



AppsGlobal.net

Presents



version 1.0

Disclaimer

Any software or source code acquired through AppsGlobal.net is provided on an 'as is' basis. AppsGlobal.net shall have no third party liability nor liability to you with respect to its obligations under this agreement or otherwise, including without limitation liability for direct, compensatory, consequential, exemplary, special, incidental or punitive damages (including, damages for data loss), even if it has been advised of the possibility of such damages. The described limitation applies to all causes of action.

You agree to indemnify and hold AppsGlobal.net harmless from any and all liability, loss, trading losses, costs, damage, or expense, including attorneys' fees you or third party may suffer (other than damage to your tangible property or injuries to employees occurring during the course of work) as a result of claims, demands, costs, or judgments arising out of third party and any other claims based on the performance of any software or any source code, including, but not limited to losses resulting from the use of AppsGlobal.net developed software, programs, web sites, source code or tutorials.

SpineHelper.net and AppsGlobal.net are independent sites not officially affiliated with Corona Labs, Inc. nor Esoteric Software (Spine). The Corona SDK name and Corona SDK logo are properties of CoronaLabs, Inc. These websites and the products and services offered are not associated, affiliated, endorsed, or sponsored by CoronaLabs, Inc. (CoronaSDK) nor Esoteric Software (Spine), nor have they been reviewed tested by them.

License

1. You MAY use anything you find in SpineHelper to:

- * make applications for free or for profit (\$).
- * make games for free or for profit (\$).

2. You MAY NOT:

- * sell or distribute SpineHelper or the sampler as your own work
- * sell or distribute SpineHelper as part of a book, starter kit, etc.

3. It would be nice if everyone who uses SpineHelper were to give me credit, but I understand that may not always be possible. So, if you can please give a shout out like: "Made with SpineHelper, by AppsGlobal.net", I would really appreciate it. Also, if you want to link back to my site that would be awesome too: <http://www.appsglobal.net/>

Special Thanks

This module wouldn't have been possible without the creation of the "matrix.lua" library (LuaMatrix) by Michael Lutz (chillcode) - original author and David Manura (maintener) <https://github.com/davidm/luas-matrix>

Introduction

Thank you for giving SpineHelper a try in your CoronaSDK projects. I created this module to help me in my own projects, but after speaking with a couple of other Corona developers that also were trying to do the same as me, I decided to add additional features and convert it into a module so other developers could benefit from it.

When I started to write the code for SpineHelper little did I know all the different things that I had to learn, it took me around four months to finish the module. I hope my module helps you create awesome apps and games, so please let me know when you publish one with SpineHelper.

Approach Taken

As with any other technical project, there are several ways of achieving the same result, so the approach I took in the implementation was the use of SpriteSheets created with TexturePacker, so in order to make SpineHelper work, you are going to have to buy a copy of the awesome TexturePacker, you can get a copy from here: <https://www.codeandweb.com/texturepacker> I think it is a great investment if you haven't already acquired a license of Texture Packer, it is a piece of software that you will use over and over again.

Also, because I was one of the original backers of the Spine Kickstarter project, I was very lucky to get a Professional License, so all the methods that you will find in SpineHelper are designed to work with the Pro Version which currently costs \$289 USD. In theory, the SpineHelper basic methods should work fine with the Essential version but I haven't tested them. In any case, should you have any problems feel free to contact me via email at hector@appsglobal.net. I think that the Pro Version of Spine is well worth the price given all the features it has, so like in the case of TexturePacker, I strongly recommend you invest in it.

Also, there is another module that is critical for the use of SpineHelper, which is "matrix.lua", this module is essential in the implementation as it is the module in charge of handling the matrices that Spine needs in order to work properly. Just make sure you include a copy of this module in the root folder of your project.

Finally, no code is ever finished and it is very likely that my module has bugs, so if you have a feature you want me to include or a bug you want me to fix, please send me an email and I'll do my best to include it or fix it as soon as possible (I still have a day job, so please be patient if I don't fix it immediately) .

SpineHelper:new(config)

Overview

This function is used when creating a new animation. We must pass a table with some initial parameters. You assign the function to handle so that you can apply the rest of methods and parameters to an animation specifically.

