Lua and Corona SDK

Haeyong Chung
Spring 2017

Slides contain examples from the official Corona docs as well as the textbook.

Announcement

- Check your quiz 1 score on canvas.

Review: Display Properties

- Instances of ***display*** objects behave in a manner similar to Lua tables.

- You can change the following properties of display objects:
  - **object.alpha:** This is the object's opacity. A value of 0 is transparent and 1.0 is opaque. The default value is 1.0.
  - **object.height** and **object.width**: decide the local coordinates.
  - **object.isVisible**: Controls whether the object is visible on the screen (true or false).
  - **object.isHitTestable**: Allows an object to continue to receive hit events even if it is not visible. If true, objects will receive hit events regardless of visibility; if false, events are only sent to visible objects. It defaults to false.
    - true: even if the object is not visible, it will receive hit events (touching w/ finger)
    - false by default
  - **object.rotation**: the current rotation angle (in degrees).

Review: Display Properties (Continued)

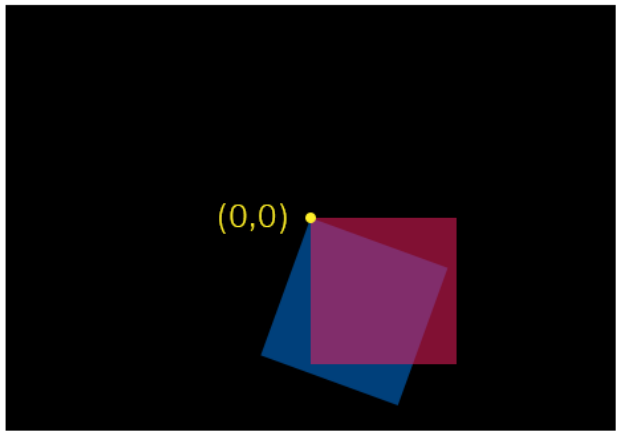
- **object.parent***:returns the parent object
- **object.contentBounds***: Table containing the boundaries: xMin, yMin, xMax, yMax
- **object.contentHeight*** and **object.contentWidth***: : This is the height and width in screen coordinates. This is similar to object.height orobject.width except that its value is affected by y scaling and rotation.
- **object.x** and **object.y**: This specifies the x and y position (in local coordinates) of the object relative to the parent -- the parent's origin to be precise.
- **object.xScale** and **object yScale**: Change scale of the object in either direction (While scaling the visual size, it will NOT change how the physics engine perceives the object)
- **object.contentCenterX** and **object.contentCenterY**: the conter of content area along x and y.

*: read only

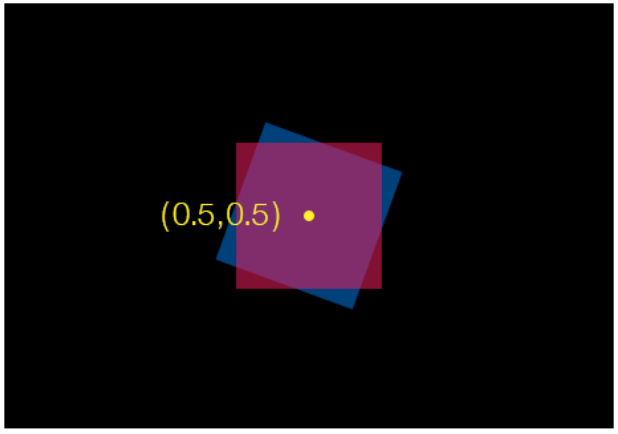
Review: Display Properties (Continued)

- **object.anchorX** and **object.anchorY:** This specifies the x and y positions of the object's alignment to the parent's origin.
  - Alignment of object relative to the (x,y) position;Values : ***0=left , 0.5=middle, 1=right***

