SliB.js

A Brazilian JavaScript Game Library

Documentation

Version 0.1 BETA

Logo designed by Fábio Manna

Portfolio: http://solucaodarte.portfoliobox.net

E-Mail: solucaodarte@gmail.com

Contents

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

○ Introduction

SliB.js is a ~~powerfull~~ JavaScript library for develop **HTML5** 2D games. I started your develop because i wanted job in something in free time... Sorry, I no have a beatifull story for you now. Wait I make a fake and beatifull story in next edition. I promisse, you go cry.

My name is **Luiz Felipe**, I am Brazilian, 19 years(currently) and an aspiring programmer. My english is very bad, sorry for any error.

In this documentation, you read a explication of use of the functions and how to make a game with **SliB.js**.

The functions and values of the library is organized in objects, the functions initiated with prefix "On", example Sys.OnError(), is personalizable runned in a event. You can define the values to a personalized function to configure events. Or define to **null** if is not more necessary.

Models and **Rooms** contain events runned in game play. This init with a upper-case char, example: Draw, Step1, Keyboard, OnMove(rooms not contain it, obviously), etc.

All this events can defineds to a personalized function. Or to **null** if you want "delete" a event.

The **Sys** object

The object **Sys** is a collection of values and functions referent of the system of the library.

Sys.supportCanvas – This contain the value true if the browser support the Canvas technology, or false if not support.

Sys.supportCookie – This contain the value true if the browser support and is enabled the cookies.

Sys.supportStorage – This contain the value true if the browser support the localStorage system.

Sys.supportAudio("format") – This return true if the sound format is supported by the browser, or false if not. Examples of formats: **mp3**, **ogg**, **wav**, etc. The format ogg is recommended for best compatibility.

Sys.OS – This contain the name of the Operational System of the computer. If unknown, return "None".

Sys.browser – This contain the name of the browser. If unknown, return "None".

Sys.toLoad – This contain the number of itens to load. Including images, sounds and scripts.

Sys.loaded – This contain the number of itens loaded.

Sys.load – This contain the percentage of the all itens is loaded.(0~100)

Sys.error – This contain the number of errors occurred in the execution of the game.

Sys.errorLoad – This contain the number of load errors occurred. Including images, sounds and scripts.

Sys.maskColor – Is the color of the mask of collision. The values aceptables is like CSS style. It is destined for debug of the game.

Sys.OnError() – This is a personalized function runned if a error occurs. The first argument is defined to the error, and if the function return true the game not stop independently the value of Game.stopOnError.

```
// EXAMPLE

Sys.OnError = function(error){

        console.error(error);

        if(error.indexOf("not defined") != -1) return true;

}
```

In this example, this show the error in the console. And if the error is a reference of a value not defined, continue the execution of the game.

Sys.OnLoadError() – This is a personalized function. This run if a load error occurs, the first argument is the object not loaded.

```
// EXAMPLE

Sys.OnLoadError = function(obj){

        if(obj instanceof Image) console.error("Error in load Image.");

        if(obj instanceof Audio) console.error("Error in load audio.");

        if(obj instanceof HTMLScriptElement) console.error("Error in
load script.");

}
```

This show a message in the console correspond the object not loaded. You can, for example, ignore the error in load sounds but stop the game if a image not load, running Game.stop()

Sys.loading(run, [text], [textColor], [font], [barColor], [bgColor]) – This init a loading bar with personalized colors and font.

run – Is a function, this is executed if the load finish.

text – Is the text apresented until not loaded. (Example: "loading...") This is apresented in the format: "text (100%)"

textColor – Is the color of the text, example: "#FA3333". The aceptable formats is like CSS formats.

font – Is the font used in text, the format is the size of the font sequently the name of the font. Example: "12px Arial"

barColor – The color of the bar progress, the color format is like the CSS formats.

bgColor – The color of the background of the area in the bar progress. You don't forget, right? Color format is like CSS formats. Example: "rgb(230, 50, 50)" or "#FA3333" or "rgba(230, 50, 50, 0.8)" etc.

Sys.drawError(text, [stop]) – **text** is the text showed in the console. If **stop** is true, stop the execution of the game.

Sys.isDefined(value) – This return true if the value if defined. This is teorically desnecessary because using if directly is the same result... And, well. Its is it.

