JSPro.com

Happy Holidays from CoffeeScript

Happy Holidays from CoffeeScript (HHFCS) is an HTML5/CoffeeScript application that uses HTML5's Audio and Canvas APIs along with CoffeeScript-based code to present a celebration of this festive time of year. Because the need for brevity prevented me from showing you how the CoffeeScript code worked in the article itself, I've create this PDF file to give you that information.

Exploring HHFCS.coffee

The HHFCS application is based on `HHFCS.coffee` and `Snowflake.coffee` source files, where the former source file is the entry-point into this application. Listing 1 presents this CoffeeScript file's contents.

***Listing 1:** Describing the `HHFCS` class in CoffeeScript*

```
class HHFCS
   @init: (ms) ->
      canvas = document.createElement "canvas"
      canvas.setAttribute "width", 800
      canvas.setAttribute "height", 528
      document.getElementsByTagName("body")[0].appendChild canvas
      HHFCS.ctx = canvas.getContext "2d"

      HHFCS.ctx.font = "30px Arial"
      HHFCS.ctx.textAlign = "center"

      HHFCS.width = canvas.width
      HHFCS.height = canvas.height

      HHFCS.imgMessage = new Image
```

```
    HHFCS.imgMessage.src = "images/message.png"

    HHFCS.imgScene = new Image
    HHFCS.imgScene.src = "images/scene.png"

    HHFCS.imgWreath = new Array
    for i in [0..2]
       image = new Image
       image.src = "images/wreath"+i+".png"
       HHFCS.imgWreath.push image

    HHFCS.curWreath = 0
    HHFCS.wreathSlowDownCounter = 0

    HHFCS.flakes = []
    for i in [0..NFLAKES-1]
       radius = rnd Snowflake.MAX_RADIUS # range [0, MAX_RADIUS)
       if (radius < Snowflake.MIN_RADIUS)
          radius = Snowflake.MIN_RADIUS
       HHFCS.flakes[i] =  new Snowflake HHFCS.ctx, radius, "#fff",
                                        rnd(HHFCS.width),
                                        -2*radius-rnd(1000), ms

    HHFCS.audJBLoaded = false
    HHFCS.audJB = document.createElement "audio"
    HHFCS.audJB.onloadeddata = new (e) ->
       HHFCS.audJBLoaded = true

    if navigator.userAgent.indexOf("Firefox") != -1 ||
        navigator.userAgent.indexOf("Opera") != -1
       HHFCS.audJB.src = "audio/jb.ogg"
    else
       HHFCS.audJB.src = "audio/jb.mp3"
    HHFCS.audJBPlaying = false

    HHFCS.startTime = new Date().getTime();

  @draw: ->
```

```
    if not allResourcesLoaded()
       HHFCS.ctx.fillStyle = "#000" # black
       HHFCS.ctx.fillRect 0, 0, HHFCS.width, HHFCS.height
       HHFCS.ctx.fillStyle = "#fff" # white
       HHFCS.ctx.fillText "Initializing...", HHFCS.width/2,
                          HHFCS.height/2
       return

    HHFCS.ctx.drawImage HHFCS.imgScene, 0, 0

    for i in [0..NFLAKES-1]
       HHFCS.flakes[i].draw HHFCS.ctx

    HHFCS.ctx.drawImage HHFCS.imgWreath[HHFCS.curWreath], 10, 395
    HHFCS.ctx.drawImage HHFCS.imgWreath[HHFCS.curWreath],
                        HHFCS.width-HHFCS.imgWreath[0].width-10, 395

    if ++HHFCS.wreathSlowDownCounter == 5
       HHFCS.curWreath = (HHFCS.curWreath+1)%HHFCS.imgWreath.length
       HHFCS.wreathSlowDownCounter = 0

    HHFCS.ctx.globalAlpha = (new Date().getTime()-HHFCS.startTime)/DURATION
    HHFCS.ctx.drawImage HHFCS.imgMessage, (HHFCS.width-HHFCS.imgMessage.width)/2,
                        (HHFCS.height-HHFCS.imgMessage.height)/2
    HHFCS.ctx.globalAlpha = 1.0

    if not HHFCS.audJBPlaying
       HHFCS.audJB.play()
       HHFCS.audJBPlaying = true

# ================================================================
# NOTE: The rest of the properties in this namespace are private.
# ================================================================

NFLAKES = 200 # maximum number of snowflakes

DURATION = 30000 # milliseconds
```

```
allResourcesLoaded = ->
   status = HHFCS.imgMessage.complete && HHFCS.imgScene.complete
   for i in [0..HHFCS.imgWreath.length-1]
      status = status && HHFCS.imgWreath[i].complete
   status = status && HHFCS.audJBLoaded


rnd = (limit) ->
   (Math.random()*limit)|0 # |0 converts to integer
```

Listing 1 first declares a class named `HHFCS`. (Coming from a Java background, I appreciate CoffeeScript's basic class structure, which the CoffeeScript compiler maps onto the JavaScript equivalent.) This is followed by a public section consisting of `init` and `draw` class method properties, and a private section that isn't exposed beyond this class.