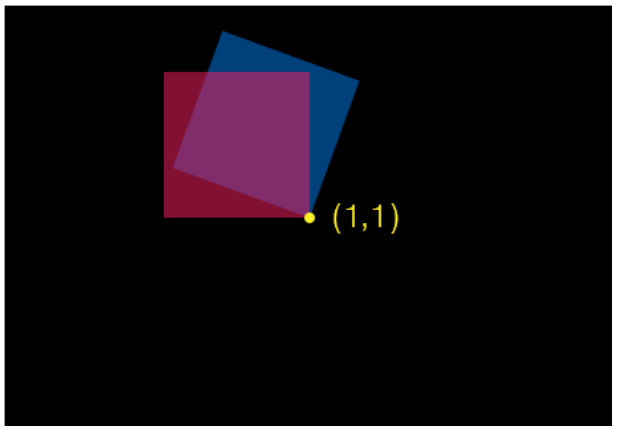
0=left



0.5=middle



1=right



Review: Display Methods

- Several methods can be called by display objects
- There are two ways this can be done: using the dot operator (“.”) or using the colon operator (“:”).
  - Examples of the dot operator
    - object = display.newRect(110, 100, 50, 50)
    - object.setFillColor(1.0, 1.0, 1.0)
    - object.translate(object, 10, 10)
  - Examples of the colon operator
    - object = display.newRect(110, 100, 50, 50)
    - object:setFillColor(1.0, 1.0, 1.0)
    - object:translate(10, 10)

Review: Syntactic Sugar: Dot vs. Colon

- Basically dot (.) and colon (:) operators are exactly the same However, the colon operator is just a syntactic sugar.
  - function object:method(arg1, arg2) is same as function object.method(self, arg1, arg2).
  - i.e., the colon is for implementing methods that pass self as the first parameter. So x:bar(3,4)should be the same as x.bar(x,3,4).

Review: Display Methods (Continued)

- Display objects share the following methods:
  - **object:rotate(deltaAngle)** or **object.rotate(object, DeltaAngle)**:adds deltaAngle (in degrees) to the current rotation property.
  - **object:scale(sx, sy)** or **object.scale(object, sx, sy):** multiplies the xScale and yScale properties using sx and sy, respectively.
  - **object:translate(deltaX, deltaY)** or object.translate(object, deltaX, deltaY): This effectively adds deltaX and deltaY to the x and y properties respectively. This will move the object from its current position.
  - **object:removeSelf()** or **object.removeSelf(object)**: This removes the display object and frees its memory, assuming that there are no other references to it.

Review: Runtime Configuration

- Config.lua allows you to
  - dynamic content scaling,
  - dynamic content alignment
  - dynamic image resolution
  - frame rate control, and etc.

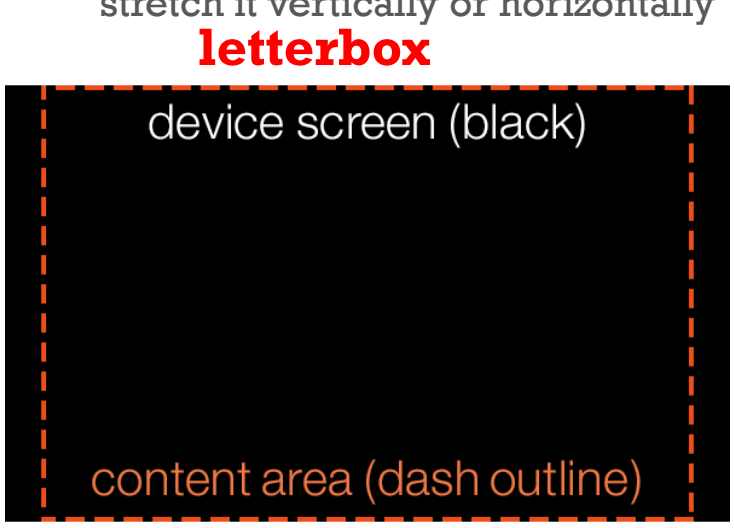
Reiview: Runtime Configuration (Continued)

