www.gameutil2d.org

[Versão JavaScript 1.0]

Criado por: Luciano Alves da Silva

Guia de Referência

[Maio 2018]

Introdução: Sobre o Framework GameUtil2D

O framework **GameUtil2D** é um conjunto de classes e métodos que permite a criação de jogos em 2D para Web usando o JavaScript (também existindo uma versão para a linguagem Java).

Características do Framework

O framework foi desenvolvido seguindo uma "filosofia": Permitir a criação de jogos de uma maneira simples, fácil e direta, sem a necessidade de criar grandes códigos para obter resultados significativos (além de possuir uma sintaxe clara e compreensiva).

Ele oferece diversas classes e métodos prontos para a criação de um jogo. Entre elas podemos citar classes para a criação de sprites (estáticos e animados), detecção de colisões, criação de personagens, detecção de dispositivos de entrada e saída (mouse e teclado), criação de cenários e etc.

Como esse framework surgiu?

Em 2012, Luciano Alves da Silva (na época professor de uma escola de desenvolvimento de jogos do Rio de Janeiro), estudava sobre os mais diversos frameworks para a criação de jogos (na época para a plataforma Android) e percebeu que a maioria das ferramentas eram complexas (e com carência de tutoriais), o que dificultava o aprendizado. A primeira versão do GameUtil2D desenvolvida surgiu para a plataforma Android, com uma série e classes e métodos facilitando a construção de jogos, eliminando a complexidade do desenvolvimento. Hoje esse framework encontra-se disponível somente para Java (Desktop Windows, Linux, Mac) e Javascript (Web e Mobile). Futuramente será disponível também em C# para o Visual Studio.

Capítulo 1: configuração do Framework para JavaScript

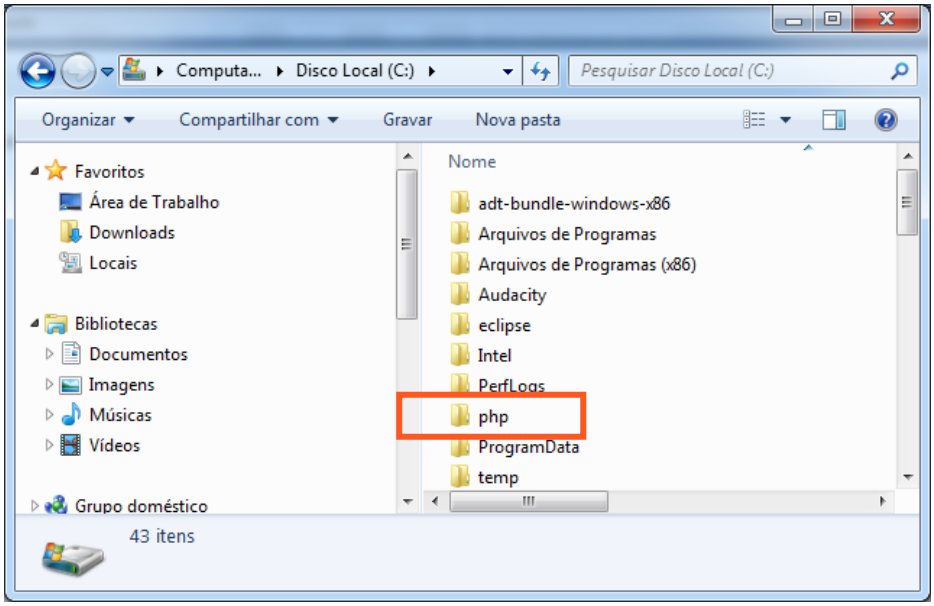
Essa documentação mostrará como instalar o GameUtil2D (Versão 1.0) para a JavaScript. O framework consiste em uma série de arquivos (sendo a maioria de extensão ".js") que permite a construção e execução do seu jogo em um navegador Web (que suporte o HTML 5). Navegadores mais antigos não dão suporte a execução do framework (como Internet Explorer 9.0 e anteriores). É aconselhável o uso de navegadores como Firefox e Chrome (e outros similares atualizados).