Sys.isDeclared(value) – This return true if all variables is declared. **Value** is a string contain the name of the variables separeted by ",".

```
// EXAMPLE

if( Sys.isDeclared("aAnyVariable, aOtherVariable")  ) alert("All ok.");
```

Sys.readonly(value) – This define a variable to read only, **Value** is a string, example: "variable" or "obj.aValueOfTheObject".

Sys.repeat(number, execution) – This repeat a code **number** times. **Execution** is a function, the first argument is defined to the index of repetition. If the function return true, this break the loop.

```
// EXAMPLE

Sys.repeat(5, function(n){

        console.log("Hello World! " + n);

});
```

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

Sys.sleep(arg) – This stop the execution of the game. Two possibility to value of **arg**. You can stop the execution in **arg** milliseconds, defining this to a number. Or define **arg** to a function, and the game is stoped until the **arg** NOT return true.

```
// EXAMPLE

Sys.sleep(1000);  // This stop the game 1000 milliseconds(1 second)

Sys.sleep(function(){

        if(aValue == 10) return true;

}); // This stop the game until aValue is not equal 10
```

Sys.delayed(time, execution) – This execute a code before **time** milliseconds. **Execution** is a function executed before time. The values **Self** and **Other** is preserved to the value in moment of Sys.delayed() is executed.

Sys.withModel(model, execution, [otherValue]) – This run a code with all entities of the **model**. This define **Self** to actual entity of the model. **Execution** is a function, and **otherValue** is the value of Other in the execution code. If not defined, this define Other to actual value of Self.

```
// EXAMPLE

var hit = 10;

Sys.withModel(enemy, function(){

        Self.life -= Other.hit;

}); // All entities of enemy decrease in 10 the life.
```

Sys.include(script, [onload]) – This load a script file. **Script** is the path of the script, and **onload** is a optional function executed when the script is loaded. This return the HTML Script Element.

Sys.OnClose –This run when the game is closed. You can define to a string(a text apresented in close the game), or a function executed when the game is closed(perfect to save datas). The text returned by the function is showed in close.

The **Data** object

The object **Data** contain functions to get, post, save, load and more.

Data.GET – This is a object contain the values GET passed by the url. Example: "http://site.com/?username=Juão". This is acessible in Data.GET.username.

Data.http(url, type, [post], [execution]) – This send a HTTP request to a webpage. **Url** is the url of the page, **type** is the type of connection("GET" or "POST"), **post** is the data sended in POST mode, and **execution** is a function executed when the response of server is received.

Its possible use the arguments **Data.http(url, type, [execution])**, discarting post value if not necessary.

Data.save(name, value) – This save a value in local storage or cookie if not disponible/supported. **Name** is the name of the data, and **value** is the value.(obviously '-')

Data.load(name) – This return the value saved by Data.save(). If not value is saved with this name, return **null**.

Data.remove(name) – This delete a value saved by Data.save().

Data.setCookie(name, expireDays, value) – This set a value in the cookie. **Name** is the name of the value in cookie, **expireDays** is the days to auto-delete this cookie, and **value** is the value saved in cookie.

Data.getCookie(name) – This return a value in a cookie. If not value saved with this name, return a void string("").

Data.delCookie(name) – Delete a value in the cookie.

Data.compact(text) – This compact a text, and return it compacted.

Data.descompact(text) – This descompact a text compacted with Data.compact(), and return this.

Data.hash(text) – This return a number value representing a hash value of the **text**.

Data.crypt(text, password) – This function encrypt/decrypt the **text**, with **password**.

The **Key** object

The object **Key** contain values and functions referent the keyboard.

Key.lastKey – This is the code of the last key pressed.

Key.lastChar – This is the char of the last key pressed.

Key.string – A string containing chars presseds. If press backspace, delete last char.

Key.inDown(key) – This return true if the **key** is pressed. This is true in the momment the key is pressed, not until is down. **Key** is a char, or code of the key.

Key.isDown(key) – This return true if the **key** is down. **Key** is a char, or code of the key.

Key.inUp(key) – This return true when **key** is released.

Key.simulate(key, down) – This simulate the press or release of the key. **Key** is a char, or code of the key. If **down** is true, this simulate the press of key. If false, this simulate the release of the key.