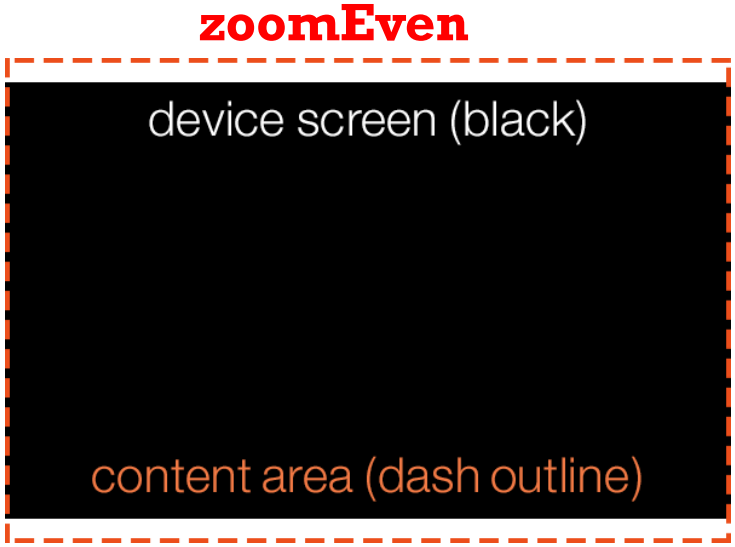
- One Config.lua Example:

```
application =
{
    content =
    {
      width = 320,
      height = 480,
      scale = "letterbox",
      fps = 30,
      xAlign = “left”,
      yAlign = “bottom”
    },
}
```

Review Runtime Configuration

- Dynamic Content Scaling
  - width (number): This is the screen resolution width of the original target device
  - height (number): This is the screen resolution height of the original target device
  - scale (string): This is a type of autoscaling from the following values:
    - letterbox: This scales up content uniformly as much as possible
    - zoomEven: This scales up content to uniformly to fill the screen, while keeping the aspect ratio
    - zoomStretch: This scales up content nonuniformly to fill the screen and will stretch it vertically or horizontally





Scale

Original Image

Letterbox

"letterbox"

ZoomEven

"zoomEven"

ZoomStretch

"zoomStretch"

Review: Runtime Configuration (Continued)

- Dynamic Image Selection: Automatically swap in higher resolution versions of your images to higher resolution devices

Review: Runtime Configuration (Continued)

- Dynamic Image Selection (continued): Automatically swap in higher resolution versions of your images to higher resolution devices
  - Add @2x suffix to the end of filename (e.g., myImage@2x.png)
  - In your project config.lua file, a table named imageSuffix needs to be added for the image naming convention and image resolutions to take effect

```
application =
{
        content =
        {
                width = 320,
                height = 480,
                scale = "letterbox",
                imageSuffix = {
                        ["@2x"] = 2,
                },
        },
}
```

  - In our main.lua, you have call your display objects by using display.newImageRect, instead of displaynewImage().

Function

- Define a function

```
function sayThis (word)
      print (word);
end
sayThis(“Hello”); -- Hello
```

- Define a function as a variable

```
local sayThis;
sayThis = function ()
…
end
```

Function (continued)

```
doThis();
function doThis ()
   print ("Yes!");
end
```

doThis() is undefined when you call it.

Function (continued)

```
local doThis;
doThis();
function doThis ()
     print ("Yes!");
end
```

Can't make the actual call before it's defined.

Shapes

- Create shapes (vector objects) such as rectangles, circles, and rounded rectangles using these methods:
  - **display.newRect([parentGroup,] x, y, width, height):** This creates a rectangle using width by height.
    display.newRoundedRect([parentGroup,] x, y, width, height, cornerRadius): This creates a rounded rectangle using width by height.
  - **display.newCircle([parentGroup,] xCenter, yCenter, radius)**: This creates a circle using the radius centered at xCenter, yCenter.

Shapes (continued)

- Style methods
  - **object.strokeWidth:** This creates the stroke width in pixels
  - **object:setFillColor(red, green, blue, alpha):** We can use the RGB codes between 0 and 1. The alpha parameter, which is optional, defaults to 1.0
  - **object:setStrokeColor(red, green, blue, alpha):** We can use the RGB codes between 0 and 255. The alpha parameter, which is optional, defaults to 1.0

Rectangle Example

```
local box = display.newRect (200, 300, 100, 100);
box.strokeWidth = 10;
box:setStrokeColor(0.5,0.5,1);
```

Text

- display.newText([parentGroup,] text, x, y, font, fontSize)
  - Fonts:
    - Font
      - Enter font name, 'New Times Roman'
      - native.systemFont
      - native.systemFontBold
    - Color and String Value
      - Size: object.size
      - Color: object:setFillColor(red,gree,blue,alpha)
      - Text: object.text – this allow you to update a sting value

Change the Appearance of the Status Bar

- change the appearance of your status bar using the **display.setStatusBar(mode)** method

Events and Listeners

- Register Events
  - Runtime Events
  - Touch Events

Register Events

- object:addEventListener(eventType , handlerFunc)
  - eventType: what type of event is this?
- object:removeEventListener (..)

