K-ENGINE

Manual de utilização do

motor de jogos-simuladores digitais

K-engine

**Diego Potapczuk**

**GEOTEC**

**SALVADOR-BA, 2013**

SUMÁRIO

[1 INTRODUÇÃO 3](#_Toc362388217)

[2 INTERFACE DO JOGO 3](#_Toc362388218)

[3 FASES 3](#_Toc362388219)

[4 MAPAS 4](#_Toc362388220)

[5 CONSTRUÇÕES 5](#_Toc362388221)

[5.1 DECORATIVAS 5](#_Toc362388222)

[5.2 FUNCIONAIS 6](#_Toc362388223)

[6 MÚSICAS E SONS 8](#_Toc362388224)

[7 CINEMÁTICAS 8](#_Toc362388225)

[8 MINI JOGOS 10](#_Toc362388226)

[9 FLUXO DO JOGO 11](#_Toc362388227)

# INTRODUÇÃO

O motor de simulador de cidades K-engine foi desenvolvido dentro de uma pesquisa de mestrado profissional realizada para o programa GESTEC pelo autor Diego Potapczuk. Essa pesquisa faz parte de um projeto maior chamado Kimera: Cidades Imaginárias realizado pelo grupo GEOTEC.

# INTERFACE DO JOGO

As imagens utilizadas na interface do jogo devem ser exportadas para o tamanho que irão ser utilizadas dentro do jogo nos formatos JPG, no caso de não utiliza transparência, e PNG, caso a imagem utilize de transparência. É recomendado a utilização do software Photoshop e da opção “salvar para web”.

A resolução da tela do jogo é de 1024 x 768px, e todos os elementos visuais devem ser desenvolvidos com esse tamanho de tela em mente para evitar a troca de tamanho dos elementos que acarretam perda de qualidade, maior uso de memória e processamento.

# FASES

O motor permite que sejam definidas diversas fases de simulação dentro do jogo, essas fases são definidas em arquivos no formato XML localizados na pasta “data/fases”. Cada arquivo possui em seu nome o nome da fase e as seguintes informações dentro do arquivo:

* Número da fase;
* Descrição da fase;
* Tempo para concluir a fase;
* População inicial da fase;
* Quantidade de dinheiro inicial;
* Enquadramento inicial da câmera;
* Música de fundo da fase;
* Objetivos que o jogador precisa concluir para finalizar a fase;
* Mensagens da fase para o jogador;
* Tamanho do mapa;
* Elementos de textura que montam o mapa;
  + Textura do mapa;
  + Tamanho da textura;
  + Posição da textura.
* Elementos emissores de som no mapa;
  + Nome;
  + Caminho;
  + Posição.
* Elementos de construções decorativas que iniciam com a fase;
  + Nome;
  + Posição.
* Elementos de construções funcionais que iniciam com a fase;
  + Construção;
  + Posição.

# MAPAS

Os mapas do jogo são carregados a partir de um conjunto de texturas, essas texturas ficam localizadas na pasta “data/texturas”. Essas texturas possuem o ponto de vista de topo, mas ao serem carregadas pelo motor, o mapa é convertido para a perspectiva isométrica, dessa forma os elementos presentes no mapa devem ser desenvolvidos com uma rotação de 45° graus para a esquerda.

O mapa tem suas cores interpretadas pelo motor para definir em sua matriz de construção quais partes do mapa permitem a construção de elementos e quais não permitem.

Essa configuração pode ser alterada na classe “DataLoader” na função “function GetElemento( cores )” e nas constantes que definem as cores em hexadecimal no começo da classe.

public static const TOLERANCIA\_COR = 30;

public static const COR\_AREA\_LIVRE = 0x9c6b4a;

public static const COR\_AREA\_LIVRE2 = 0xc6bda5;

public static const COR\_AREA\_LIVRE3 = 0x6b8442;

public static const COR\_MAR = 0x666a00;

function GetElemento( cores )

{

var i;

