

Below is a sample of code from Dan Hill's "Messy Web" final project for Interactive Multimedia Programming. With a major in computer science and minor in art Dan was one of my most advanced programming students along with his classmate Yanka Li also from the school of computer science. As a second time student of mine, Dan took advantage of this class to deeply explore Director's programming environment and its connectivity to the web and 3D. I only wish we had two semesters for him to work further on the graphical user interface. However, he is off to Microsoft for the summer, 2006 and then the Ph.D. program in Human Computer Interaction at the University of Maryland with Ben Shneiderman this fall.

```
-----
-- SPRITE SCRIPT
--
-- Controls converting the web to content.
--
-- We use a sprite as the intermediary for the web content.
--
-- TODO: move pXmlCount to the web service side
-----

-- TODO list:
-- * fix the camera rotation
-- * delete webpages that have no links and are out in no where
-- * implement graphed linking
-- * add a direction to each webpage
-- * have structures branch off in the new direction
-- * spatial navigation based on mouse direction
-- * place holder graphics
-- * implement graph focus
-- * implement navigation

-- * make http://www.google.com work
-- * make http://www.yahoo.com work
-- * TODO: remove ModelNum
-----

-- MODULAR PARTS
-----

-- The modular parts include:
-- * physics simulation - used as a method for controlling data
-- * graphics updates based on position
-----

-----

-- PHYSICS SIMULATION - Organizing lots of data
-----

-- A dampened physics simulation is an way to manipulate lots of data.
-- The visual web consists springs with attributes that vary based on
-- the perspective of the user.
--
-- The types of springs include:
-- * webpage links - controlling distance and sizes
-- * neighbor links - keeping the neighbor links away from each other
-----

-----

-- GRAPHICS ENGINE - Displaying prioritized data
-----

-- Use the camera's focus and orientation, a priority is placed on
-- webpages closest to the center and with relationships near the
-- viewing plane.
--
-- As objects move out of focus, the following decreases: the amount of
-- content in each page, the number of visible links and the size of the
-- content.
```

```

-----
-- GLOBAL VARIABLES
-----

global pWebpages -- list of visual webpages that are in the scene
global pLinks -- list of all the links on the screen
global pLinkToParent
global pLinkToChild
global pUrlToWebpage -- list of all the visible links
global pIsDynamicUpdateOn -- should we update the content
global pWebpagesToRemove

global pAttributeProp -- name of the attribute property in the property list
global pTextProp -- name of the text property in the property list
global pXmlCount -- count of number xml objects still out for request

-- specifies the update type when we are close to a webpage
-- values in {#preMultNormal,#preMultInverse, #multNormal, #multInverse}
property pPreFaceCamTransform

-- specifies the update type when we are close to a webpage
-- values in {#preMultNormal,#preMultInverse, #multNormal, #multInverse}
property pPostFaceCamTransform

property pFrameCount

-- member that represents the 3d shockwave object
property pScene

-----
-- HANDLERS
-----
-- list of functions:
-- * beginSprite
-- * keyDown
-- * exitFrame
-----

-- @main: Initializes lots of properties and creates the first object
on beginSprite me
  -- initialize the transform values with a random motion style
  --applyRandomMotionStyle()
  --
  -- initialize globals
  pAttributeProp = "!ATTRIBUTES"
  pTextProp = "!CHARDATA"
  pWebpages = []
  pLinks = []
  pLinkToParent = [:]
  pLinkToChild = [:]
  pUrlToWebpage = [:]
  pXmlCount = 0
  pIsDynamicUpdateOn = true
  pScene = member("3dWorld")
  pFrameCount = 1
  applyRandomMotionStyle()
  --
  -- initialize the web service
  initWebToXmlService()
  --
  -- TODO: create an empty blank webpage without any content
  theUrl = standardizeUrl(member("WebAddress").text)
  --
  --

