float magnitude, float angle );**

**float s;**

**Vector2D a(3,5);**

**Vector2D b(3,9);**

Produto escalar entre a e b.

**s = a.dotProduct(b); // saida = 54**

Normalização de c.

**Vector2D c(4,3);**

**c = c.normalize(); // saida = ( 0.8, 0.6 )**

Arredondamento para maior inteiro.

**c.ceil(); // saida = ( 1,1 )**

Soma entre vetores, com o operador ‘+’ sobrecarregado.

**c = a + b; // saida = ( 6,14 )**

* 1. **– Video:**  Classe responsável por gerenciar todos os recursos de vídeo SGL. A classe utiliza o padrão de projeto singleton, ou seja, permite apenas uma única instância de si mesmo. Através dela, tem-se acesso a todas as rotinas pertinentes (de atualização de tela, posicionamento, outros eventos de rotina) para o gerenciamento de vídeo SGL. Como exemplo foi criada um mini-janela. Definimos o tamanho e modo (WINDOWED, FULLSCREEN) no construtor da classe. Por padrão, a cor de fundo do vídeo é preta, podendo ser alterada. Também adicionamos um ícone e um título para nossa janela. A função refresh() deve ser chamada após alguma alteração dentro da tela. Caso contrário, nada de novo aparecerá. É importante saber também que a imagem a ser carregada, no caso o ícone, deve estar dentro pasta de projeto do programa.

**Color corVideo("aqua");**

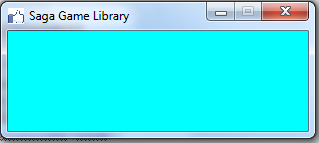
**Video video( 300, 100, DisplayMode::WINDOWED );**

**video.setIcon("nice.jpg");**

**video.setBackgroundColor(corVideo);**

**video.setTitle( "Saga Game Library" );**

**video.refresh();**



* 1. **– Bounding Box:** define um retângulo que será usado no tratamento de colisões.
  2. **– Geometrics:** ela é usada para desenhar elementos geométricos básicos. Possui como atributos um inteiro para armazenar a espessura da linha e duas variáveis da classe Color, uma para cor da linha e outra para cor de preenchimento. O construtor padrão inicializa a espessura com 1, a cor de preenchimento como branca e cor da linha como preta. Outro construtor dá ao usuário a liberdade de definir os valores como queira.

Com ela é possível desenhar linhas, triângulos, retângulos, retângulos abaulados, elipses, círculos, arcos e ranhuras.

* 1. – Scene: (game loop)
  2. Time Handler: (gerenciadora do tempo.)

int i;

TimeHandler time;

time.start();

for (i = 0; i<1000000000; i++){} //saida = 3.30395

time.pause();

double t = time.getTicks();

cout << t << endl;

1.12 – Resource e Ressource Manager: a classe Ressource é a provedora de recursos. Quando um arquivo é carregado, seja texto, imagem ou áudio, ele é armazenado em uma instância de Ressource. Se o mesmo arquivo for usado novamente, é a Ressource quem o proverá. Assim ele não precisará ser carregado pela segunda vez. Ressouce Manager é quem faz o mapeamento e e gerenciamento das Ressources.