This contains the value of various key codes. See the list:

Key.any – **Key.alt** – **Key.backspace** – **Key.ctrl** – **Key.enter** – **Key.esc** – **Key.shift** – **Key.tab** – **Key.win** (Windows key) – **Key.left** – **Key.up** – **Key.right** – **Key.down** – **Key.f1** – **Key.f2** – **Key.f3** – **Key.f4** – **Key.f5** – **Key.f6** – **Key.f7** – **Key.f8** – **Key.f9** – **Key.f10** – **Key.f11** – **Key.f12** – **Key.nb0** – **Key.nb1** – **Key.nb2** – **Key.nb3** – **Key.nb4** – **Key.nb5** – **Key.nb6** – **Key.nb7** – **Key.nb8** – **Key.nb9** – **Key.nbPlus** – **Key.nbMinus** – **Key.nbBar** – **Key.nbDot** – **Key.numlock** – **Key.scrollLock** – **Key.pause** – **Key.insert** – **Key.home** – **Key.pageUp** – **Key.del** – **Key.end** – **Key.pageDown**

The **Mouse** object

The object **Mouse** contains values and functions refferent the mouse.

Mouse.showContext – If true, show the context menu when press right button of mouse. The default value is false.

Mouse.x – This is the position X of the mouse in room.

Mouse.y – This is the position Y of the mouse in room.

Mouse.cursor – This is the cursor of the mouse in the area of the game.

The possible values is: "alias", "all-scroll", "auto", "cell", "context-menu", "col-resize", "copy", "crosshair", "default", "e-resize", "ew-resize", "grab", "grabbing", "help", "move", "n-resize", "ne-resize", "nesw-resize", "ns-resize", "nw-resize", "nwse-resize", "no-drop", "none", "not-allowed", "pointer", "progress", "row-resize", "s-resize", "se-resize", "sw-resize", "text", "vertical-text", "w-resize", "wait", "zoom-in", "zoom-out".

Also is possible define the cursor to a file. Using the format: "url(cursor.png)"

Mouse.inUp(button) – Return true when the button of the mouse is released.

Mouse.inDown(button) – Return true when the button of the mouse is pressed. This NOT return true until button is down, only return true when mouse button is pressed.

Mouse.isDown(button) – Return true if the mouse button is down.

Mouse.inWheel(direction) – Return true if the mouse scroll is in the **direction**. The possible values is: **Mouse.wheelUp**, **Mouse.wheelDown** and **Mouse.any**.

Mouse.inCircle(x, y, radius, button, [absolute]) – Return true if the **button** is down inside the circle area. If **absolute** is true, this verify the position of the mouse in window. If not, verify the position of the mouse in the room.

Mouse.inRectangle(x, y, width, height, button, [absolute]) – Return true if the **button** is down inside the rectangle area. If **absolute** is true, this verify the position of the mouse in window. If not, verify the position of the mouse in the room.

Mouse.simulate(button, down) – Simulate the press or release of the **button** of the mouse. If **down** is true, simulate the press of the button. If not, simulate the release of the button.

The possible values of button is: **Mouse.left**, **Mouse.right**, **Mouse.center** and **Mouse.any**.

The **Touch** object

This object contains values and functions referent the touch system in mobile devices. **SliB.js** simulate the first touch like the click of left button of mouse. And the drag of the first touch like the move of mouse.

The max of simultaneous touchs depend of the device.

Touch.support – This is true if the device support touch screen system.

Touch.number – Number of touchs simultaneous on the screen.

Touch.inDown(n) – This return true when touch of number **n** in the screen is started. The number of touch start with 0...(The first touch is 0, the second is 1....)

Touch.isDown(n) – This return true if the touch of number **n** is down.

Touch.inUp(n) – This return true when touch of number **n** is up.

Touch.inCircle(x, y, radius, [absolute]) – Return true if touch in the area of the circle. If **absolute** is true, verify the position of touch in screen. If not, verify the position of touch in the room.

Touch.inRectangle(x, y, width, height, [absolute]) – Return true if touch in the area of the rectangle. If **absolute** is true, verify the position of touch in screen. If not, verify the position of touch in the room.

Touch.get(n) – This return a object referent touch **n**. Containing the values: **x**, **y**, **width** and **height**. If the touch **n** is not down, return null.

```
// EXAMPLE

var t1 = Touch.get(0);

if(t1 == null) alert("Not touching.");

else alert("The position of the first touch is: " + t1.x + ", " + t1.y);
```

Touch.simulate(x, y, down) – This simulate the touch in the position **x** and **y**. If **down** is true, simulate the touch. If not, simulate the end of touch.

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

The **Game** object

The object **Game** contains functions and values referent the game. All the functions which desire a entity, is possible use a model. (The function use the first entity of the model)

Game.stopOnError – If defined to true, stop the if a error occurs. If false, not stop.