```

```

-- request the first webpage
theWebpage = getWebpage_addIfNeeded(theUrl)
theWebpage.NumTotalLinks = 1
end

-- @main: for testing purposes, when we get a space bar, lets apply a new random
style
on keydown me
  if _key.key = SPACE then
    pIsDynamicUpdateOn = not pIsDynamicUpdateOn
    --applyRandomMotionStyle()
  end if
end

-- @main: Updates the visual sites and checks to see if any more web requests
-- need to be processed
on exitFrame me
  --checkMomentumSum()

  if pIsDynamicUpdateOn = false then
    return
  end if

  -- 1) Update the visual sites
  -- TODO: make time dynamic
  if pWebpages <> empty then
    me.updateSimulation(0.03)
  end if

  --
  -- 2) Run the xml web processes
  -- TODO: add in a wait timer for future webgets
  if pXmlCount <> 0 then

    -- process any webToXml events if they exist
    finishedList = xProcessWebToXmlEvents()

    -- if the content finishes
    if finishedList.count > 0 then
      repeat with theUrl in finishedList
        -- assign the prop list to the webpage
        webpage = pUrlToWebpage.GetProp(theUrl)
        myPropList = getXml(theUrl)
        webpage.XmlPropList = myPropList

        -- Break down the property list into a list of content
        webpage.SetProp(#Content, createContentList(myPropList, [], theUrl))

        -- decrement the number of xml counts
        pXmlCount = pXmlCount - 1
      end repeat
    else
      go to the frame
    end if
  end if
end

-----
-- Webpage Constructor and Destructor functions
-----

on getWebpage_addIfNeeded theUrl
  -- If the child link url exists, then get the reference
  theWebpage = pUrlToWebpage.GetAProp(theUrl)

  if theWebpage = void then
    theWebpage = createEmptyWebpage(theUrl)

```

```

        if isXmlGetStarted(theUrl) = false then
            -- get the new webpage
            getXmlFromWeb(theUrl)
            pXmlCount = pXmlCount + 1
        end if
    end if
end if

return theWebpage
end

-- @main: creates an empty visual webpage
on createEmptyWebpage theUrl
    webpage = [:]

    -- add a prop called VisualTree with a default empty property list
    webpage.AddProp(#VisualTree, [:])
    webpage.AddProp(#Url, theUrl)
    webpage.AddProp(#XmlPropList, [:])
    webpage.AddProp(#DisplayType, #Empty)
    webpage.AddProp(#Links, [])
    webpage.AddProp(#Content, [])
    webpage.AddProp(#NumVisibleContent, 0)
    webpage.AddProp(#NumChildLinks, 0)
    webpage.AddProp(#NumTotalLinks, 0)

    -- physics attributes
    webpage.AddProp(#Mass, 1.0)
    webpage.AddProp(#VelocityVec, vector(0,0,0))
    webpage.AddProp(#ForceVec, vector(0,0,0))

    -- the link is loading
    visualItem = CreateTextVisualItem(theUrl & " is loading", theUrl &
"+isloading", [:])
    -- Add to some position in the visual tree
    randomlyAddVisualItemToTree(webpage.GetProp(#VisualTree), visualItem)

    -- place it in the global tables
    pUrlToWebpage.AddProp(theUrl, webpage)
    pWebpages.append(webpage)

return webpage
end

-- @main: Adds a link to the visual webpage.
-- @details:
on AddLink theParent, theChild

    -- initialize the new link
    newLink = [:]
    pLinkToParent.AddProp(newLink, theParent)
    pLinkToChild.AddProp(newLink, theChild)
    --newLink.AddProp(#Parent, theParent)
    --newLink.AddProp(#Child, theChild)
    newLink.AddProp(#Ks, 1.0)
    newLink.AddProp(#Kd, 0.2)
    newLink.AddProp(#PreferredDirection, randomvector())
    newLink.AddProp(#DefaultLength, 200.0)
    newLink.AddProp(#springName, theParent.url & "^" & theChild.url & "^" &
string(random(1001)) )

    CreateVisualSpring(newLink.springName)

    -- increment the stats in the webpage
    theChild.NumTotalLinks = theChild.NumTotalLinks + 1
    theParent.NumTotalLinks = theParent.NumTotalLinks + 1
    theParent.NumChildLinks = theParent.NumChildLinks + 1