Runtime Events

- All mobile applications use runtime events.
- No specific target on your mobile screen and Broadcast to all listners
- Example – enterFrame
  - event.name is the string "enterFrame“
  - event.time is the time in milliseconds since the start of the application

```
local playBtn = display.newImage("playbtn.png")
local function listener(event)
      print("The button appeared.")
end
Runtime:addEventListener("enterFrame", listener )
```

Runtime Events (continued)

- Example – **accelerometer**
  - event.name is the string "accelerometer"
  - event.xGravity is the acceleration due to gravity in the x direction
  - event.yGravity is the acceleration due to gravity in the y direction
  - event.zGravity is the acceleration due to gravity in the z direction
  - event.xInstant is the instantaneous acceleration in the x direction
  - event.yInstant is the instantaneous acceleration in the y direction
  - event.zInstant is the instantaneous acceleration in the z direction
  - event.isShake is true when the user shakes the device

Touch Events

- When a user's finger touches the screen, they are starting a sequence of touch events, each with different phases.
  - event.name is the string "touch"
  - event.x: the x position in the screen coordinates of the touch
  - event.y: the y position in the screen coordinates of the touch
  - event.target: a reference to the display object associated with the touch event.

Touch Events (continued)

```
local box = display.newRect (200,300,100,100);
box.strokeWidth = 10;
box:setStrokeColor(0.5,0.5,1);
-- Upon tapping the box, make it bigger
local function touched (event)
  event.target:scale ( 1.1, 1.1);
end
box:addEventListener("touch", touched);
```

Touch Events (continued)

- event.phase is a string that identifies where in the touch sequence the event occurred:
  - "began"
  - "moved”
  - "ended“
  - “cancelled”
- event.xStart: the x position of the touch from the "began" phase of the touch sequence
- event.yStart is the y position of the touch from the "began" phase of the touch sequence

- You can check to see which phase it is at in the handler function
  - if (event.phase == " began ") do

Touch Events (continued)

```
local box = display.newRect (200,300,100,100);
box.strokeWidth = 10;
box:setStrokeColor(0.5,0.5,1);
-- Upon tapping the box, make it bigger
local function touched (event)
  print ("phase: " .. event.phase );
  event.target:scale ( 1.1, 1.1);
end
box:addEventListener("touch", touched);
```

Tab Events

- It generates a hit event when the user touches the screen.
- It is similar to the touch event except that it doesn't have any event phases.
- Event properties are as follows:
  - event.name: the string "tap"
  - event.numTaps returns the number of taps on the screen (either single or double tap)
  - event.x: the x position in the screen coordinates of the tap
  - event.y: the y position in the screen coordinates of the tap

Event handling - tap

```
local box = display.newRect (200,300,100,100);
box.strokeWidth = 10;
box:setStrokeColor(0.5,0.5,1);
-- Upon tapping the box, make it bigger
local function tapped(event)
     print(event.numTaps)
     event.target:scale ( 1.1, 1.1);
end
box:addEventListener("tap", tapped);
```

Touch Events: Dragging An Object

```
local box = display.newRect( 100, 100, 100, 100 )
box:setFillColor( math.random(),1,0.5 );
local function move ( event )
  if event.phase == "moved" then
    event.target.x = event.x;
    event.target.y = event.y;
  end
end
box:addEventListener( "touch", move )
```

Drag and Drop Objects Properly

```
local box = display.newRect(0,0, 100,100);
box.x = display.contentCenterX -- Move the object to the
box.y = display.contentCenterY  -- center of the screen
local function drag (event)
    print (event.target,event.phase);
    if event.phase == "began" then
        event.target.markX = event.target.x
        event.target.markY = event.target.y
    elseif event.phase == "moved" then
        local x = (event.x - event.xStart) + event.target.markX;
        local y = (event.y - event.yStart) + event.target.markY;
        event.target.x, event.target.y = x, y;
    elseif event.phase == "ended" then
      --event.target:setFillColor( <RANDOM> );
    end
end

x:addEventListener ("touch", drag);
```