Game.fps – This is the fps(Frames Per Second) of the game.

Game.room – The actual room.

Game.inRoom(ent) – Return true if the entity **ent** exist in the actual room.

Game.inView(ent) – Return true if the entity **ent** is visible in the view.

Game.inGroup(ent, group) – Return true if the entity **ent** is in the **group**.

Game.entityRemove(ent) – Delete the entity **ent**.

Game.setCanvas(canvas) – This set the canvas utilized to draw the game. If **canvas** is a string, get the tag with the id **canvas**.

***Game.run()** – Start the execution of the game.

***Game.stop()** – Stop the execution of the game. Is possible use the Game.run() to continue the execution, but is possible the game not continue in some old navigators.

* Read only values.

The **Draw** object

Draw is a ~~big~~ collection of values and functions referent the draw of the game.

Draw.refresh – This is the interval of cycles to refresh the draw of the game. If define to -1, not auto refresh the draw (A cycle of the game is 5 milliseconds). The function Math.idealRefresh() return the value ideal referent the speed of the room.

Draw.colorFill – The color of the fill of the draw. The color is like CSS style.

Draw.colorBorder – The color of the border like CSS style.

Draw.alpha – The alpha of the draw, this is a value 0~1.

Draw.font – The font of the text to draw. This is the size of the text sequently the name of the font. Example: "12px Arial".

Draw.lineStyle – The style of the line. The possible values is: **butt**, **round** and **square**.

Draw.textAlign – The alignment of the text. The possible values is: **left**, **center** and **right**.

Draw.redraw() – Redraw the game. (including Draw events of the rooms and entities)

Draw.setTarget(canvas) – Set a canvas to draw. All draw functions and all draws of the system of the **SliB.js** is drawed in this canvas. Caution to not forget to reset the target.

Draw.resetTarget() – Reset the target to canvas of the game.

Draw.rectangle(x, y, width, height, [outline]) – Draw a rectangle in the position **x** and **y**, with **width** and **height**. If **outline** is true, draw only the border line. If not, draw a fill rectangle.

Draw.point(x, y, [size]) – Draw a point in position **x** and **y**. Size is the size of the point.(Really? Yeah!)

Draw.circle(x, y, radius, [outline], [angleEnd], [angleStart], [closed])

– Draw a circle. If **outline** is true, draw only the border line. **angleEnd** is the final angle of the (semi-)circle, **angleStart** is the initial angle. If **closed** is true, draw two lines closing the circle in "pacman" format.

Draw.line(x1, y1, x2, y2, [size]) – Draw a line from **x1** and **y1**, to **x2** and **y2**. Size is the size of the line.

Draw.radius(x, y, length, angle, [size]) – Draw a radius in position **x** and **y**, to direction of **angle**. Size...You know.

Draw.curve(x1, y1, x2, y2, x3, y3, [size]) – Draw a line with bezier curve, from **x1** and **y1**, to **x3** and **y3**. **x2** and **y2** is a point to curve the line.

Draw.text(text, x, y, [color], [width]) – Draw the **text** in position **x** and **y**. **color** is the color of the text. The value is like CSS style. **width** is the width of the text, perfect for stretch a text.

Draw.sprite(sprite, imageid, x, y) – Draw a subimage of the **sprite** in position **x** and **y**. **imageid** is the index of the subimage. The values of transformations of the sprite is applied to the image drawed.

Draw.image(image, x, y, [xoffset], [yoffset], [xscale, yscale], [angle])

– Draw a image in position **x** and **y**. **xoffset** and **yoffset** is the point of the image drawed in x and y. **xscale** and **yscale** is the scale of the image, the normal scale is 1. **angle** is the angle of the image.

* The possible values of **image** is a image element, canvas element or a video element.

Draw.imageRepeat(image, x, y) – Draw a image repeating for fill all the canvas.

Draw.mask(ent, [simple]) – Draw the mask of the entity. **ent** is the entity. If **simple** is true, draw the simple mask of collision. If not, draw the advanced.

Draw.clear(x, y, width, height) – Clear a rectangle area in position **x** and **y**. You can use Draw.clear(All) to clear all the canvas.