```

```

-- add the links to the appropriate lists
pLinks.append(newLink)

return newLink
end

-- @main removes the latest link
on DeleteLink theParent, theLink

    theChild = pLinkToChild.getProp(theLink)

    theChild.NumTotalLinks = theChild.NumTotalLinks - 1
    theParent.NumTotalLinks = theParent.NumTotalLinks - 1
    theParent.NumChildLinks = theParent.NumChildLinks - 1

    pLinks.deleteOne(theLink)
    pLinkToChild.deleteProp(theLink)
    pLinkToParent.deleteProp(theLink)

    DeleteVisualSpring(theLink.springName)

    if theChild.NumTotalLinks + theChild.NumChildLinks = 0 then
        pWebpagesToRemove.append(theChild)
    end if
    if theParent.NumTotalLinks + theParent.NumChildLinks = 0 then
        pWebpagesToRemove.append(theParent)
    end if
end

-- @main: remove the child links
-- @TODO: keep a cache full of the most frequently visited webpages
on removeWebpageFromScene theWebpage
    -- Keep removing the links until none are left
    repeat while theWebpage.length <> 0 then
        RemoveLatestLink(theWebpage)
    end repeat

    -- TODO: visually remove the item from the shockwave element
end

-----
-- Webpage Update functions
-----

-- returns the transform for a particular site
on getTransformOfVisualSite site
    theItem = site.GetProp(#VisualTree).GetAProp(#Item)
    if theItem <> void then
        return theItem.GetAProp(#Model).transform
    end if
end

on checkMomentumSum
    momentum = vector(0,0,0)
    repeat with webpage in pWebpages then
        momentum = momentum + webpage.velocityVec * webpage.mass
    end repeat
    --put momentum
end

-- For now, do a stupid linear blend (x axis = distance, y axis = num of visible
pieces)
--
-- max _____

```

```

--
--
-- 0 --- /
--      +   +
--      200 100
--
-- calculate the percent of the scale
on getOnScreenPercentage me, thePosition
  origin = sendsprite(1, #getOrigin, me)

  delta = thePosition - origin

  len = delta.length
  if len > 600.0 then
    percent = 0.0
  else if len < 300.0 then
    percent = 1.0
  else
    percent = (600.0 - len)/300.0
  end if

  return percent
end

on getMassFromPercentage percent
  return 1.0 + 2.0 * percent
end

-- @main: updates the visual contents on the webpages
-- TODO: remove small objects from the 3d scene
on updateVisualContentForWebpages me, theCameraOrigin

  pWebpagesToRemove = []

  -- for each visual site ...
  repeat with webpage in pWebpages
    -- initialize the webpage values
    --
    -- set force to 0
    webpage.setProp(#ForceVec, vector(0,0,0))

    percent = getOnScreenPercentage(me,
getTransformOfVisualSite(webpage).position)
    --
    -- adjust the mass based on the distance
    --
    mass = getMassFromPercentage(percent)
    webpage.velocityVec = webpage.velocityVec * (webpage.mass / mass)
    webpage.mass = mass
    theSize = 0.4 + 0.6 * percent
    theScale = theSize/webpage.VisualTree.Item.Model.Transform.scale.x
    webpage.VisualTree.Item.Model.Transform.scale(vector(theScale, theScale,
theScale))

    numShouldBeVisible = integer(percent * float(webpage.Content.count-1) + 1)

    if webpage.NumVisibleContent > numShouldBeVisible then
      -- remove the last item
      content = webpage.Content[webpage.NumVisibleContent - 1]

      if content.getaProp(#active) <> void then
        if content.getprop(#active) then
          if content.type = #text then
            DeleteTextVisualItem(webpage, content.getProp(#VisualItem))
            content.deleteProp(#VisualItem)
          else if content.type = #link then
            DeleteLink(webpage, content.link)
            content.deleteProp(#link)
          else if content.type = #img then
            DeleteImageVisualItem(webpage, content.getProp(#VisualItem))