Parameters

A lua table with at least this mandatory 3 fields:

spineJson: The name of the json file exported from Spine (i.e skeleton.json)

imageSheetName: The name of the image sheet name exported from Texture Packer (i.e. spineTutorial.png)

texturePackerLuaFile: The name of the lua file exported from Texture Packer without any extension (i.e. spineTutorial)

Other non mandatory parameters are:

debug: For debugging only (true or false)

debugAabb: For debugging only (true or false)

skin: The name of the skin you want your character to have, otherwise it will take the first one available

animationName: The name of the animation you want to apply, otherwise it will take the first one available

animLoop: Whether or not you want the animation to play forever (true or false)

scale: scale of the animation on a scale from 0 to 1

displayGroup: The name of the display group you want the animation to be inserted to

Example

```
local SpineHelper = require ("SpineHelper")

local bg = display.newRect(display.contentCenterX, display.contentCenterY,
display.contentWidth, display.contentHeight)
bg.fill = {0, 0.5, 1}

local sceneGroup = display.newGroup()

local config = {
    spineJson = "orangeGuySpine.json",
    imageSheetName = "spineTutorial1.png",
    texturePackerLuaFile = "spineTutorial1",
    --debug = true,
    -- debugAabb = true,
    skin = "greenGuy",
    animationName = "walking",
    animLoop = true,
    scale = 0.5,
    --displayGroup = sceneGroup
}

local orangeGuy = SpineHelper:new(config)
orangeGuy.x = 100
orangeGuy.y = 300
```

In this example you can see how we require the SpineHelper module at the top, then we setup the config table and at the bottom we assign the new:() method to a handle called "orangeGuy" that we can then manipulate like any object

Object:playAnimation()

Overview

This method is used when you want an animation to be played after it was stopped. This function is automatically called when you create a new animation using the SpineHelper:new(config) method.

Parameters

None

Example



```
timer.performWithDelay(5000, function()  
    orangeGuy:stopAnimation()  
  
    timer.performWithDelay(5000, function()  
        orangeGuy:playAnimation()  
    end, 1)  
end, 1)
```

In the example above, we resume playing the animation after being stopped for 5 seconds.

Object:stopAnimation()

Overview

This method is used when you want an animation to be stopped.

Parameters

None

Example

```
local orangeGuy = SpineHelper:new(config)
orangeGuy.x = 100
orangeGuy.y = 300

timer.performWithDelay(5000, function()
    orangeGuy:stopAnimation()
end, 1)
```

In the example above, the initial animation is stopped after 5 seconds of being playing

Object:changeSkin(skinName)

Overview

This method is used when you want to change the skin applied to an animation.

Parameters

Name of the skin in quotes per what was defined in Spine

Example

```
local orangeGuy = SpineHelper:new(config)
orangeGuy.x = 100
orangeGuy.y = 300

timer.performWithDelay(5000, function()
    orangeGuy:changeSkin("orangeGuy")
end, 1)
```

In the example above, the default skin is changed for the one called "orangeGuy" after five seconds

Object:changeAnimation(params)

Overview

This method is used when you want to change the animation applied to an animation.

Parameters

A lua table with at least the name of the animation:

newAnimation: Name of the new animation (mandatory)

transitionTime: time to transition from old to new animation, default is 0.5 seconds

delay: time before new animation gets applied, default is 0 seconds

track: default is 0

loop: whether or not you want the new animation to be looped, default is "true"

Example

```
local orangeGuy = SpineHelper:new(config)
orangeGuy.x = 100
orangeGuy.y = 300

timer.performWithDelay(5000, function()
    orangeGuy:changeAnimation({newAnimation="jumping"})
end, 1)
```

In the example above, the default animation is changed for a new one called "jumping", using the default parameters

Object:changeItem(params)

Overview

This method is used when you want to change or hide an image (item) that is attached to a slot. A slot can have any number of images attached to it but only one can be shown at a time.

Parameters

A lua table with at least the name of the slot, in which case would hide the item:

slot: Name of the slot that is the parent of the image item (mandatory)

item: Name of the image that you want to hide or replace by another image that is under the same slot

Example



In the example above, we are replacing an item in the slot named "tools_arrow" by another image item called "tools_lightning" after five seconds

Object:drawBoundingBox(params)

Overview

This method is used whenever you want to add interactivity to a portion of the animation by drawing a bounding box on top of the visible images. Bounding boxes are not visible, but you can apply touch events of physics to them. Animation images will follow the bounding box wherever it moves.