Group

- Group objects are a special type of display object.
- In a display group, you can add/remove display objects as children of the group.
- In this case, the display group becomes a ***parent*** of those display objects.

```
local group = display.newGroup()
local rect = display.newRect( 100, 100, 50, 50 )
rect:setFillColor( 0.7 )
group:insert( rect ) --add rect to group
rect.parent:remove( rect ) --removes rect from group
```

Displaying Groups

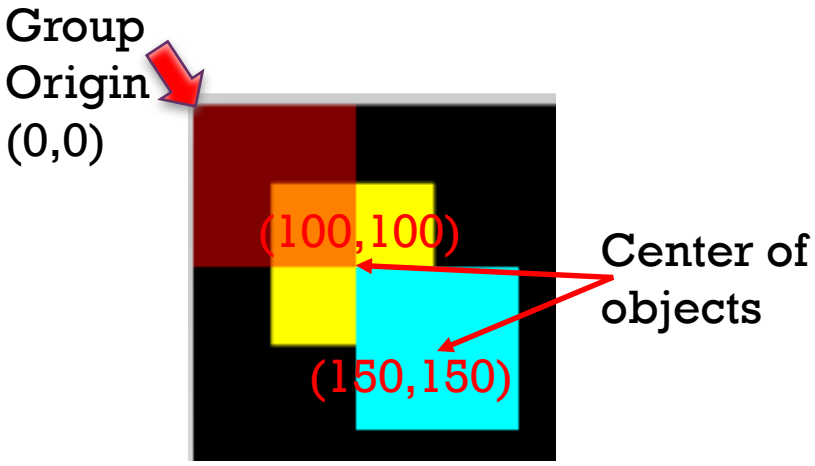
```
local ref = display.newRect (50, 50, 100, 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (100, 100, 100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (150, 150, 100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b); -- insert b into g
ref:toFront();
```

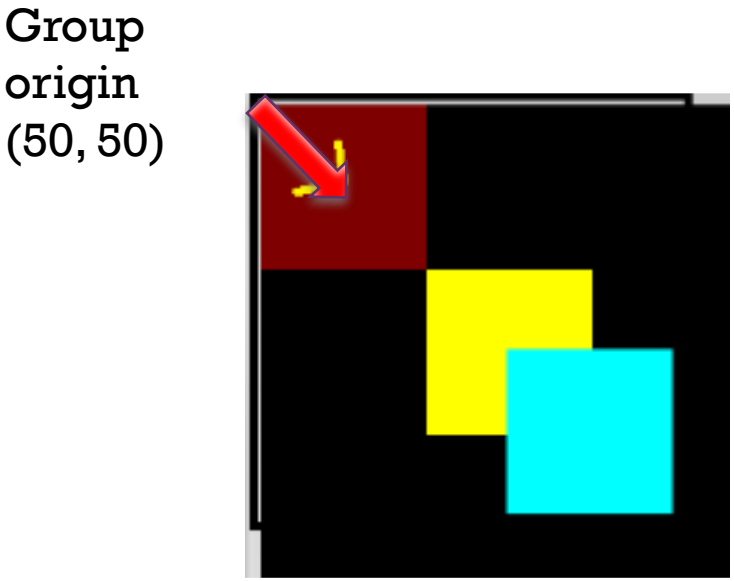
Displaying Groups (continued)

```
local ref = display.newRect (50, 50, 100, 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (100, 100, 100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (150, 150, 100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b); -- insert b into g
ref:toFront();
g.x = 50; -- Does this do what you think it does?
g.y = 50;
```

Displaying Groups

- Group position (.x and .y) moves the origin of the group
- child objects' positions are relative to the origin of the parent (the group)