```

```

        content.deleteProp(#VisualItem)
    end if
end if
content.deleteProp(#active)
end if

webpage.NumVisibleContent = webpage.NumVisibleContent - 1
else if webpage.NumVisibleContent < numShouldBeVisible then

content = webpage.Content[webpage.NumVisibleContent + 1]
webpage.NumVisibleContent = webpage.NumVisibleContent + 1

if content.type = #text then
visualItemName = webpage.Url & "^" & content.Text
if pScene.model(visualItemName) = void then
visualItem = CreateTextVisualItem(content.Text, visualItemName, [])
content.AddProp(#VisualItem, visualItem)
randomlyAddVisualItemToTree(webpage.GetProp(#VisualTree), visualItem)
content.AddProp(#active, true)
else
webpage.Content.deleteOne(content)
content.AddProp(#active, false)
end if

else if content.type = #link then
if pWebpages.count < 10 then
childWebpage = getWebpage_addIfNeeded(content.url)
link = AddLink(webpage, childWebpage)
content.addProp(#link, link)
parentTransform = getTransformOfVisualSite(webpage)
childTransform = getTransformOfVisualSite(childWebpage)
transVec = parentTransform.position + randomVector() * 150
childTransform.translate(transVec)
p = getOnScreenPercentage(transVec)
childWebpage.mass = getMassFromPercentage(me, p)

content.AddProp(#active, true)
else
content.AddProp(#active, false)
end if
else if content.type = #img then
visualItemName = webpage.Url & "^" & content.img
if pScene.model(visualItemName) = void then
visualItem = CreateImageVisualItem(content.img, visualItemName, [])
content.AddProp(#VisualItem, visualItem)
randomlyAddVisualItemToTree(webpage.GetProp(#VisualTree), visualItem)
content.AddProp(#active, true)
else
webpage.Content.deleteOne(content)
content.AddProp(#active, false)
end if
end if
end if

-- TODO: update the contents of the visual sites
-- find the delta vector
-- delta = getTransformOfVisualSite(visualSite).worldposition - origin
-- make the site bigger if it is too small
-- if delta.length < 100 and visualSite.GetProp(#Type) = #TooSmall then
-- shrink the site if it is too small
-- else if delta.length >= 100 and visualSite.GetProp(#Type) <> #TooSmall
then
-- end if

-- TODO: update the number of dependent links based on position

end repeat

momentum = vector(0,0,0)
repeat with webpage in pWebpagesToRemove then

```

```

pWebpages.deleteOne(webpage)
pUrlToWebpage.DeleteProp(webpage.url)
removeAllVisualItemsFromTree(webpage.VisualTree)

momentum = momentum + webpage.velocityVec * webpage.mass
end repeat

-- distribute the lost momentum
momentum = momentum / pWebpages.count
repeat with webpage in pWebpages
  webpage.velocityVec = webpage.velocityVec + momentum / webpage.mass
end repeat
end

on updateSpringPositions
  repeat with link in pLinks
    myModel = pScene.model(link.springName)

    myModel.transform = transform()

    theParent = pLinkToParent.getProp(link)
    theChild = pLinkToChild.getProp(link)
    parentOrigin = getTransformOfVisualSite(theParent).position
    childOrigin = getTransformOfVisualSite(theChild).position
    delta = parentOrigin - childOrigin
    len = delta.length()

    myModel.transform.translate(0, 0.5, 1)
    myModel.transform.scale(1, len, 1)

    -- TODO: handle vertical cases

    axis = vector(0,1,0).cross(delta)
    angle = vector(0,1,0).angleBetween(delta)

    myModel.transform.rotate(vector(0,0,0), axis, angle)

    myModel.transform.translate(childOrigin.x, childOrigin.y, childOrigin.z)
  end repeat
end

-- @main: calculates the spring forces for the webpages
on calcLinkForcesForWebpages
  -- loop through all springs
  -- if the force is infinity, make sure to print a warning
  repeat with link in pLinks

    theParent = pLinkToParent.getProp(link)
    theChild = pLinkToChild.getProp(link)
    parentOrigin = getTransformOfVisualSite(theParent).position
    childOrigin = getTransformOfVisualSite(theChild).position

    relativeVelocity = theParent.VelocityVec - theChild.VelocityVec

    -- TODO: optimize
    delta = parentOrigin - childOrigin
    len = delta.length

    -- if the length isn't 0, then calculate the force
    if len <> 0 then
      force = delta * (1/len) * (len - link.DefaultLength) * link.Ks +
relativeVelocity * link.Kd

      -- TODO: apply Kd

      -- Add delta to child's force