*// Se todos os pontos foram verdes, é uma área livre.*

var b = true;

for (i = 0; i < cores.length; i++)

b = b && CompararCores(cores[i], DataLoader.COR\_AREA\_LIVRE);

if (b) return Fase.AREA\_LIVRE;

b = true;

for (i = 0; i < cores.length; i++)

b = b && CompararCores(cores[i], DataLoader.COR\_AREA\_LIVRE2);

if (b) return Fase.AREA\_LIVRE;

b = true;

for (i = 0; i < cores.length; i++)

b = b && CompararCores(cores[i], DataLoader.COR\_AREA\_LIVRE3);

if (b) return Fase.AREA\_LIVRE;

*// Se ao menos um for cinza, é uma pista.*

for (i = 0; i < cores.length; i++)

if (CompararCores(cores[i], DataLoader.COR\_PISTA))

return Fase.PISTA;

*// Se todas forem azul, é mar.*

b = true;

for (i = 0; i < cores.length; i++)

b = b && CompararCores(cores[i], DataLoader.COR\_MAR);

if (b) return Fase.MAR;

*// Area desconhecida / inutilizada.*

return 0;

}

# CONSTRUÇÕES

O motor k-engine permite a existência de dois tipos de construções, as construções decorativas e as funcionais. As construções decorativas geralmente possuem a finalidade de decorar a cidade mas não geram nenhuma contribuição direta para ela, porém elas podem ser usadas como construções especiais, como no caso do Castelo do Rei Kimera ou na Casa dos Guardiões. As construções funcionais possuem funções especificas a depender de seu tipo e categoria, e podem gerar diversas contribuições para a cidade como arrecadação de dinheiro, habitar e empregar a população, aumentar os índices de comércio, educação, segurança, saúde e infraestrutura.

## DECORATIVAS

As construções decorativas ficam localizadas na pasta "data/construcoes/decorativo" e cada construção possui 3 arquivos relacionados com ela, sendo um no formato XML que possui suas definições, um no formato FLA que é o arquivo fonte com a parte visual da construção e por último um arquivo no formato SWF que é o arquivo compilado a partir do FLA que será utilizado pelo motor.

Dentro do arquivo XML das construções decorativas existem as seguintes informações:

* Nome da construção;
* Descrição;
* Caminho para o arquivo gráfico;
* Tamanho em células de mapa que a construção ocupa;
* As áreas livres existentes dentro da área da construção;
* Tamanho da perspectiva.
* Função

Para utilizar uma construção decorativa com uma função especial no jogo, é necessário definir uma tag para a construção no registro “Função” do XML, após isso deve-se implementar a função especial dentro da classe “Game” na função “function MenuAtualizarConstrucao2(evt)” conforme mostrado no exemplo a seguir.

if(decorativa.GetFuncao() == "castelo"){

castelo\_mc.atualizar();

if(evt)

{

graficoClicado = evt.data.grafico;

*// Calcula a posição da janela de atualização para que fique exatamente sobre*

*// a construção.*

var pan2 = render.GetPan();

var posG2 = IsoMath.isoToScreen( new Pt(graficoClicado.x, graficoClicado.y) );

var pos2 = new Point(

stage.stageWidth/2 + (( Math.abs(pan2.x) - Math.abs(posG2.x))\*render.GetZoom()),

stage.stageHeight/2 - (( Math.abs(pan2.y) - Math.abs(posG2.y))\*render.GetZoom())

);

castelo\_mc.x = pos2.x;

castelo\_mc.y = pos2.y;

TweenLite.to(fade\_mc , 0.5, {autoAlpha: 1});

TweenLite.to(castelo\_mc , 0.5, {autoAlpha: 1});

}

return;

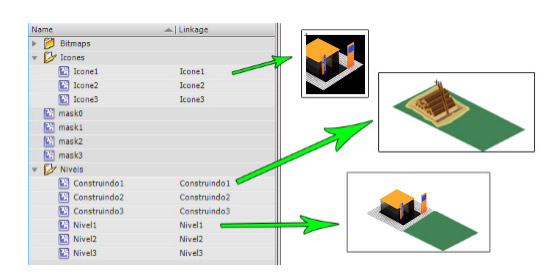
}

## FUNCIONAIS

As construções funcionais ficam localizadas na pasta "data/construcoes/funcional" e cada construção possui 3 arquivos relacionados com ela, sendo um no formato XML que possui suas definições, um no formato FLA que é o arquivo fonte com a parte visual da construção e por último um arquivo no formato SWF que é o arquivo compilado a partir do FLA que será utilizado pelo motor.