Instalação

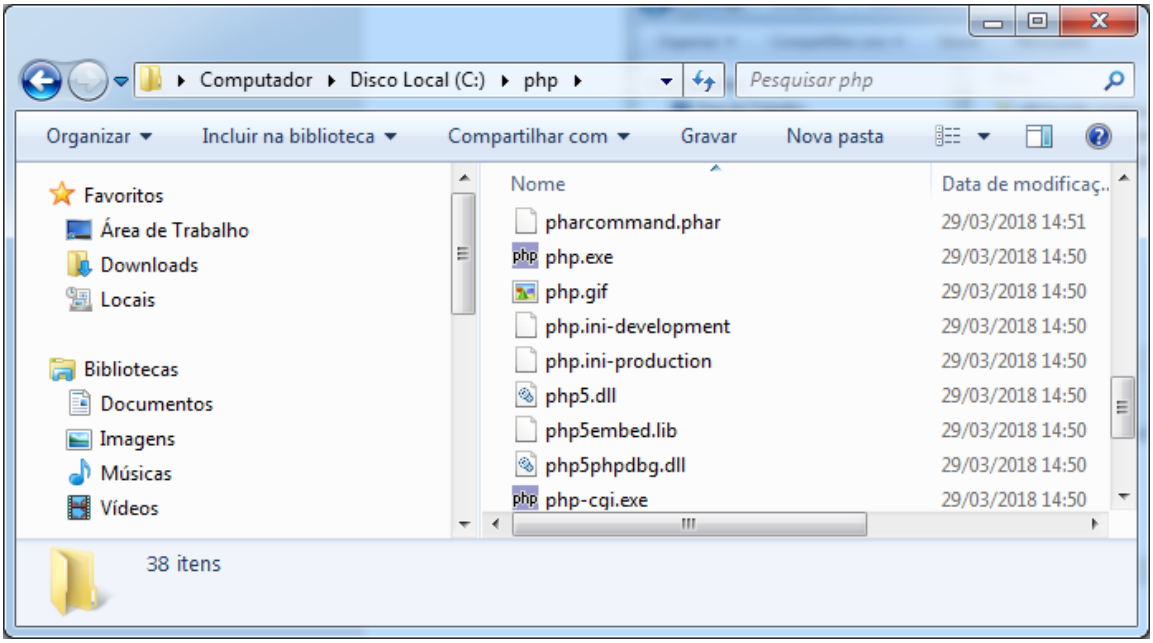
Para a construção e execução do seu jogo, é necessário o PHP instalado em sua máquina (caso contrário, o framework não funcionará).

Como instalar o PHP na máquina?

No site do gameutil2d.org, na seção de downloads você poderá encontrar baixar o PHP em sua máquina (que está no formato ".zip"). Após efetuar o download do PHP crie no diretório raiz "C:\" uma pasta chamada PHP:

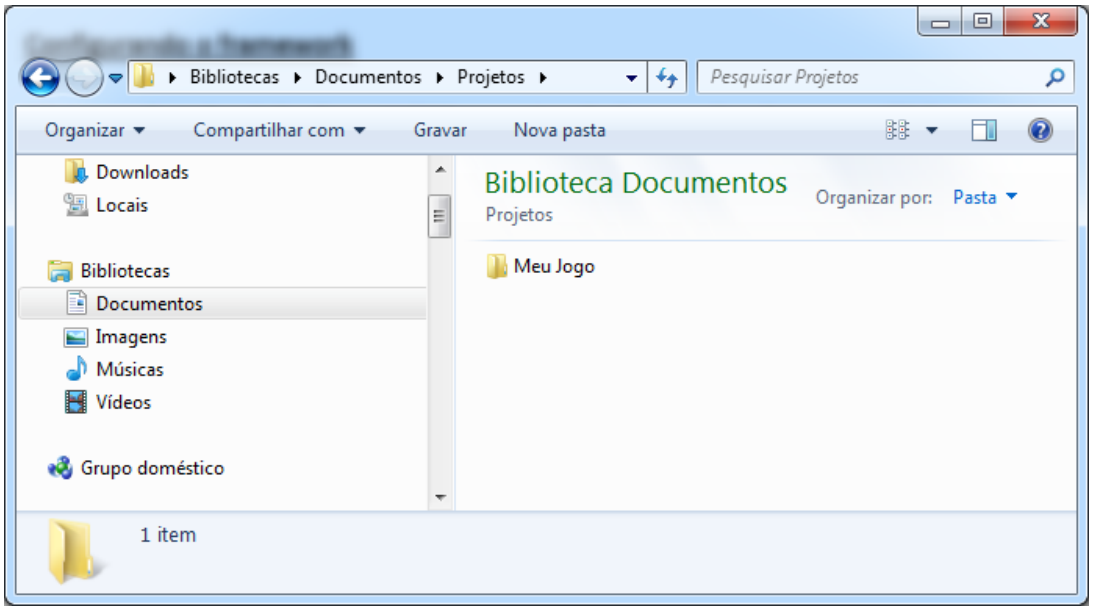


Extraia todo o conteúdo do arquivo “.zip” dentro da pasta PHP. O resultado após a extração deverá estar de acordo com a imagem a seguir:

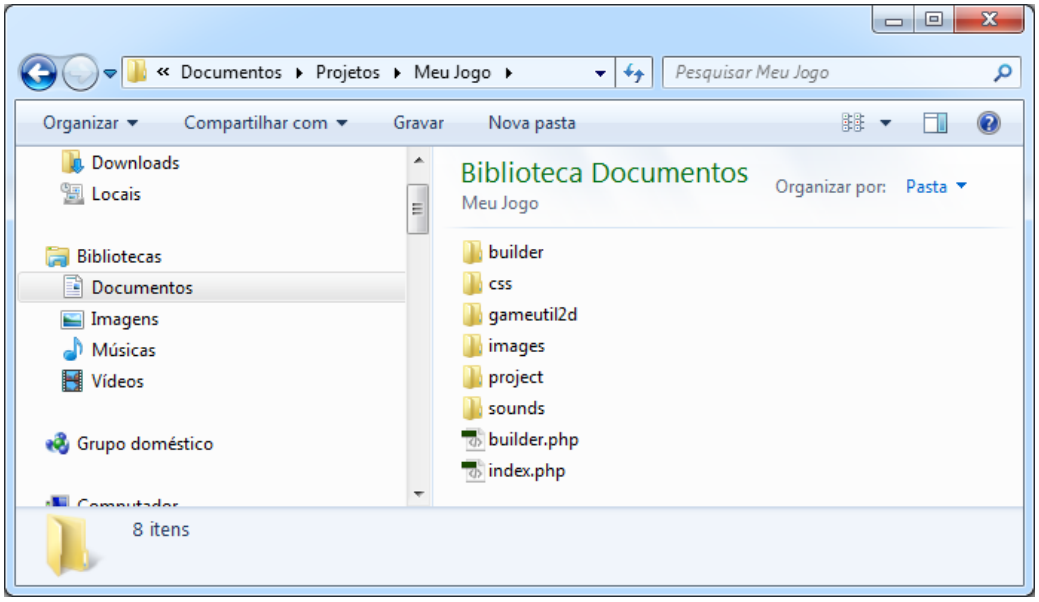


Configurando o framework

Efetue o download do GameUtil2D para JavaScript no site do gameutil2d.org. Crie uma pasta em um diretório desejado (Exemplo: “Meu Jogo”).



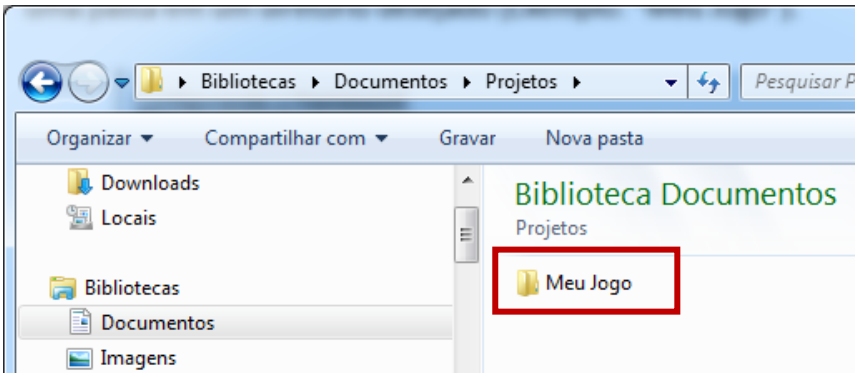
Extraia o arquivo do framework no diretório que você criou acima (como por exemplo “Exemplo”). Veja o resultado abaixo:



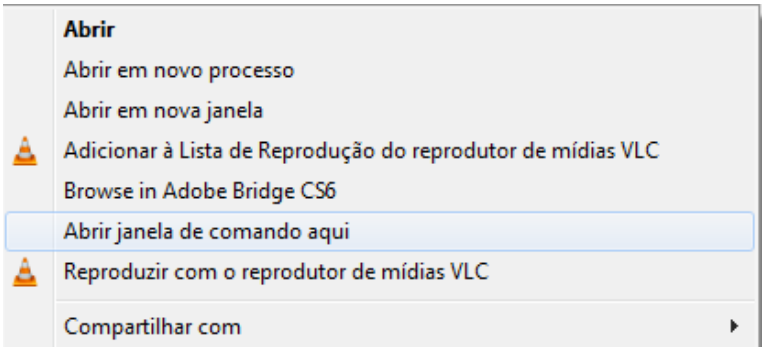
Executando com o PHP

Para a execução dos jogos construídos com o GameUtil2D será necessário o uso do PHP. Vamos aprender como configurar o PHP para executar o nosso jogo:

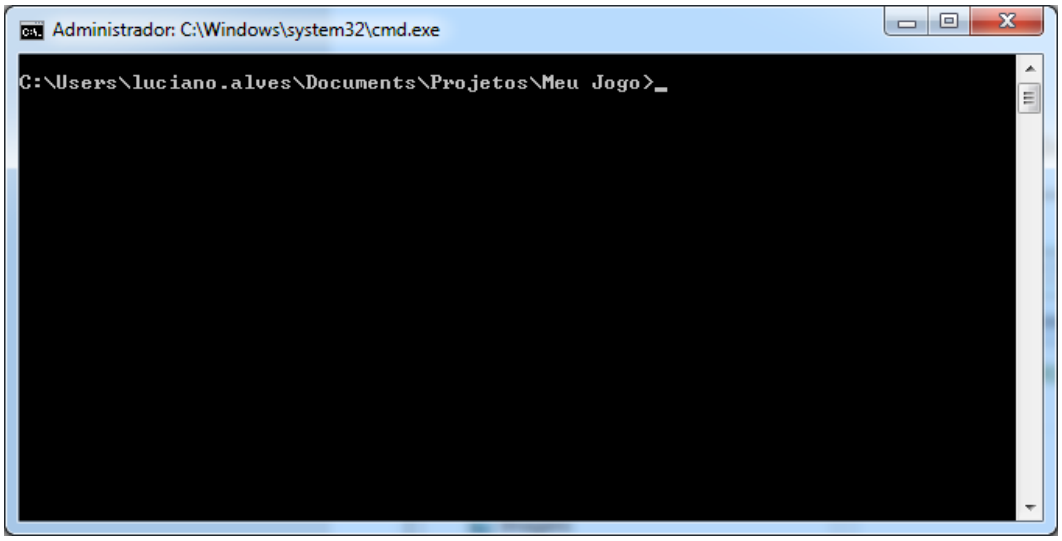
1 - Volte para o diretório que criamos (subir um nível)