```

```

        theChild.ForceVec = theChild.ForceVec + force
        -- Add -delta to parent's force
        theParent.ForceVec = theParent.ForceVec - force
    end if
end repeat
end

on rotateLinkBasedOnCamera cameraOrigin
    repeat with link in pLinks
        theParent = pLinkToParent.getProp(link)
        theChild = pLinkToChild.getProp(link)
        parentOrigin = getTransformOfVisualSite(theParent).position
        childOrigin = getTransformOfVisualSite(theChild).position

        averageOrigin = parentOrigin * childOrigin / 2.0

        diff = cameraOrigin - averageOrigin
        diff = diff / 500

        force = link.preferredDirection + diff

        -- Add delta to child's force
        theChild.ForceVec = theChild.ForceVec + force
        -- Add -delta to parent's force
        theParent.ForceVec = theParent.ForceVec - force
    end repeat
end

on calcCameraForcesForWebpages deltaT, cameraOrigin, cameraLookVec
    totalForces = vector(0,0,0)

    repeat with webpage in pWebpages
        delta = cameraOrigin - webpage.VisualTree.Item.Model.worldPosition
        dotProduct = cameraLookVec.dot(delta)

        if dotProduct < 200.0 and dotProduct > -200.0 then
            theForce = dotProduct/50.0 * cameraLookVec
            webpage.ForceVec = webpage.ForceVec + theForce
            totalForces = totalForces + theForce
        end if
    end repeat

    totalForces = - totalForces / pWebpages.count

    repeat with webpage in pWebpages
        webpage.ForceVec = webpage.ForceVec + totalForces
    end repeat
end

-- @main: updates the locations of the webpages
-- @details: Euler integration (TODO: code Midpoint or RK4)
on updateLocationForWebpages deltaT
    -- For each webpage
    repeat with webpage in pWebpages
        -- Apply the Force to the Velocity
        webpage.VelocityVec = webpage.VelocityVec + webpage.ForceVec / webpage.mass
    * deltaT
        --
        -- Apply the Velocity to the position
        t = getTransformOfVisualSite(webpage)
        if t <> void then
            t.translate(webpage.VelocityVec * deltaT)
        end if
    end repeat
end
end

```

```

on updateRotation me, deltaT, origin
  t = pScene.camera(1).transform.duplicate()

  repeat with webpage in pWebpages
    t.translate(-t.position.x, -t.position.y, -t.position.z)
    delta = getTransformOfVisualSite(webpage).position - origin
    delta = delta * (1.0/100.0)

    subUpdatePosition(me, webpage.visualTree, deltaT, delta.length, t)
  end repeat
end

-- @main: updates the simulation over time
-- @details: uses the deltaT as the time increment since the last update.
--           Adds and subtracts detail based on positions.
on updateSimulation me, deltaT
  -- get the origin
  origin = sendsprite(1, #getOrigin, me)

  cameraVec = pScene.camera(1).worldPosition - origin
  cameraVec.normalize()

  updateVisualContentForWebpages(me, origin)

  calcLinkForcesForWebpages()

  rotateLinkBasedOnCamera(origin)

  calcCameraForcesForWebpages(deltaT, origin, cameraVec)

  updateLocationForWebpages(deltaT)

  updateSpringPositions()

  updateRotation(me, deltaT, origin)
end

-----
-- Style
-----

-- @main: applies a random motion style to the webpages
on applyRandomMotionStyle
  pPreFaceCamTransform = getRandomTransformStyle()
  pPostFaceCamTransform = getRandomTransformStyle()
end

on getRandomTransformStyle()
  case random(4) of
    1: return #preMultNormal
    2: return #preMultInverse
    3: return #multNormal
    4: return #multInverse
  end case
end

-- applies a particular style to the camera transformation
on applyTransformStyle theStyle, camTransform, modelTransform
  case theStyle of
    #preMultNormal : camTransform.premultiply(modelTransform)
    #preMultInverse : camTransform.premultiply(modelTransform.inverse())
    #multNormal : camTransform.multiply(modelTransform)
    #multInverse : camTransform.multiply(modelTransform.inverse())
  end case
end