Parameters

A lua table with three mandatory fields as follows:

bone: Name of the bone where the bounding box is attached to (mandatory)

slot: Name of the slot where the bounding box is attached to (mandatory)

boundingBox: Name of the bounding box (mandatory)

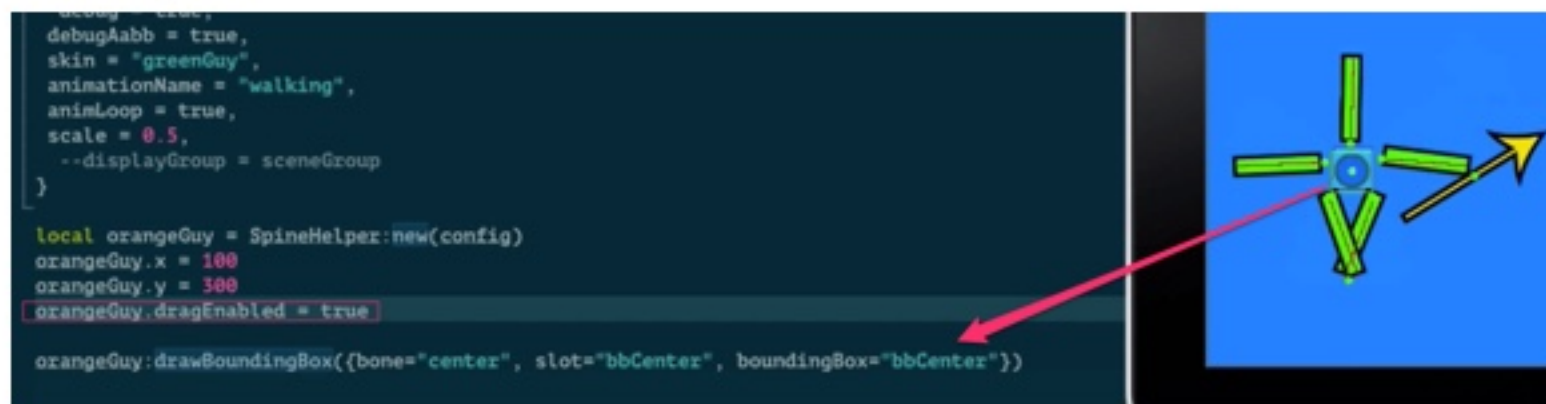
Other optional parameters are the following:

anchorX: Reference point of the bounding box on the X axis, default is 0.5

anchorY: Reference point of the bounding box on the Y axis, default is 0.5

hasBody: Whether or not you want the bounding box to be a physics object, default is "false"

Example



In the example above, we are "drawing" a bounding box called "bbCenter" that is located at the slot called "bbCenter", which is underneath the bone called "center".

If you add a property called "dragEnabled = true", you can drag the object by touching on the bounding box (by default this property is set to false). Animation will be stopped during the time the object is dragged but it will resume as soon as you stop touching the bounding box.

If you don't want the animation to continue playing after you stop dragging the object, you can set a property called "playAnimationAfterDrag = false" (by default it is set to true)

Object:drawBoundingBox(params) - (continued)

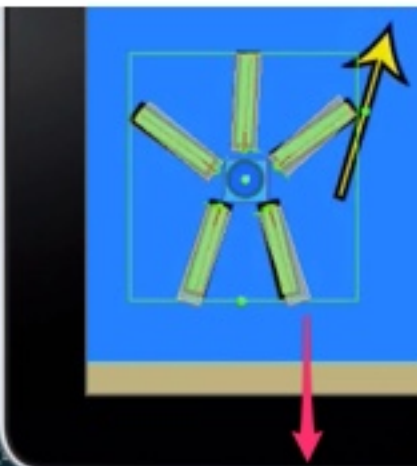
Enabling Physics on bounding boxes

if you want the bounding box to be physics enabled, you have to add a parameter called "hasBody=true" as it is illustrated below:

```
debug = true,
debugAabb = true,
skin = "greenGuy",
animationName = "walking",
animLoop = true,
scale = 0.5,
--displayGroup = sceneGroup
}

local orangeGuy = SpineHelper:new(config)
orangeGuy.x = 100
orangeGuy.y = 300
orangeGuy.dragEnabled = true
orangeGuy.playAnimationAfterDrag = true

orangeGuy:drawBoundingBox({bone="center", slot="bbCenter", boundingBox="bbCenter"})
orangeGuy:drawBoundingBox({bone="head", slot="bbHead", boundingBox="bbHead", anchorX=0, anchorY=0.5, hasBody=true })
orangeGuy:drawBoundingBox({bone="rArm", slot="bbrArm", boundingBox="bbrArm", anchorX=0, anchorY=0.5, hasBody=true })
orangeGuy:drawBoundingBox({bone="lArm", slot="bblArm", boundingBox="bblArm", anchorX=0, anchorY=0.5, hasBody=true })
orangeGuy:drawBoundingBox({bone="rLeg", slot="bbrLeg", boundingBox="bbrLeg", anchorX=0, anchorY=0.5, hasBody=true })
orangeGuy:drawBoundingBox({bone="lLeg", slot="bblLeg", boundingBox="bblLeg", anchorX=0, anchorY=0.5, hasBody=true })
```



If animation is playing, animation takes precedence on the control of the bounding boxes, but as soon as the animation is stopped, then physics will take control of the bounding boxes.