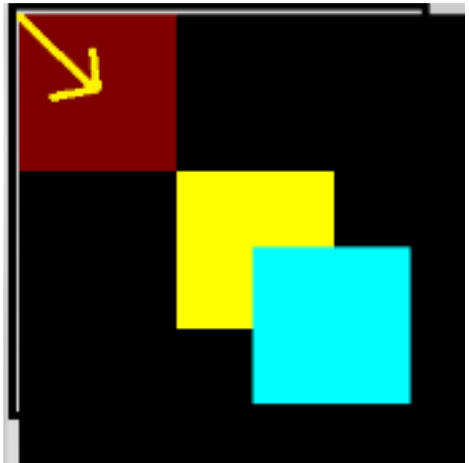




Displaying Groups (continued)

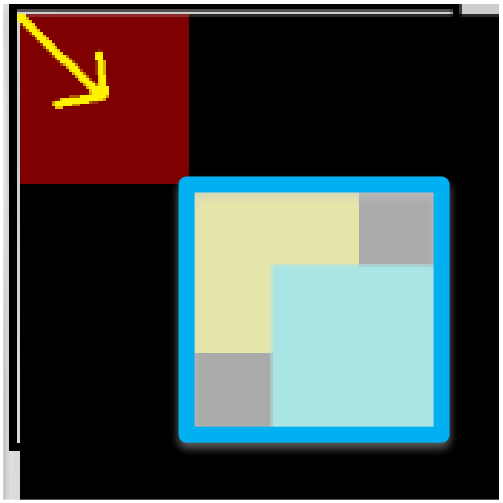
- anchorX and anchorY for the group does not change anything visually…
  - **group.anchorChildren = false** by default
    - set to **true** to get all child objects to obey anchor points
    - g.anchorX = 0;
    - g.anchorY = 0;
    - g.anchorChildren = true;

What is happening?

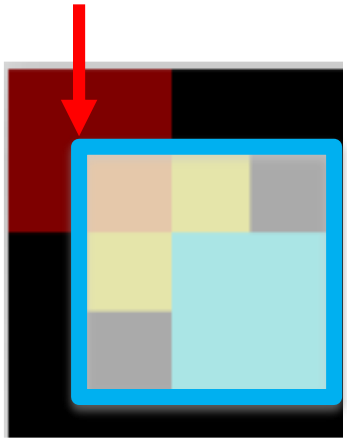


Displaying Groups (continued)

- Setting the g.anchorChildren to true ***creates a bounding box around all of the objects in the group.***



group's anchor point of 0, 0

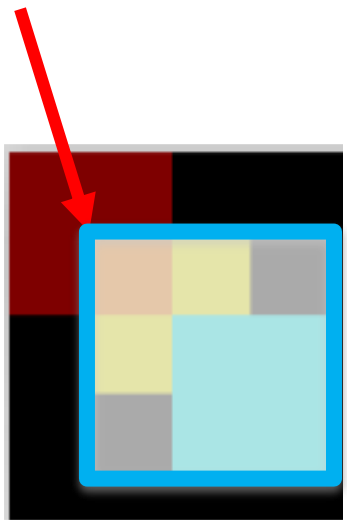


Basically sets the anchor point while viewing all of the bounded child objects as one object.

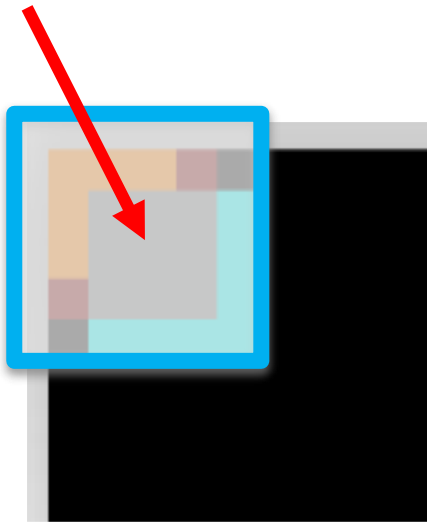
Displaying Groups (continued)