Draw.projection(x, y, xscale, yscale, angle, px, py, pwidth, pheight)

– Draw a projection of a part of the room. **x** and **y** is the position to draw. **xscale** and **yscale** is the scale of the projection draw(1 is normal, 0.5 is the half, etc.). **angle** is the angle of the projection draw. **px** and **py** is the position to get the projection in the room. **pwidth** and **pheight** is the length width and height to get the projection in the room.

The **Col** object

This object contains functions referent the system of collision. Containing a object "simple" with the functions referent the simple collision.

Now, the "advanced" collision is a bit slowly, and incompatible with old browsers. The **SliB.js** use a substitute function to run the system in old browsers, but this is very slowly.(**SliB.js** show a alert in the console about this in old browsers, and set the value of Col.compatible to false)

* In all functions you can use models in local of entities. The function use the first entity of the model.

* All functions define the value of **Other** to the entity or tileset collided with the first entity. If not collide, set **Other** to null.

Col.compatible – This is true if the advanced system of collision is supportable. If false, the system work but slowly… Or with bugs, in some browsers. I recommend not use this system if this value is false. (but, you can use the simple collision)

Col.compatibility() – This set all the functions of advanced collision to simple collision. Use it for compatibility for browsers which not support advanced collision. (remember set the simple mask of all the objects.)

Col.entity(ent1, ent2) – Return true if entity **ent1** is colliding with entity **ent2**.

Col.rectangle(x, y, width, height, ent) – Verify if the entity **ent** collide with rectangle in position **x** and **y**. (No, this not define Other to the rectangle)

Col.model(ent, model) – Return true if the entity **ent** is colliding with any entity of the **model**.

Col.tileset(ent, tileset) – Return true if entity **ent** is colliding with same part of the **tileset**. Define null to the part of the tileset. This is a object with the values: **tileset**(the tileset of this part), **tilex** and **tiley**(the position in the image of the tileset), **x** and **y**(position in the room).

* All this values is possible to edit. For example, is possible edit **x** and **y** to move it.

Col.group(ent, group) – Return true if entity **ent** is colliding with same entity or tileset member of the **group**. If exists members entities and tilesets in this group, is possible verify if Other is defined to a entity or a part of tileset. Example:

```
if( Col.group(player, 2) ){
        if(Other instanceof Entity) alert("Colliding with an entity.");
        else alert("Colliding with a part of one tileset.");
}
```

Col.onPosition(x, y, ent, obj) – This verify the collision of entity **ent** with **obj**, but verify the collision like the entity **ent** is positioned in position **x** and **y**. **obj** is possible defined to a entity, model, tileset or group.

* Any value which not is a entity, model or tileset is considered a group.

Col.point(x, y, ent) – Return true if the point **x** and **y** is colliding with the entity **ent**.

The sub-object **simple**.

This is not the end, now is the functions to simple collision…

But, its is simple. Is the same functions as "advanced" collision, but using the sub-object simple.

Example: Col.simple.entity(), Col.simple.model(), Col.simple.point(), etc. (the arguments is identic of the "advanced" version)

And exist a additional function in simple collision:

Col.simple.particle(ent, particle) – Return true if entity **ent** is colliding with a particle of **particle**.

The **VStyles** object

This contains the styles to use in virtual keys.(the styles is objects containing values of the apearance of the virtual key)

To edit or create a new style, the values in the object is:

mode	– The type of the virtual key. The values is: "circle" or "rectangle"
colorBorder	– The color of the border.(color is like CSS style)
color	– The color of the virtual key.

colorOnClick – The color of the virtual key until clicked.(or touched)

alpha – The alpha of the virtual key. The value is from 0 at 1.

alphaOnClick – The alpha value until the virtual key is clicked.(or touched)

font – The font of the text in virtual key. This is the size of the text sequently the font name. Example: "12px Arial".

textColor – The color of the text.

See the list of styles defaults:

VStyles.circle - A circular button style.

The constructor **VKey**

new VKey([x], [y], [button], [style])

This create a new virtual key.

x and **y** is the position of the virtual key in the screen.(not in the room)

button is the key to simulate a press. Or a function to execute on click.(or touch)

style is a object with values of the style of the virtual key. (See about VStyles object)

The values of the object created by VKey is:

x and **y** – The position in the screen.

button – A key to simulate on click, or a function to execute on click.

style – The style of the virtual key.

text – The text on the virtual key.

width, **height** – The length width and height of the virtual key.(if rectangular style)

radius – The radius of the circle of the virtual key.(if circular style)

actived– If true, this button is usual and possible to click.

visible – If true, this is drawed in the screen. If not, is possible click but is invisible.

clicked – True if it is clicked.

touched – True if it is touched. Is same value of Vk.clicked, is some for visual of the code.

delete() – Delete this virtual key.