You can also modify the default physics parameters, here is the list of them:

```
--Enable Physics
if (params.hasBody) then

    self.physicsEnabled = true
    --Allow physics parameters to be passed by parameters:
    polygon.density = params.density or 0.3
    polygon.friction = params.friction or 0.3
    polygon.bounce = params.bounce or 0.2
    polygon.isSensor = params.isSensor or false
    polygon.bodyType = params.bodyType or "dynamic"
```

Recommendation: Although it is technically possible to add several bounding boxes to the different body pieces of a character, I recommend you keep them as simple as possible so you don't overcomplicate physics.

Object:createJoint(parentBone, childBone)

Overview

This method is useful to create joints between bones with physics enabled. Useful especially when you are dragging a bone and you want other bone to follow it

Parameters

parentBoneName: Name of the first bone we want to add a joint to (mandatory)

childBoneName: Name of the second bone we want to add a joint to (mandatory)

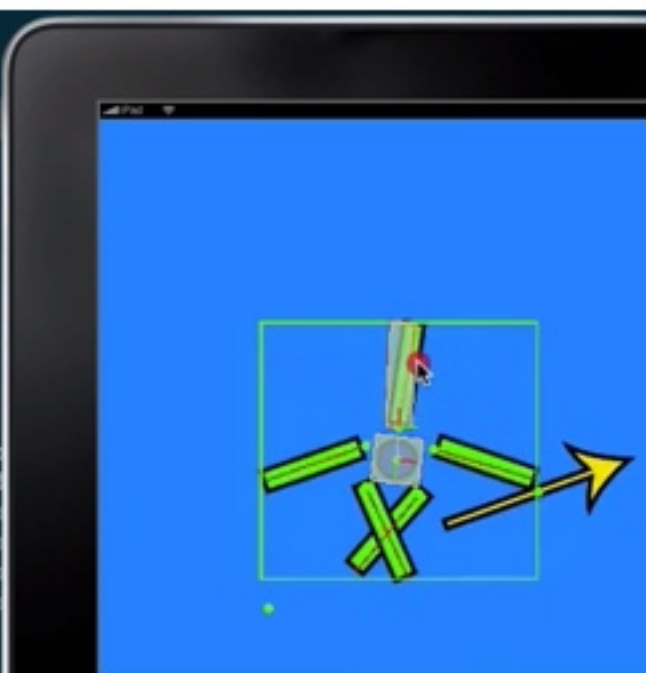
rotationLimitA: Parameter that limits the rotation of a bone (Optional)

rotationLimitB: Parameter that limits the rotation of a bone (Optional)

Note: You also need to enable physics on the bones you want to drag by adding the parameter "hasBody=true"

Example

```
debugAabb = true,  
skin = "greenGuy",  
animationName = "walking",  
animLoop = true,  
scale = 0.5,  
--displayGroup = sceneGroup  
}  
  
local orangeGuy = SpineHelper:new(config)  
orangeGuy.x = 100  
orangeGuy.y = 300  
orangeGuy.dragEnabled = true  
orangeGuy.playAnimationAfterDrag = true  
  
orangeGuy:drawBoundingBox({bone="center", slot="bbCenter", bound  
orangeGuy:drawBoundingBox({bone="head", slot="bbHead", bound  
--orangeGuy:drawBoundingBox({bone="rArm", slot="bbrArm", bou  
--orangeGuy:drawBoundingBox({bone="lArm", slot="bblArm", bou  
--orangeGuy:drawBoundingBox({bone="rleg", slot="bbrLeg", bou  
--orangeGuy:drawBoundingBox({bone="lleg", slot="bblLeg", bou  
  
orangeGuy:createJoint("center", "head")
```



In the example above, when I drag the head, the center bone will follow it. Ipad is covering the rest of properties for the bounding Boxes, but it has the parameter of "hasBody=true". Note that also the property of "dragEnabled" is set to true so we can drag the bones.

Object:hideBone(slotName, boundingBoxSlot)

Overview

This method allow us to hide a specific bone so that it is no longer visible. An example of this is when you want to hide a bone when a bullet hit a particular bone. In order to use this method, you need to create first the bounding boxes around the bones you want to hide.