CoffeeScript has many nice features, including indentation instead of brace characters for marking blocks (e.g., the bounds of a method or for loop) and optional semicolons. However, indentation can be problematic. For example, I've sometimes inserted an alert method call at the wrong indent level, which has resulted in this method not being called at the right point (or the compiler generating an error message).

Each of the `init` and `draw` method properties starts with a header beginning with an `@`-prefixed name. When a property starting with `@` appears underneath a class, the name is added to the class, and can be considered a class property. In this case, `init` and `draw` are properties of the `HHFCS` class.

Continuing, the header is followed by an optional parameter list and thin arrow (`->`), which introduces a function property. The parameter list assigned to `init` is `(ms)`—`init` takes a single parameter consisting of the number of milliseconds used as an interval between successive calls to `draw`. In contrast, `draw` has no parameter list because it's called with no arguments.

Consider `init`. This method first creates a `canvas` object by calling the Document Object Model's `createElement` method with a `"canvas"` argument. Notice the absence of round brackets in the method call. Although often optional, round brackets are sometimes necessary (as you'll see). Also, notice the absence of `var`—CoffeeScript forbids this problematic keyword.

> **NOTE:** One problem with JavaScript is that you can accidentally introduce a global variable by forgetting to specify `var` before a variable name when introducing the variable. CoffeeScript avoids this problem by not letting you specify `var`, and inserting this keyword as necessary.

Next, `init` assigns a width and height to the canvas by invoking the `setAttribute` method on the `canvas` object twice. It then appends the canvas to the page's body, and obtains a 2D context for drawing on the canvas. I've prefixed the `ctx` variable with "`HHFCS.`" so that I can add `ctx` as a property of `HHFCS`.

> **NOTE:** As you previously learned, prefixing a method property with @ adds that property to the class. However, the meaning of the @ symbol changes when used from within a method. In this context, @ becomes shorthand for "`this.`".

CoffeeScript lets you remove round brackets from method calls, but doing this can be problematic. For example, when I remove them from `"body"`, as in `document.getElementsByTagName "body"[0].appendChild canvas`, CoffeeScript yields the incorrect `document.getElementsByTagName("body"[0].appendChild(canvas));`.

Moving on, `init` initializes the canvas context's `font` and `textAlign` properties, which will be used to control the display of an `Initializing...` message that's drawn when not all of the image and audio resources have loaded. For convenience, the canvas width and height are saved for later access.

At this point, `init` starts to load the various image resources. The code that loads the sequence of wreath images demonstrates another nice CoffeeScript feature: for loop combined with range. The `for i in [0..2]` syntax assigns 0, 1, and 2 to `i` during successive loop iterations, and is equivalent to (and more compact than) the equivalent `for (i = 0; i < 2; i++)`.

`init` now introduces `curWreath` and `wreathSlowDownCounter` properties that control the wreath animation, and which will be explained when I discuss the `draw` method. Then, a `flakes` array of `NFLAKES Snowflake` objects is created (one object per snowflake), an audio resource is selected and its loading begins, and a time reference is obtained (discussed later).

Notice the `HHFCS.audJB.onloadeddata = new (e) ->` syntax for assigning an event handler to run after the audio resource has loaded. The `new` keyword is necessary because event handlers are objects and the absence of `new` assigns a function (not an object) to `onloadeddata`. Chrome and Safari ignore the assigned function (and there is no audio) when `new` is absent.

Let's now consider the `draw` method. It first executes `if not allResourcesLoaded()` to determine if all image and audio resources have loaded—`not` is an alias for `!`. If they haven't all loaded, a white `Initializing...` message on a black background is presented. The previous assignment of `"center"` to `textAlign` makes it easy to center the text.

Assuming that all resources have loaded, `draw` proceeds to draw the background image, followed by all snowflakes and the wreath (in two locations). To slow down the wreath animation so that it looks more realistic, the wreath index advances only after every 5 `draw` calls, which `wreathSlowDownCounter` tracks.