```

```

-----
-- Url Methods
-----

-- @main: find a the directory the url is contained in
on findDirectory theUrl
  l = theUrl.length
  c = theUrl.char[1-3..1]
  if theUrl.char[1-3..1] = "html" or theUrl.char[1-2..1] = "htm" or
theUrl.char[1-2..1] = "php" or theUrl.char[1-2..1] = "cgi" then
    len = theUrl.length
    repeat with i = 1 to len
      if theUrl.char[len - i] = "/" then
        directoryUrl = theUrl.char[1..(len - i)]
        return directoryUrl
      end if
    end repeat
    return ""
  end if
  if theUrl.char[1] = "/" then
    return theUrl
  else
    return theUrl & "/"
  end if
end

-- @main: an attempt to standardize urls
-- TODO: debug this
on standardizeUrl theUrl
  -- TODO: remove @ out of links
  -- TODO: add "index.html"

  -- Remove the initial "http://" if it exists
  if theUrl contains "http://" then
    theUrl = theUrl.char[8..theUrl.length]
  end if
  --
  -- Fold any ../
  i = offset("../", theUrl)
  repeat while i <> 0 then
    -- if "../" is the first string, then remove it and try again
    if i = 1 then
      theUrl = theUrl.char[i+3..theUrl.length]
    else
      -- remove the previous directory
      theUrl = findDirectory(theUrl.char[1..i-1]) &
theUrl.char[i+3..theUrl.length]
    end if
    i = offset("../", theUrl)
  end repeat
  --
  -- Remove any ./
  i = offset("./", theUrl)
  repeat while i <> 0 then
    theUrl = theUrl.char[1..i] & theUrl.char[i+2..theUrl.length]
    i = offset("./", theUrl)
  end repeat
  --
  -- Make everything lower case
  -- repeat with i = 0 to theUrl.length
  --   charNum = charToNum(theUrl.char[i])
  --   if charNum > 64 and charNum < 91 then
  --     theUrl.char[i] = charNum + 32
  --   end if
  -- end repeat
  --
  -- return the url
  return "http://" & theUrl
end

```

```

-- @main: creates a link between two webpages
on standardizeChildUrl parentUrl, childUrl
-- if child address contains "http://", then return the childUrl

if childUrl.char[1..7] = "http://" or childUrl.char[1..3] = "www" then
    return standardizeUrl(childUrl)
end if
--
if childUrl.char[1] = "/" then
    childUrl = childUrl.char[2..childUrl.length]
end if

--
-- else ...
return standardizeUrl(findDirectory(parentUrl) & childUrl)
end

-----
-- UNORGANIZED SECTIONS
-----

-----
-- TODO: add back in the style function I deleted
-----

-- requests all of the linked webpages
--on requestLinkedWebPages(myPropList, url)
-- -- create links
-- L = FindAllLinks(myPropList, [])
--
-- -- find base url
-- t = url.length
-- repeat with i = 1 to t
--     if url.char[t - i] = "/" then
--         baseurl = url.char[1..(t - i)]
--         i = t
--     end if
-- end repeat
--
-- repeat through the links found
-- repeat with link in L
--     -- if the link does not contain http,
--     -- then append it to the base url
--     if link.contains("http") = false then
--         link = baseurl & link
--     end if
--
--     -- check to see if the new webpage exists
--     if isXmlGetStarted(link) = false then
--
--         -- get the new webpage
--         getXmlFromWeb(link)
--         pXmlCount = pXmlCount + 1
--     end if
--
-- end repeat
--end

-- TODO: evaluate if this necessary
-- create random transforms in the visual tree
--on createRandomTransformsInVisualTree root
-- theItem = root.GetaProp(#Item)
-- if theItem <> void then
--     theModel = theItem.GetaProp(#Model)
--     randomV = randomVector() * 100.0
--     theModel.transform.position = randomV