2) Segure a tecla SHIFT e clique com o botão direito na pasta e selecione a opção “Abrir janela de comando aqui”



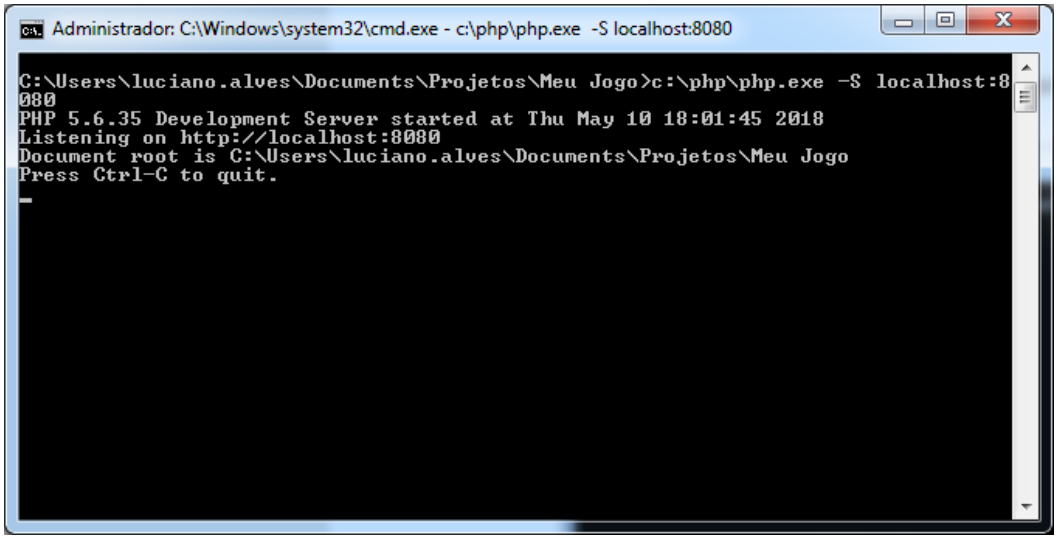
3) Deverá ser aberto o prompt de comando conforme mostra a figura abaixo:



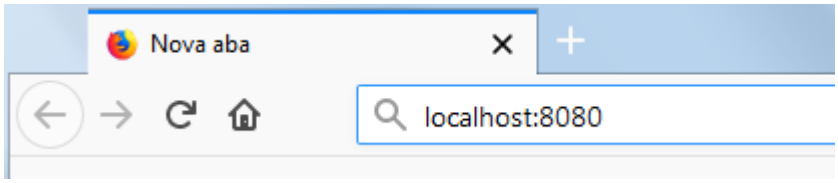
4) Agora vamos digitar o seguinte comando a seguir (sem os aspas)

c:\php\php.exe –S localhost:8080

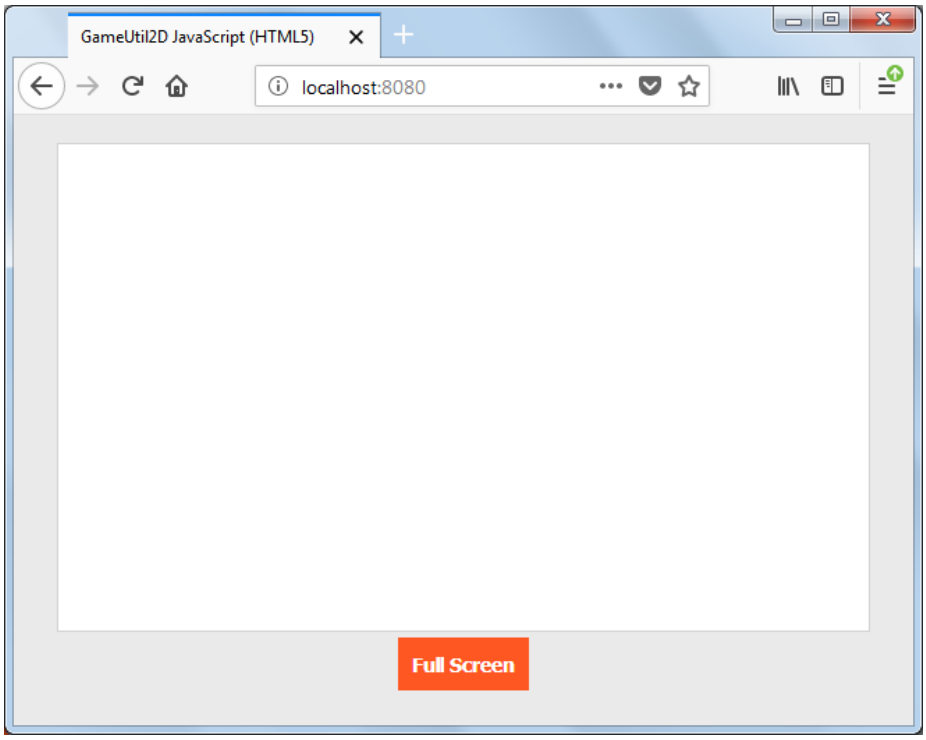
5) Após digitar o comando abaixo pressione [ENTER], e teremos o seguinte resultado:



6) Mantenha a janela do prompt aberta e abra o navegador de Internet . Feito digite o seguinte endereço:



7) Feito será aberta a seguinte tela abaixo:



Onde se encontra o arquivo que executa o jogo?