Parameters

A lua table with the following parameters:

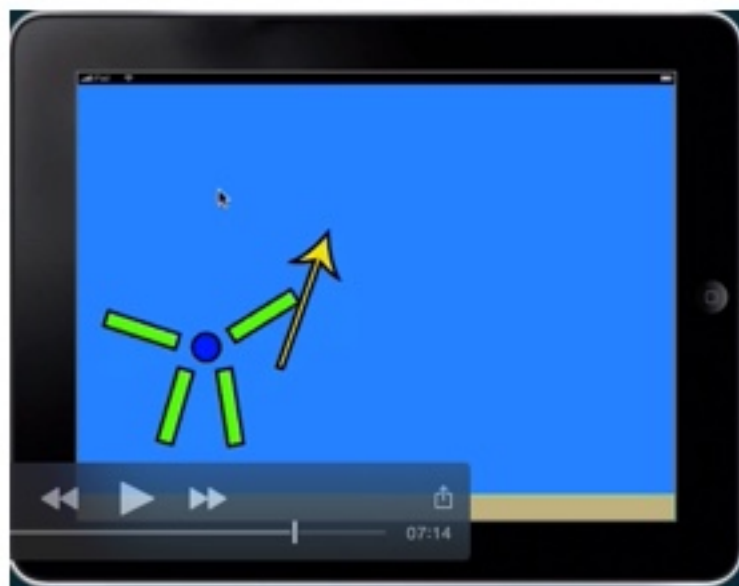
slotName: name of the slot where the bone you want to hide is located in

boundingBoxSlot: Name of the bounding box of the bone you want to hide

Example

```
timer.performWithDelay(5000, function()  
    orangeGuy:hideBone("headSkin", "bbHead")  
end, 1)
```

In the example above, the image of the head bone is hidden after 5 seconds. The bone is still there, it's just the image that is no longer visible giving the impression that the bone is gone.



Object:identifyBoundingBoxes()

Overview

Sometimes you need to have access to a particular boundingBox or specific areas of the animated object, so you can add interactivity with code. What this method does is to identify the location of the boundingBox in its table, and display the information in Terminal, and once you know its location, you can then assign it to a handle so you can manipulate it like any display object.

Parameters

None

Example

```
46
47 orangeGuy:identifyBoundingBoxes()
48
49
50
```

Output Local console

```
2014-06-01 20:36:45.565 Corona Simulator[38277:507] Non-Physics drag enabled
2014-06-01 20:36:45.569 Corona Simulator[38277:507] Drag is enabled
2014-06-01 20:36:45.570 Corona Simulator[38277:507] Non-Physics drag enabled
2014-06-01 20:36:45.575 Corona Simulator[38277:507] Drag is enabled
2014-06-01 20:36:45.576 Corona Simulator[38277:507] Non-Physics drag enabled
2014-06-01 20:36:45.581 Corona Simulator[38277:507] Drag is enabled
2014-06-01 20:36:45.582 Corona Simulator[38277:507] Non-Physics drag enabled
2014-06-01 20:36:45.588 Corona Simulator[38277:507] Drag is enabled
2014-06-01 20:36:45.588 Corona Simulator[38277:507] Non-Physics drag enabled
2014-06-01 20:36:45.589 Corona Simulator[38277:507] BoundingBox Name: center => YourInstance.imgBoxes[1]
2014-06-01 20:36:45.590 Corona Simulator[38277:507] BoundingBox Name: head => YourInstance.imgBoxes[2]
2014-06-01 20:36:45.590 Corona Simulator[38277:507] BoundingBox Name: rArm => YourInstance.imgBoxes[3]
2014-06-01 20:36:45.591 Corona Simulator[38277:507] BoundingBox Name: lArm => YourInstance.imgBoxes[4]
2014-06-01 20:36:45.591 Corona Simulator[38277:507] BoundingBox Name: rLeg => YourInstance.imgBoxes[5]
2014-06-01 20:36:45.591 Corona Simulator[38277:507] BoundingBox Name: lLeg => YourInstance.imgBoxes[6]
Program completed in 11.78 seconds (pid: 38277).
```

Let's say that we want to add a tap event to the head, so by running the method "identifyBoundingBoxes()" we see in Terminal that the head is located in the position No. 2. With this information we can assign it to a handle and then add an event listener to it.

```
local headImage = orangeGuy.imgBoxes[2]

function headImage:tap(event)
    print ("Head was tapped")
end

headImage:addEventListener("tap", headImage)
```