The constructor **Mask**

new Mask(width, height);

This create a new mask of collision.

w**idth** and **height** is the length of the mask.

The values of the object created by Mask is:

beginPoint() – Init the points of the advanced mask.

endPoint() – End the points of the advanced mask.

points(x1, y1, x2, y2, ...) – Draw points in the advanced mask.

rectangle(x, y, width, height) – Draw a rectangle in the advanced mask.

circle(x, y, radius) – Draw a circle in the advanced mask.

simple(x, y, width, height) – Set values of the simple mask.

The constructor **Room**

new Room()

Create a new room.

The values of the object created by Room is:

Background – The background of the room.

bgColor – The color of the background of the room.

paused – If true, pause all the room.

persistent – If true, when alter the room all the values of this room is preserved.(including entities, tilesets, particles, etc.)

Start – This is a function executed when the room is started.

Step1 – This is a function executed in the init of step of the game.

Step2 – This is a function executed in the end of step of the game.

InPause – This function is executed in the step if the room is paused.

Draw – This function is executed in the draw of the game.

speed – The speed of the room, this is the number of steps per second.

time – The number of steps executed after the start of room.(if persistent, this value is preserved too)

view – The view of the room. The values of this object is:

view.x and **view.y** – The position in the room of the view.

view.follow – A entity for view follow, if set for a model this follow the first entity of this model.

wmax and **hmax** – The max length after center to follow the entity.

xmin, **ymin**, **xmax** and **ymax** – The position min and max for view.

setViewLimit(xmin, ymin, xmax, ymax) – Set the position min and max of the view.

setFollowLimit(wmax, hmax) – Set the max length to follow the entity.

entity(model, x, y) – Add a new entity for view.

addTile(tileset, tilex, tiley, x, y) – Add a part of **tileset** for room. **tilex** and **tiley** is the position in the tileset image. **x** and **y** is the position in the room.

removeTile(x, y, all) – Remove a tileset in position **x** and **y**. If **all** is true, remove all tilesets in this position. If you use **rm.removeTile(All)**, remove all tiles in the room.

The constructor **Background**

new Background(image, [repeat])

Create a new background.

image – The image file of the background.

repeat – If true, repeat the background for fill the canvas.

The values of the object created by Background is:

image	– The image file of the background.
x and **y**	– The position of the background in room.
xscale and **yscale**	– The scale of the image of the background.
repeat	– If true, repeat the background for fill the canvas.

xspeed and **yspeed** – The speed of the background, this move the background in the step of the game.

The constructor **Tileset**

new Tileset(image, twidth, theight)

Create a new tileset.

image – The image file of the tileset.

twidth and **theight** – The length of a tile.

Author: Luiz Felipe (Superbomber / DieBoy)
E-Mail: felipe.silva337@yahoo.com
Date: 13/11/2016

The values of the object created by Tileset is:

twidth and **ts.theight** – The length of a tile.

group – The groups of the tileset. Is possible define to a value(1, "wall", etc.) or a array containing values. See about Game.inGroup()

image – The element image of the tileset.

The constructor **Part**

new Part([image], [xoffset], [yoffset], [timelife], [gravity])

Create a new particle.

image – The image file of the particle.

xoffset and **yoffset** – The initial point of the particle.

timelife – The number of steps of the particle's live.

gravity – The gravity force to apply in the particle.

The values of the object created by Part is:

image – The image file of the particle.

xoffset and **prt.yoffset** – The initial point of the particle.

timelife – The number of steps of the particle's live.

gravity – The gravity force to apply in the particle.

xscale and **prt.yscale** – The scale of the particle.

xscaleTransform and **prt.yscaleTransform** – The tranformation value of the scale. (edited by step)

alphaTransform – The transformation value of the alpha. (edited by step)

angleTransform – The transformation value of the angle. (edited by step)

setScale(xscale, yscale) – Define the scale of the particle.

setTransform(xscale, yscale, alpha, angle) – Define the tranformations values.

prt.emit(x, y, [force], [direction]) – Emit a new particle. **x** and **y** is the position in the room. **force** is a force to apply in the particle. **direction** is the direction of the particle move with the force.