Toda a programação do jogo deve ser feita dentro do arquivo “Game.js”, presente na pasta “project”. No arquivo encontramos uma classe chamada “Game” com a seguinte estrutura:

```
function Game()
{

    this.Initialize = function()
    {

    }

    this.Update = function(keyboardState, mouseState, touchState)
    {

    }
    this.Draw = function(canvas)
    {

   }

}
```

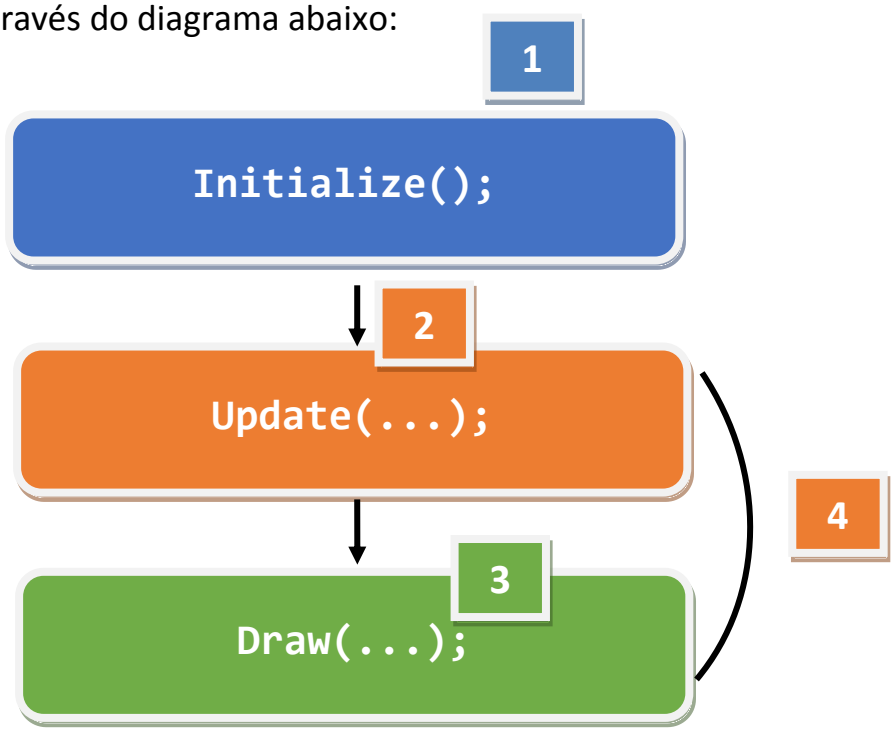
Descrição dos métodos adicionados

Vamos entender agora os métodos presentes classe **Game** do arquivo “Game.js”

Método	Descrição
`function Initialize()`	Esse método é executado quando o jogo é carregado na memória. Nesse método iniciamos todos os objetos (como imagens, sons e etc.)
`function Update(keyboardState, mouseState,touchState)`	Método responsável por executar a lógica do jogo (como movimentos, ações e etc.)
`function Draw(canvas)`	Método responsável por desenhar todos os elementos do jogo na tela.

Ciclo de execução os métodos do Jogo

Vamos entender agora o ciclo de execução dos métodos do GameUtil2D adicionamos em nossa classe através do diagrama abaixo:



1) O primeiro método à ser executado é o **Initialize,** responsável por carregar todos os objetos do jogo.

2) Depois de processado o método **Initialize**, será executado o método **Update** responsável por processar a lógica do jogo.

3) Depois de processado o método **Update**, será executado o método **Draw** responsável por desenhar na tela as atualizações das ações do jogo.

4) Depois de processado o método **Draw**, retornamos ao método **Update**, processamento novamente a lógica do jogo. E assim segue o ciclo **Draw** e **Update**.

Capítulo 2: Guia de referência – Classes e métodos

Nesse capítulo será mostrado um guia completo que mostra todas as classes do framework GameUtil2D para JavaScript. Vamos conferir em seguida:

2.1) A classe GameElement

Essa uma classe genérica, utilizada como classe base para a construção das outras classes presentes dentro desse deste framework. Vejamos os seus métodos na tabela abaixo:

Método	Descrição
`function SetX(value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`function SetY(value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`function SetWidth(value)`	Método responsável por definir a largura do elemento.
`function SetHeight(value)`	Método responsável por definir a altura do elemento.
`function SetBounds(x, y, width, height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`function MoveByX(value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro “value” for positivo, desloca o elemento para direita, caso contrário, move para esquerda.

Método	Descrição
`function MoveByY(value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`function SetTag(tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`function GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`function GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`function GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`function GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`function GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`function IsTouched(x, y)`	Método que retorna verdadeiro se o elemento foi clicado (ou tocado) na tela, baseado nas coordenadas X e Y onde ocorreu o clique (ou toque).
`function Draw(canvas)`	Método abstrato (implementado nas classes derivadas como Image, AnimationSprites e Character) responsável por desenhar o elemento na tela

2.2) A classe Images

A classe **Images** é responsável por exibir as imagens (sprites e texturas) na tela do seu jogo. Ela é derivada da classe **GameElement** (ou seja, possui todos os métodos da classe **GameElement**).