- Try changing the anchor point to 0.5, 0.5:

group's anchor point of 0, 0



group's anchor point of 0.5, 0.5

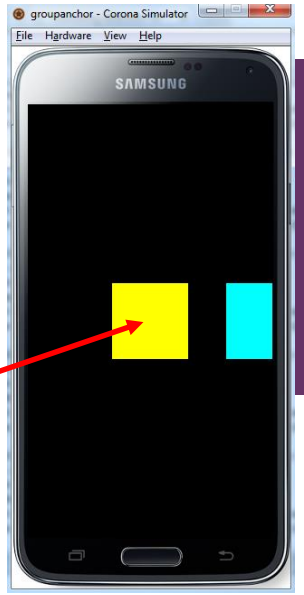


Anchor points

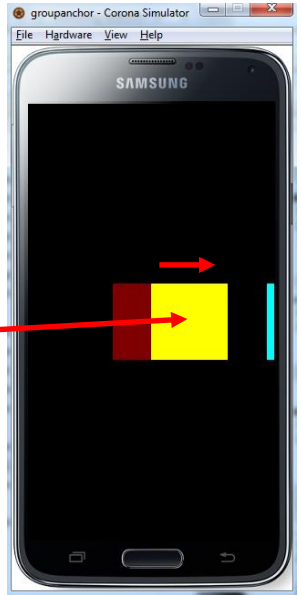
- For typical display objects:
  - Determines how the object is placed relative to it's coordinate
- For a group:
  - by default, anchor points are not used
  - The default origin is at (0,0), but a display group does not
- really have bounds
  - If anchorChildren is set to true, behavior is not what one
- might expect:
  - It does not really affect the child objects' anchoring
  - It forms a bounding box around the entire group and moves
  - the anchor point for the group
    - Child objects relative' positions stay the same
- It is not recommended

Displaying Groups (continued)

```
xx=display.contentCenterX;
yy=display.contentCenterY;
local ref = display.newRect (xx, yy, 100 , 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (xx, yy, 100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (xx+150, yy, 100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b);
g.x = 50;
```



g.x = 50;



Rotating Group

```
xx=display.contentCenterX;
yy=display.contentCenterY;
local ref = display.newRect (xx, yy, 100 , 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (xx, yy, 100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (xx+150, yy, 100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b);
g.x = 50;
g:rotate(45);
```

Rotating Group (continued)

```
xx=display.contentCenterX;
yy=display.contentCenterY;
local ref = display.newRect (xx,yy, 100 , 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (      100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (       100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b);
g.x = xx;
g.y = yy;
g:rotate(45);
```

Timer Functions

- timer.performWithDelay(delay, listener [, iterations])
  - 1st parameter is time in milliseconds
  - last parameter is number of iterations.
    - 0 or -1 -> infinite loop
  - returns a reference to the timer (i.e., TimerID; need to be saved for later)

Timer Functions

- timer.performWithDelay(delay, listener [, iterations])
  - 1st parameter is time in milliseconds
  - last parameter is number of iterations.
    - 0 or -1 -> infinite loop
  - returns a reference to the timer (i.e., TimerID; need to be saved for later)

```
local timerRef =
timer.performWithDelay(
500,
function()
  g.rotation = g.rotation+10;
end,
0
)
```

Timer Functions (continued)

- **timer.pause(timerID)**
- **timer.resume(timerID)**
- **timer.cancel(timerID)**

```
local function myEvent()
        print( "myEvent called" )
end
myTimerID = timer.performWithDelay( 3000, myEvent ) -- wait
3 seconds
result = timer.pause( myTimerID ) -- Pauses myTimerID
print( "Time paused at " .. result )
result = timer.resume( myTimerID ) -- Resumes myTimerID
print( "Time resumed at " .. result )
```

Timer Functions (continued)

```
local timerRef = timer.performWithDelay(
500,
function()
        g.rotation = g.rotation+10;
end,
0
)
```

Timer Functions (continued)

```
xx=display.contentCenterX;
yy=display.contentCenterY;
local ref = display.newRect (xx,yy, 100 , 100);
ref:setFillColor(1,0,0,0.5);
local a = display.newRect (0, 0, 100 , 100);
a:setFillColor( 1, 1, 0);
local b = display.newRect (150, 0, 100 , 100);
b:setFillColor( 0, 1, 1);
local g = display.newGroup ();
g:insert (a); -- insert a into g
g:insert (b);
g.x = xx
g.y = yy
local timerRef = timer.performWithDelay(
500,
function()
       g.rotation = g.rotation+10;
end,
0
)
```

Other Time-related Functions

- **system.getTimer()**

```
local t = system.getTimer()
-- do something
local t2 = system.getTimer() - t;
```