At this point, `draw` fades in a centered message by assigning the result of expression `(new Date().getTime()-HHFCS.startTime)/DURATION` to the canvas context's `globalAlpha` property. This expression initally evaluates to 0, which means that the message is transparent (invisible). Over a 30-second period, it approaches (and eventually exceeds) 1—the image remains fully opaque.

For some reason, it's not possible to start playing the audio from the event handler assigned to the previously mentioned `onloadeddata` property. Instead, that handler assigns `true` to a property that's tested at the end of `draw`. When set to `true`, the audio playing begins and this property is reset to `false`.

The remainder of Listing 1 is dedicated to establishing some private properties. These properties are not accessible beyond `HHFCS` because of the indentation and = assignments (e.g., `NFLAKES = 200`). The result of the indentation and = is to introduce local variables and local functions in the equivalent JavaScript.

Exploring Snowflake.coffee

Listing 1's `HHFCS.coffee` source code referenced an external `Snowflake` class, which is responsible for describing and drawing a snowflake according to various characteristics. Listing 2 presents `Snowflake.coffee`.

***Listing 2:** Describing the `Snowflake` class in CoffeeScript*

```
class Snowflake
   @MAX_RADIUS: 30
   @MIN_RADIUS: 3

   constructor: (@ctx, @radius, @strokeStyle, @x, @y, @msInterval) ->
      @startX = @x
      @startY = @y
      @path = []
      for branch in [0..5]
         angle = toRadians branch*60.0+30.0
         snowflakeBranch this, 0.0, 0.0, rotateX(@radius, 0.0, angle),
                         rotateY(@radius, 0.0, angle), 0
      radiusDiff = Snowflake.MAX_RADIUS-@radius
      if (radiusDiff == 0)
         radiusDiff = 1 # prevent division by zero
      @incr = @ctx.canvas.height/radiusDiff/(@msInterval/5)

   draw: ->
      @ctx.strokeStyle = @strokeStyle
      @ctx.beginPath()
      for element in @path
         if element.cmd == LINETO
            @ctx.lineTo @x+element.x, @y+element.y
         else
            @ctx.moveTo @x+element.x, @y+element.y
      @ctx.closePath()
      @ctx.stroke()
      @y += @incr
      if @y > @ctx.canvas.height
```

```
      @x = @startX
      @y = @startY

# ================================================================
# NOTE: The rest of the properties in this namespace are private.
# ================================================================

BRANCH_ANGLE = 30.0*Math.PI/180.0
BRANCH_FACTOR = 0.33
SHRINK_FACTOR = 0.66

LINETO = 0
MOVETO = 1

rotateX = (x, y, angle) ->
   x*Math.cos(angle)+y*Math.sin(angle)

rotateY = (x, y, angle) ->
   -x*Math.sin(angle)+y*Math.cos(angle)

snowflakeBranch = (self, startX, startY, endX, endY, depth) ->
   return if depth == 4
   self.path.push { cmd: MOVETO, x: startX, y: startY }
   self.path.push { cmd: LINETO, x: endX, y: endY }
   cX = startX+(endX-startX)*BRANCH_FACTOR
   cY = startY+(endY-startY)*BRANCH_FACTOR
   nendX = cX+(endX-startX)*SHRINK_FACTOR
   nendY = cY+(endY-startY)*SHRINK_FACTOR
   rX1 = rotateX(nendX-cX, nendY-cY, BRANCH_ANGLE)+cX
   rY1 = rotateY(nendX-cX, nendY-cY, BRANCH_ANGLE)+cY
   rX2 = rotateX(nendX-cX, nendY-cY, -BRANCH_ANGLE)+cX
   rY2 = rotateY(nendX-cX, nendY-cY, -BRANCH_ANGLE)+cY
   snowflakeBranch self, cX, cY, rX1, rY1, depth+1
   snowflakeBranch self, cX, cY, rX2, rY2, depth+1

toRadians = (degrees) ->
   degrees*Math.PI/180.0
```

Listing 2 first declares class `Snowflake`, and then adds `MAX_RADIUS` and `MIN_RADIUS` properties to this class (accessible via `Snowflake.MAX_RADIUS` and `Snowflake.MIN_RADIUS`). They respectively identify the maximum and minimum radius of any generated snowflake, and are accessed from `HHFCS`'s `init` method.

A constructor is now declared for initializing a `Snowflake` object. CoffeeScript requires that constructors be named via the `constructor` keyword to improve the quality of stack trace information. As with regular methods, the thin arrow (`->`) is used to signify that this is a block of code to be executed.