O arquivo FLA das construções funcionais é onde se encontra a representação gráfica da construção. Esse arquivo deve possuir os seguintes elementos em sua biblioteca para ser utilizado pelo motor:

* Ícone da construção.
  + Esse ícone será apresentado na tela de construções representando essa construção.
* Construção em processo de construção.
  + Essa imagem será usada enquanto a construção não estiver pronta.
* Construção finalizada.
  + Essa imagem será usada quando a construção estiver finalizada.



É importante que os objetos na biblioteca tenham o campo “Linkage” preenchido de forma correta pois será ele usado para o motor achar o objeto correto.

O arquivo XML das construções funcionais é separado em duas categorias, informações da construção e informações dos níveis da construção. Na categoria da construção existem as seguintes informações:

* Nome da construção;
* Categoria;
* Descrição;
* Caminho para o arquivo gráfico;
* Tamanho em células de mapa que a construção ocupa;
* As áreas livres existentes dentro da área da construção;
* Tamanho da perspectiva.

Na categoria dos níveis de construção existem as seguintes informações:

* Nível da construção;
* Nome do nível;
* Descrição do nível;
* Custo da construção;
* Tempo necessário para a construção estar disponível;
* Custo de demolição;
* Custo de manutenção;
* Tempo necessário para a construção ficar pronta;
* Tipo de contribuição que a construção gera;
* Quantidade da contribuição;
* Categoria de contribuição gerada;

# MÚSICAS E SONS

As músicas e sons utilizadas no jogo ficam localizadas na pasta “data/musicas” e devem ser salvas no formato MP3.

É necessário o carregamento das músicas e sons dentro do jogo antes de sua utilização:

gestorSom.CarregarSomArquivo('musica-menu-principal', 'data/musicas/menu\_inicial.mp3', 'musicas');

gestorSom.CarregarSomArquivo('musica-power-up', 'data/musicas/tema\_power\_up.mp3', 'musicas');

gestorSom.CarregarSomArquivo('musica-password', 'data/musicas/password.mp3', 'musicas');

Após isso é necessário a execução do mesmo:

gestorSom.Reproduzir('musica-carregando', 99999);