* The values **part**(the particle object), **time**(time left of life), **gravity**, **xscale**, **yscale**, **alpha**, **force**, **direction**, **x**, and **y** exist in the object of particle emited. See about Col.simple.particle(), this define the value of null for this object.

The constructor **Sprite**

new Sprite(images, xoffset, yoffset)

Create a new sprite.

images – The images of the sprite. You can use the format: "img1.png;img2.png;...". Or use a array.

xoffset and **yoffset** – The initial position of the sprite.

The values of the object created by Sprite is:

images – A array containing the images of the sprite.

image – The actual subimage.

xoffset and **yoffset** – The initial point of the sprite.

xscale and **yscale** – The scale of the sprite.

angle – The angle of the sprite.

subimage – The number of the sub-image.

The constructor **Model**

new Model([sprite])

Create a new model.

sprite – The sprite of the model. If not defined, create a new void sprite for the model.

The values of the object created by Model is:

sprite – The sprite of the model.

mask – The mask of collision.

angle – The angle of movement.

speed – The speed of movement.

persistent – If true, the entities of this model is global. Exist in all the room.

parent – The parent model of this model. In collision functions or Sys.withModel() destined for the parent model, too run with this model entities.

group – The groups of the model. Is possible define to a value(1, "wall", etc.) or a array containing values. See about Game.inGroup()

visible – If false, this draw your sprite.

layer – The layer of the entities of model. Is possible edit this in the entity in game execution.

paused – If true, not run events(Step, Keyboard, etc.) of the entities of this object. But run Draw event and InPause.

Create – A function executed in create a new entity.

Step1 – A function executed in the init of the game step.

Step2 – A function executed in the end of the game step.

Draw – A function executed in the draw of the game.

Mouse – A function executed in the step if the mouse move or click.

Keyboard – A function executed in the step if pressed or released a key.

InPause - A function executed in the step if the entity is paused.

OnMove(dir, dif) – A function executed when the position x or y of the entity is modified. The argument passed for function **dir** is the axis vector modified("x" or "y"), **dif** is the diference of the previous position.

* Caution for not loop this editing the position on this. If you want move the entity NOT running this event, use the values __x__ and __y__ of the entity.(And this not edit the value of xprevious and yprevious)

getEntity(i) – Return the entity of id **I** of this model.

The constructor **Entity**

new Entity(model, x, y)

Create a new entity in the actual room.

model – The model of the entity.

x and **y** – The position in the room.

All the values of the model is copied to the entity(except getEntity(), obviously). But this contains others new values:

model – The model of the entity.

x and **y** – The position in the room.

xprevious and **yprevious** – The previous position in the room.

remove() – Remove the entity.

Global values

All – A special object used in functions.

Self – A special object defined in event functions to the entity or room. And in Sys.withModel() defining this to the actual entity of the model.

Other – A special object defined in functions of collision to the entity, tileset or particle. And in Sys.withModel() to the actual value of Self, or to a personalized value.

global – Same as window. Destined to create global values.

createCanvas([width], [height]) – Create and return a new canvas element with length **width** and **height**. This define the value of context to the context of the canvas.

createSound(src) – Create and return a new audio element. **src** is the file of audio.

Math implemetations

This is the mathematicals functions implemented by **SliB.js**

Math.roundGrid(value, grid) – Return the **value** rounded to a **grid** value.

Math.idealRefresh() – Return the ideal value to Draw.refresh referent the room speed.

Math.idealSpeed() – Return the ideal value to speed of the room referent the value of Draw.refresh.

Math.rad2deg(value) – Return the **value** converted from radians to degrees.

Math.deg2rad(value) – Return the **value** converted from degrees to radians.

Math.distance(x1, y1, x2, y2) – Return the distance from point (**x1**, **y1**) to point (**x2**, **y2**)

Math.angle(x1, y1, x2, y2) – Return the angle from point (**x1**, **y1**) to point (**x2**, **y2**)

Math.radiusGetPoint(radius, angle) – Return a object containing the values **x** and **y** of the point, obteined by calculate the **radius** and **angle** to get the vector in this indication.

Math.randombet(min, max) – Return a value between **min** and **max**.

Math.choose(v1, v2, v3, ...) – Return one of the values passed by argument randomly.

Math.seed – The actual seed.

Math.rseed() – Set the seed to a random value.

Math.srandom(v1, [v2]) – Return a random value generated using the seed. If not define **v2**, return a random value between 0 and **v1**. If define, return a random value between **v1** and **v2**.