`Snowflake`'s constructor presents a parameter list consisting of six parameters:

- `ctx`: the context of the canvas on which the snowflake will be drawn. I would have preferred to pass the context to `Snowflake`'s `draw` instance method instead, but I need this context to perform a calculation in the constructor.
- `radius`: the radius of the snowflake. This value must range from `MIN_RADIUS` to `MAX_RADIUS` (inclusive).
- `strokeStyle`: the color in which the snowflake is drawn. Although all snowflakes are colored white, a future version of this application might vary the color among shades of gray to create a more 3D appearance. Snowflakes that are farther from an observer (i.e., smaller snowflakes) could look a bit grayer than snowflakes that are closer to the observer.
- `x`: snowflake horizontal origin. A snowflake is drawn relative to this horizontal origin.
- `y`: snowflake vertical origin. A snowflake is drawn relative to this vertical origin.
- `msInterval`: the application's `setInterval()` delay value. This value is used in a calculation that determines the speed at which snowflakes fall. Smaller snowflakes take longer to fall.

CoffeeScript offers a convenient shortcut for setting instance properties. Arguments passed to parameters that are prefixed with the `@` symbol (e.g., the constructor's six parameters) are automatically assigned to same-named instance properties. As a result, I don't need to specify `this.ctx = ctx` and make similar assignments for the other constructor parameters.

The constructor first saves the `x` and `y` parameter values in `startX` and `startY` instance properties. These values are saved so that the snowflake can be reset to its original position after falling to the ground (past the bottom of the canvas). As an exercise, introduce a random factor to horizontally reposition the snowflake after it passes the bottom of the canvas.

A `path` instance property is now introduced and initializes to an empty array. This property will contain the instructions for drawing a snowflake. These instructions are generated via CoffeeScript's "for loop combined with range" feature, which is responsible for drawing the snowflake's six branches. Each of the iterations generates an angle at which the branch is positioned, and invokes `snowflakeBranch` to draw the branch.

The remainder of this constructor calculates the value of an `incr` instance property, which is used by `draw` to vertically advance the snowflake. The calculation takes the snowflake's radius into account so that smaller snowflakes have smaller increments. The calculation also takes `msInterval` into account so that no snowflake falls too quickly, which would then be hard to view.

The `draw` instance method is very simple. It uses the saved `ctx` reference to assign the stroke style to the context, begins a path, loops over `path` (via `for element in @path`) to access each drawing instruction object, executes a `lineTo` or `moveTo` context operation based on the instruction's `cmd` value, closes and strokes the path, increments `y`, and resets the snowflake to its start position when it falls off the canvas.

Snowflake creation requires six calls to the `snowflakeBranch` method. Because this method is defined via indentation and = (so that it's private to `Snowflake`), it cannot directly access a `Snowflake` object's `path` instance property. To access `path`, I pass (in the constructor) `this` (a reference to the current `Snowflake` object) as the method's first argument.

The recursive `snowflakeBranch` method is invoked with `startX`, `startY`, `endX`, and `endY` values that define the start and end points of a line. This method first specifies `return if depth == 4` (almost everything is an expression in CoffeeScript) to specify a stopping point for the recursion. It then appends objects identifying lineto/moveto instructions for drawing this line to the array.

Continuing, `snowflakeBranch` calculates the line's center point (`cx` and `cy`) and uses this value to generate two smaller (and rotated) lines that add detail to the branch. Finally, these new values are recursively passed to `snowflakeBranch` by invoking this method for each of these new lines. The final value passed to `snowflakeBranch` defines the current depth, which is incremented until the stopping point is reached.

Learning More About CoffeeScript

I had a lot of fun creating this application, mainly because CoffeeScript makes it easier to write JavaScript code. If you would like to learn more about this amazing technology, you should check out the following resources:

- An Introduction to CoffeeScript (http://jspro.com/coffeescript/an-introduction-to-coffeescript/)
- Classes in CoffeeScript (http://jspro.com/coffeescript/classes-in-coffeescript/)
- CoffeeScript Cookbook (http://coffeescriptcookbook.com/)
- CoffeeScript website (http://coffeescript.org/)
- List Processing In CoffeeScript (http://jspro.com/coffeescript/list-processing-in-coffeescript/)
- The Little Book on CoffeeScript (http://arcturo.github.com/library/coffeescript/index.html)
- Unleash Your Inner Ninja with Jump Start CoffeeScript (http://jspro.com/coffeescript/unleash-your-inner-ninja-with-jump-start-coffeescript/)