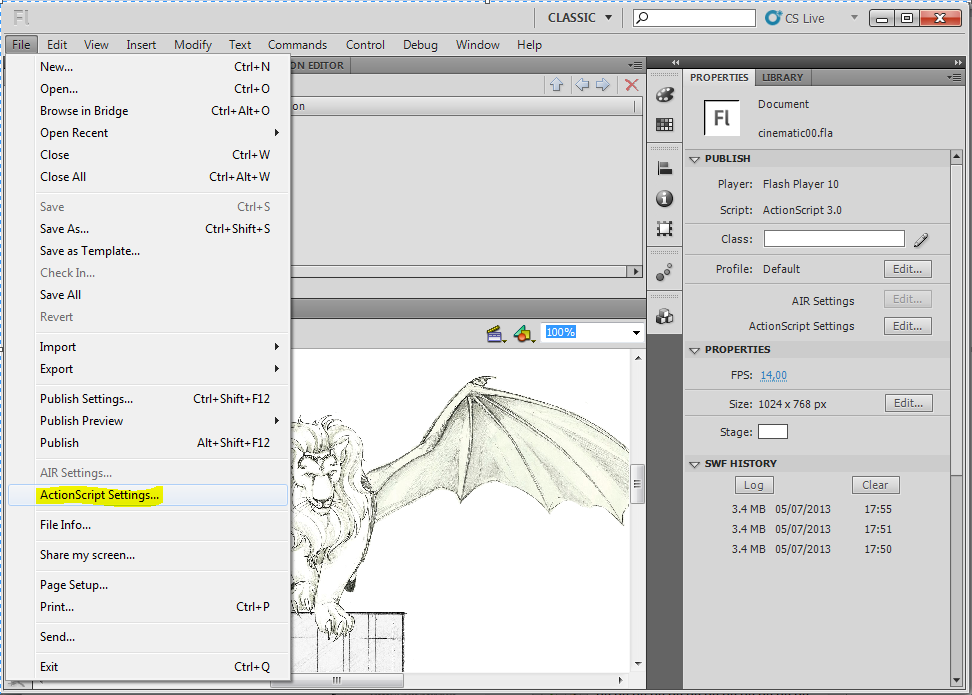
# CINEMÁTICAS

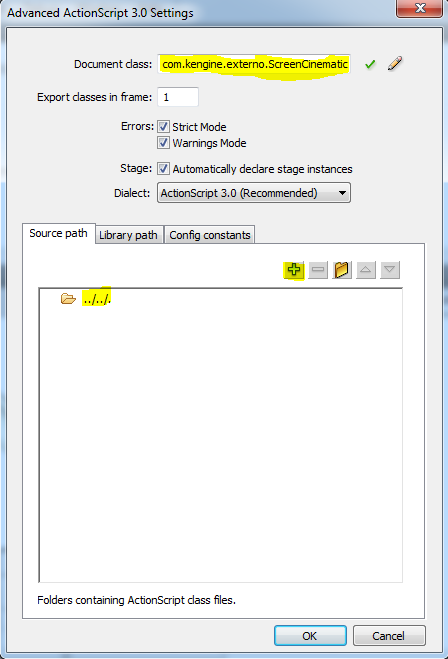
As cinemáticas e animações do jogo devem ser desenvolvidas dentro do software Flash Professional CS5 usando a resolução de tela do jogo 1024x768px e devem ser contidas dentro de um objeto de MovieClip.

As cinemáticas não possuem limite de duração e podem utilizar das diversas ferramentas de design gráfico e animação presentes no aplicativo, porém devem ser otimizadas para não usar imagens com tamanho grande ou exportadas sem otimização.

Os arquivos FLA das cinemáticas devem ser salvos na pasta “data/cinematicas” para no momento da compilação ter acesso as classes do motor e assim as funcionalidades necessárias para o seu funcionamento.

As cinemáticas devem ter como Document Class o valor “com.kengine.externo.ScreenCinematic” e devem definir o “Source Path” como “../../.” conforme mostrado em seguida pelas figuras.





No último frame da cinemática deve existir o seguinte código ActionScript que trata do encerramento da cinemática e carrega a tela seguinte:

stop();

this.finishCinematic();

Para carregar as cinemáticas dentro do motor é necessário o seguinte código:

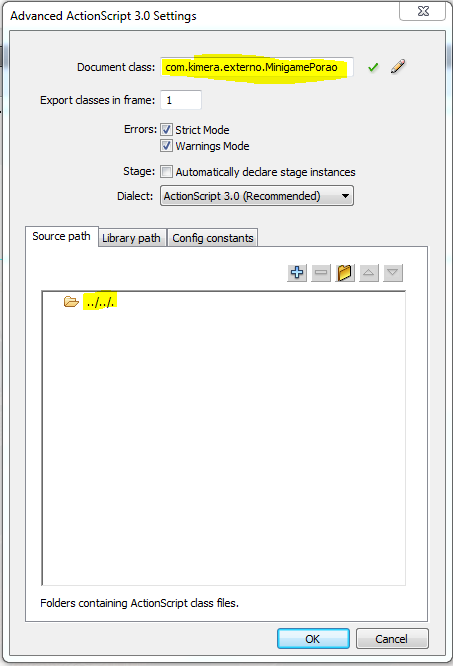
var abertura = new ScreenLoader("data/cinematicas/cinematic00.swf", endAbertura);

stage.addChild(abertura);

# MINI JOGOS

Os mini jogos apresentados dentro do motor K-engine devem ter sua programação em uma classe dentro do pacote de jogo, nesse caso na pasta “com/kimera/externo” e devem estender a classe “com.kengine.externo.Screen”.

O objeto gráfico do mini jogo deve ser desenvolvido dentro de um arquivo Flash separado que tenha como classe o arquivo previamente criado com a codificação do mini jogo conforme figura a seguir.