- **os.time()**

```
local time = os.time() -- This is time in
seconds
local dateTime = os.date("*t", time) -- *t is
the one of the, there are other formats too.
print(dateTime.year, dateTime.month,
dateTime.hour, dateTime.min)
```

Transitions

- transition.to(target, params)
  - Call a function / Animate a display object over a specific period of time.
- transition.from(target, params)
  - the starting property values are specified in the parameters table
- **target**: a display object that will be the target of the transition.
- **params**: This is a table that specifies the properties of the display object, which will be animated

Transitions (continued)

- time: This specifies the duration of the transition in milliseconds. By default, the duration is 500 ms (0.5 seconds).
- transition: This is by default easing.linear. https://docs.coronalabs.com/api/library/easing/index.html
- delay: This specifies the delay in milliseconds (none by default) before the tween begins.
- delta: This is a Boolean that specifies whether noncontrol parameters are interpreted as final ending values or as
- changes in value. The default is nil, meaning false.
- onStart: This is a function or table listener that is called before the tween begins.
- onComplete: This is a function or a table listener that is called after the tween completes.
- x, y: Moves the target from its current **x/y** coordinate to another.

Check other parameters at
https://docs.coronalabs.com/api/library/transition/to.html

Transitions (continued)

```
W = display.contentWidth
H = display.contentHeight
local square = display.newRect( 0, 0, 100, 100 )
square:setFillColor( 1, 0.5, 0.5 )
square.x = W/2; square.y = H/2
local square2 = display.newRect( 0, 0, 50, 50 )
square2:setFillColor( 1, 1, 1 )
square2.x = W/2; square2.y = H/2
transition.to( square, { time=1500, x=250, y=400 } )
transition.from( square2, { time=1500, x=275, y=0 } )
```

Easing

- Easing Test:

```
local circle = display.newCircle( 100, 100, 40 )
circle:setFillColor( 0, 0, 1 )
transition.to( circle, { time=400, y=200, transition=easing.inExpo }
```

https://docs.coronalabs.com/daily/api/library/easing/index.html

Demo for transition.to

- How can we create an animated shape (infinite loop)?

Drag + transition.to


```
local radius = 50.0
local X = display.contentWidth/2
local Y = display.contentHeight/2
local i = 0;

local myCircle = display.newCircle( X, Y, radius )
        myCircle:setFillColor( 0, 0.5, 0.75 )
        myCircle.strokeWidth = 9
        myCircle:setStrokeColor( 1, 1, 1 )
function draw()
        i = i + 1;
        radius = radius + math.sin( i / 4);
        transition.to( myCircle.path, { time=10, radius=radius, onComplete=draw } )
end
draw();
local function move ( event )
        if event.phase== "began" then
                event.target.markX= event.target.x
                event.target.markY= event.target.y
        elseif event.phase== "moved" then
                local x = (event.x-event.xStart) + event.target.markX
                local y = (event.y-event.yStart) + event.target.markY
                event.target.x= x;
                event.target.y= y;
        end
end
myCircle:addEventListener( "touch", move )
```

Updating Text at Runtime

```
local timevalue=0;
local barH = 100;
local bar = display.newRect(display.contentCenterX, 50,
display.contentWidth, barH);
local timeText = display.newText ("TIME:", 100, bar.y,
native.systemFont, 50);
timeText:setFillColor (0,0,0);
local timeVal = display.newText (timevalue, timeText.x +
timeText.width, bar.y, native.systemFont, 50);
timeVal:setFillColor (0,0,0);
```

You can update the text by using the .text property on the text object:
timeVal.text = …

Updating Text at Runtime

```
local timevalue=0;
local barH = 100;
local bar = display.newRect(display.contentCenterX, 50,
display.contentWidth, barH);
local timeText = display.newText ("TIME:", 100, bar.y,
native.systemFont, 50);
timeText:setFillColor (0,0,0);
local timeVal = display.newText (timevalue, timeText.x +
timeText.width, bar.y, native.systemFont, 50);
timeVal:setFillColor (0,0,0);
local timerRef = timer.performWithDelay(
1000,
function()
  --timeVal.text = tostring(timeVal.text+1);
  timeVal.text = timeVal.text+1;
end,
0
)
```