Vejamos na tabela abaixo seus métodos:

Método	Descrição
`function Images(image_name, x, y, width, height)`	Construtor da classe responsável por carregar a imagem na memória. No parâmetro do construtor informamos um argumento do tipo o nome da imagem, as coordenadas do elemento (parâmetros x e y), a largura (width) e a altura (height).

Método	Descrição
function SetX(value)	Método responsável por definir a posição do elemento na coordenada X da tela.
function SetY(value)	Método responsável por definir a posição do elemento na coordenada Y da tela.
function SetWidth(value)	Método responsável por definir a largura do elemento.
function SetHeight(value)	Método responsável por definir a altura do elemento.
function SetBounds(x, y, width, height)	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
function MoveByX(value)	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para direita, caso contrário, move para esquerda.
function MoveByY(value)	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
function SetTag(tag)	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
function GetTag()	Método responsável por retornar a "tag" definida para o elemento do jogo.
function GetX()	Método responsável por retornar a coordenada X do elemento na tela do jogo.
function GetY()	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
function GetWidth()	Método responsável por retornar largura do elemento do jogo.
function GetHeight()	Método responsável por retornar altura do elemento do jogo.
function IsTouched(x, y)	Método que retorna verdadeiro se o elemento foi clicado ou tocado na tela, baseado nas coordenadas X e Y onde ocorreu o clique (ou toque).
function Draw(canvas, *[flipEffect]*)	Método responsável por desenhar a imagem na tela, com efeito de Flip (inversão). Se o parâmetro "effect" (**que é opcional**) possuir o valor "FlipEffect.HORIZONTAL" (ou true), inverte a imagem na horizontal.

2.3) A classe AnimationSprites

A classe **AnimationSprites** é responsável por exibir uma animação de sprites dentro de um jogo. Ela é derivada da classe **GameElement**:

Vejamos na tabela abaixo seus métodos:

Método	Descrição
`function AnimationSprites(x, y, width, height)`	Construtor da classe responsável por carregar o objeto que ira exibir a animação. No parâmetro do construtor informamos as coordenadas da animação (parâmetros x e y), a largura (w) e a altura (y) das sprites que estarão no objeto.
`function Add(image_name)`	Método responsável por adicionar uma imagem dentro do objeto, que fará parte da animação de sprites.
`function Start(frames, loop)`	Método que inicia a animação de sprites. Nesse método definimos o intervalo de troca das imagens em frames (no parâmetro "frames"), e se a animação ocorrerá em loop (caso o parâmetro "loop" seja true) ou não (caso o parâmetro "loop" seja false)
`function Stop()`	Método que encerra a execução da animação.
`function Draw(canvas, [flipEffect])`	Método responsável por desenhar a animação na tela, com efeito de Flip (inversão). Se o parâmetro "effect" (que é opcional) possuir o valor "FlipEffect.HORIZONTAL" (ou true), inverte a animação na horizontal.
`function SetX(value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`function SetY(value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`function SetWidth(value)`	Método responsável por definir a largura do elemento.
`function SetHeight(value)`	Método responsável por definir a altura do elemento.
`function SetBounds(x, y, width, height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`function MoveByX(value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para direita, caso contrário, move para esquerda.

Método	Descrição
`function MoveByY(value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`function SetTag(tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`function GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`function GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`function GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`function GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`function GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`function IsTouched(x, y)`	Método que retorna verdadeiro se o elemento foi clicado ou tocado na tela, baseado nas coordenadas X e Y onde ocorreu o clique (ou toque).

2.4) A classe Collision

Essa classe é responsável por verificar a colisão entre objetos dentro da tela do jogo. Vejamos seus métodos abaixo:

Método	Descrição
`function Check(obj1, obj2)`	Método responsável verificar se houve uma colisão entre dois objetos. Retorna "true" quando ocorrer uma colisão, e falso caso contrário.
`function CheckInScene(obj, scene, any_object_with_tag)`	Método responsável por verificar se houve a colisão de um objeto com algum outro elemento qualquer dentro de uma cena (ver sobre a classe Scene mais para frente), que possua uma tag especificada no parâmetro (any_object_width_tag). O método retorna "true" se ocorreu uma colisão, e falso caso contrário.

Método	Descrição
`function CheckAndReturn (obj, scene, any_object_with_tag)`	Verifica se houve a colisão de um objeto com algum outro elemento qualquer dentro de uma cena que possua uma tag especificada no parâmetro (any_object_width_tag). Diferente do método "Check", esse retorna a instância do elemento que colidiu com o objeto (se uma colisão acontecer). Caso nenhuma colisão tenha acontecido, o método retorna **null**.

2.5) A classe Box

Essa classe (derivada da classe **GameElement**), possui somente uma única finalidade: definir pontos de colisão. Veja seus métodos abaixo:

Método	Descrição
`function(x, y, width, height,[tag])`	Construtor da classe responsável por carregar os pontos de colisão, onde informamos as coordenadas X e Y, como também a largura, altura e sua tag (que é opcional).