Para carregar o mini jogo dentro do motor deve se usar o seguinte código:

var minigame = new ScreenLoader("data/minigames/minigame\_00.swf", iniciarCineMundoKimera);

game.stageObj.addChild(minigame);

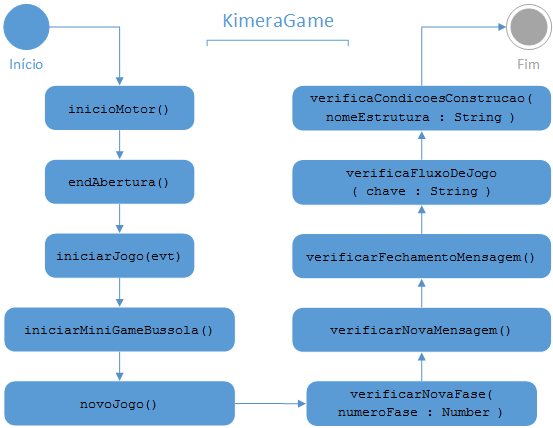
# FLUXO DO JOGO

O fluxo do jogo deve ser definido por uma classe que estenda a classe “com.kengine.Engine”, no caso do jogo Kimera a classe é chamada de “com.kimera .KimeraGame”.

Dentro dessa classe deve existem diversas funções já definidas que podem ser utilizadas para controlar o fluxo do jogo e da fase e que serão abordadas a seguir:

* inicioMotor() – Função chamada ao se iniciar o motor, utilizada para apresentar a abertura do jogo;
* verificarNovaFase(Number) – Função chamada ao se iniciar uma nova fase. É passado o número da fase que se iniciou;
* verificarNovaMensagem() – Função chamada ao se receber uma nova mensagem na fase;
* verificarFechamentoMensagem() – Função chamada ao fechar uma mensagem;
* verificarFluxoDeJogo(String) – Função chamada por diversas ações de fluxo dentro do jogo, como invocar Tílion, adicionar pedra mágica e escrever carta voadora. É passado a chave da ação acionada;
* verificarCondicoesConstrucao(String) – Função chamada ao se criar uma nova construção. É passado o nome da nova construção.

A seguir é apresentada uma imagem que demonstra de forma simplificada o fluxo de ações do jogo Kimera e outra que apresenta o Diagrama de Classe do KimeraGame.



Fluxo de execução simplificado da classe KimeraGame

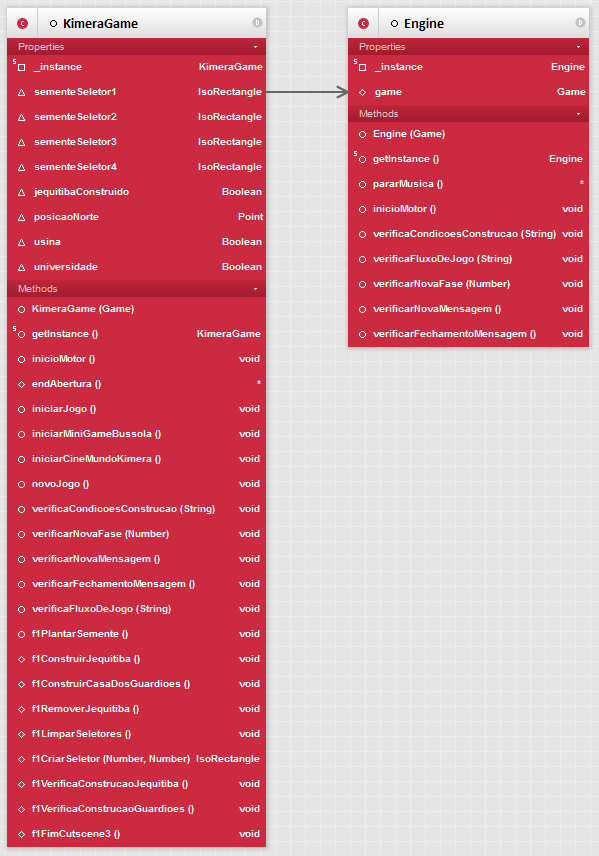


Diagrama de classe do KimeraGame