2.6) A classe Character

Com a classe **Character** podemos criar personagens voltados para jogos no estilo plataforma. A classe já possui sistema de física (pulo e queda), detecção de colisões e etc. Vejamos seus métodos abaixo:

Método	Descrição
`function (x, y, width, height)`	Construtor da classe responsável por carregar o objeto que irá exibir o personagem. No parâmetro do construtor informamos as coordenadas da animação (parâmetros x e y), a largura (width) e a altura (height) das sprites que estarão no objeto.
`function AddNewSpritesIdle(animation_name, image_names)`	Método responsável por adicionar as sprites que representam o estado de repouso de um personagem (quando o ele fica parado em movimento). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer a aquela animação.

Método	Descrição
function AddNewSpritesMove(animation_name, image_names)	Método responsável por adicionar as sprites que representam o movimento do personagem (que pode ser dele correndo ou andando). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
function AddNewSpritesAttack(animation_name, image_names)	Método responsável por adicionar as sprites que representam o ataque do personagem (como um soco, chute ou qualquer outro golpe). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
function AddNewSpritesJumping(animation_name, image_names)	Método responsável por adicionar as sprites que representam o pulo do personagem. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
function AddNewSpritesDamage(animation_name, image_names)	Método responsável por adicionar as sprites que representam o personagem sofrendo algum dano. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
function AddNewSpritesMove(animation_name, image_names)	Método responsável por adicionar as sprites que representam o personagem morrendo. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
function Idle(animation_name, frames, loop)	Executa a animação do personagem em estado de repouso (parado). Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
function MoveToRight(animation_name, frames, loop)	Executa a animação do personagem se movendo para a direita. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).

Método	Descrição
`function MoveToLeft(animation_name, frames, loop)`	Executa a animação do personagem se movendo para a esquerda. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`function Attack(animation_name, frames, loop)`	Executa a animação do personagem atacando. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`function Jump(animation_name, frames, loop, jump_shift)`	Executa a animação do personagem pulando. Nesse método devemos informar o nome da animação (animation_name), o intervalo da troca das sprites (em frames), se a animação ficará em loop (true) ou não (false) e o valor de deslocamento do pulo do personagem (jump_shift).
`function SufferDamage(animation_name, frames)`	Executa a animação do personagem sofrendo dano (além de mostrar o personagem piscando na tela). Nesse método devemos informar o nome da animação e o intervalo da troca das sprites (em frames).
`function Die(animation_name, frames)`	Executa a animação do personagem morrendo. Nesse método devemos informar o nome da animação e o intervalo da troca das sprites (em frames).
`function IsAttacking()`	Método que retorna verdadeiro se o personagem está atacando, caso contrário, retorna falso.
`function IsDamaged()`	Método que retorna verdadeiro se o personagem está sofrendo dano, caso contrário, retorna falso.
`function IsDying()`	Método que retorna verdadeiro se o personagem está morrendo (quando a animação dele morrendo está em execução), caso contrário, retorna falso.
`function IsDead()`	Método que retorna verdadeiro se o personagem está morto (quando a animação dele morrendo é encerrada), caso contrário, retorna falso.
`function IsGround()`	Método que retorna verdadeiro se o personagem está no chão, caso contrário, retorna falso.

Método	Descrição
function SetMaxFramesDamageDuration(frames)	Nesse método definimos em frames o tempo de duração do dano do personagem.
function CollisionBySide(scene)	Método que retorna verdadeiro se o personagem sofreu uma colisão pelos lados com objetos que são definidos dentro de um cenário.
function AddCollisionElementOfSideByTag(tag)	Nesse método definimos os elementos (objetos) de colisão, que o nosso personagem poderá se colidir durante seu movimento pelos lados. Os elementos de colisão são representados por "tags" (e que deverão ser definidos pelo método "SetTag") e deverão estar presentes dentro de um objeto do tipo Scene, onde o personagem estará adicionado.
function AddCollisionElementOfFallByTag(tag)	Nesse método definimos os elementos (objetos) de colisão, que o nosso personagem poderá se colidir durante uma queda após o pulo. Os elementos de colisão são representados por "tags" (e que deverão ser definidos pelo método "SetTag") e deverão estar presentes dentro de um objeto do tipo Scene, onde o personagem estará adicionado.
function Update(scene)	Método responsável por processar o pulo, a queda, as colisões, o ataque e toda a física do personagem.
function SetEnableJumpAndFall(jump_fall)	Quando o método Update está em uso, o mesmo processa o pulo e a queda do personagem. Esse método habilita ou desabilita o pulo e queda do personagem.
function TurnToRight()	Esse método vira nosso personagem para a direita (inverte as imagens), se o mesmo estiver em sentido contrário.
function TurnToLeft()	Esse método vira nosso personagem para a esquerda (inverte as imagens), se o mesmo estiver em sentido contrário.
function InvertSprites()	Inverte todas as sprites do personagem (realiza um efeito de flip na horizontal).
function SetX(int value)	Método responsável por definir a posição do elemento na coordenada X da tela.

Método	Descrição
function SetY(int value)	Método responsável por definir a posição do elemento na coordenada Y da tela.
function SetWidth(value)	Método responsável por definir a largura do elemento.
function SetHeight(value)	Método responsável por definir a altura do elemento.
function SetBounds(x, y, width, height)	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
function ShiftSceneElementsOnUpdate(can_shift)	Esse método permite habilitar o deslocamento de cenário quando o personagem pular (ao invés do personagem se deslocar), bastando definir o parâmetro *can_shift* como **true**.
function SetFixedElementsByTag(tag_names)	Usado em conjunto com o método *ShiftSceneElementsOnUpdate* define quais elementos NÃO SERÃO deslocados quando o personagem pular pela sua tag.
function MoveByX(value)	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo).
function MoveByY(value)	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo).
function SetTag(tag)	Método responsável por definir uma tag “rotulo” para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
function GetTag()	Método responsável por retornar a “tag” definida para o elemento do jogo.
function GetX()	Método responsável por retornar a coordenada X do elemento na tela do jogo.
function GetY()	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
function GetWidth()	Método responsável por retornar largura do elemento do jogo.
function GetHeight()	Método responsável por retornar altura do elemento do jogo.
function IsTouched(x, y)	Método que retorna verdadeiro se o elemento foi clicado ou tocado na tela, baseado nas coordenadas X e Y onde ocorreu o clique ou toque.

2.7) A classe Scene

Com a classe Scene podemos criar um cenário, onde estarão todos os elementos que serão visualizados no jogo. Seu uso é opcional, mas, quando for utilizar a classe Character para a construção de personagens, seu uso torna-se (teoricamente) obrigatório. Vejamos os seus métodos.

Método	Descrição
`function Scene()`	Construtor da classe Scene.
`function Add(element)`	Método que adiciona um elemento dentro do objeto scene.
`function Get(index)`	Método que retorna um objeto (do tipo GameElement) presente em um determinado índice (especificado no parâmetro index).
`function RemoveIndex(index)`	Remove um elemento do objeto scene, baseado em seu índice.
`function Remove(element)`	Remove um elemento do objeto scene, baseado em sua instância.
`function GetCount()`	Retorna o total de elementos dentro do objeto.
`function GetCountByType(type)`	Conta e retorna o total de elementos dentro do objeto de um determinado tipo de dados, informada no parâmetro “type”.
`function GetCountByTag(tag)`	Conta e retorna o total de elementos dentro do objeto que possuem uma determinada tag, informada no parâmetro “tag”.
`function Elements()`	Método que retorna um array, que contém todos os elementos presentes dentro do objeto scene.
`function MoveByX(value)`	Desloca todos os elementos dentro do objeto scene pela coordenada X da tela (partindo da posição de origem de cada um).
`function MoveByY(value)`	Desloca todos os elementos dentro do objeto scene pela coordenada Y da tela (partindo da posição de origem de cada um).

2.8) A classe TextDrawable

Com a classe **TextDrawable** podemos exibir uma informação na tela do jogo (ideal para a construção de uma HUD (Head-Up Display)). Vejamos seus métodos a seguir:

Método	Descrição
`function TextDrawable()`	Construtor da classe TextDrawable.
`function SetFontFamily(font_name)`	Definimos o nome da fonte a ser utilizada.
`function SetFontSize(size)`	Definimos o tamanho do texto a ser exibido na tela.
`function SetColor(color)`	Definimos a cor do texto a ser exibido na tela
`function SetX(value)`	Define a posição na coordenada X onde iremos exibir o texto.
`function SetY(value)`	Define a posição na coordenada Y onde iremos exibir o texto.
`function SetXY(x, y)`	Define a posição nas coordenadas X e Y onde iremos exibir o texto.
`function DrawString(canvas, text)`	Exibe seu texto na tela do jogo.
`function MoveByX(value)`	Desloca o texto pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro “value” for positivo, desloca o texto para direita, caso contrário, move para esquerda.
`function MoveByY(value)`	Desloca o texto pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro “value” for positivo, desloca o texto para baixo, caso contrário, move para cima.

2.9) A classe Song

Essa classe só possui uma única finalidade: Armazenar a música que será reproduzida (através da classe **MediaPlayer**) em um jogo. Vejamos o seu método.

Método	Descrição
`function Song(song_name)`	Construtor da classe, que armazena a música a ser reproduzida (que deve estar na pasta "sounds" do seu projeto e no formato "ogg").

2.5.2) A classe MediaPlayer

Essa classe é responsável por reproduzir as músicas (armazenadas dentro de uma instancia do tipo **Song**) dentro de um jogo. Vejamos seus métodos.

Método	Descrição
`function Play(song,loop)`	Reproduz a música armazenada dentro de uma instância do tipo Song. No parâmetro "loop" definimos se ela se repetirá (true) ou não (false).
`function Stop()`	Encerra uma música em execução.

2.5.3) A classe SoundEffect

Essa classe é responsável por reproduzir os sons de efeito dentro de um jogo. Vejamos seus métodos.

Método	Descrição
`function SoundEffect(sound_effect_name)`	Construtor da classe, onde informamos o nome do arquivo de som de efeito (sem extensão). Os arquivos de som devem estar dentro da pasta "sounds" e no formato ".ogg".
`function Play()`	Reproduz o som de efeito no jogo.