```

```

--
--     repeat with theChild in root.GetProp(#Children)
--         createRandomTransformsInVisualTree theChild
--     end repeat
--
-- end if
--end

--a sub update function that gets called recursively
on subUpdatePosition me, root, deltaT, radius, faceCamTransform
    theModel = root.GetAProp(#Item).GetAProp(#Model)
    if theModel <> void then
        t = theModel.transform
        p = t.position

        -- update the position
        --     if radius < 2.0 and radius <> 0 then
        --         lenSqrdd = p.x * p.x + p.y * p.y + p.z * p.z
        --         if lenSqrdd <> 0 then
        --             p = p * 200.0 * 200.0 / radius / radius / lenSqrdd
        --         end if
        --     end if

        -- update the rotation
        if radius < 1.0 then
            faceCamTransform.position = p

            -- interpolate the new position
            t.InterpolateTo(faceCamTransform, 100 * (1.0 - radius))

            -- modify faceCamTransform for next level
            applyTransformStyle(pPreFaceCamTransform, faceCamTransform, t)

            else -- rotate more or less depending on distance
                t.rotate(p, root.GetProp(#TransAxisOfRotation), radius * radius * deltaT *
root.GetProp(#TransVelocity))
            end if

            -- repeat on children
            repeat with theChild in root.GetProp(#Children)
                me.subUpdatePosition(theChild, deltaT, radius, faceCamTransform)
            end repeat

            if radius < 1.0 then
                -- modify faceCamTransform for next level
                applyTransformStyle(pPostFaceCamTransform, faceCamTransform, t)
            end if
        end if
    end

on removeVisualItemFromTreeNode root, visualItem
    if root.getAProp(#Children) <> void then
        repeat with theChild in root.getProp(#Children)
            if theChild.getProp(#Item) = visualItem then
                root.getProp(#Children).deleteOne(theChild)
                return #Done
            end if
            if removeVisualItemFromTreeNode(theChild, visualItem) = #Done then
                return #Done
            end if
        end repeat
    end if
    return #NotDone
end

on removeVisualItemFromTree webpage, visualItem
    theResult = removeVisualItemFromTreeNode(webpage.GetProp(#VisualTree),
visualItem)
end

```

```

-- randomly adds a visual item into a visual tree
on randomlyAddVisualItemToTree root, visualItem
-- if root doesn't contain a level, create a new one and return it
if root.GetProp(#Item) = void then
    root.AddProp(#Item, visualItem)
    root.AddProp(#TransAxisOfRotation, randomVector())
    root.AddProp(#TransVelocity, 6-random(11))
    root.AddProp(#Children, [])
else
    -- Otherwise, count the number of visualtrees in the current level.
    -- Pick a random number between 1 and the number of items in the
    -- current tree + 1.
    theLength = root.GetProp(#Children).length
    i = random(theLength + 1)

    -- randomly add to the tree of the number, if the random number
    -- indicates a tree value
    if i < theLength + 1 then
        -- add it to the visual tree
        randomlyAddVisualItemToTree(root.GetProp(#Children)[i], visualItem)
    else -- otherwise, add to the end of the children list
        theChild = [:]
        -- add it to the visual tree
        randomlyAddVisualItemToTree(theChild, visualItem)
        visualItem.model.translate(root.Item.Model.worldPosition + randomVector())
* 50)
        root.GetProp(#Children).append(theChild)
        root.GetProp(#Item).GetProp(#Model).addChild(visualItem.GetProp(#Model))

    end if
end if
end

on removeAllVisualItemsFromTree root
    repeat with child in root.Children
        removeAllVisualItemsFromTree(child)
    end repeat

    pScene.deleteModel(root.Item.Model.Name)
    root.deleteProp(#Item)
    root.deleteProp(#TransAxisOfRotation)
    root.deleteProp(#TransVelocity)
    root.deleteProp(#Children)
end

-----
-- Content Creation Functions
-----

-- @main: parses the proplist and makes a list of content
-- @args: myPropList - an xml property list
--        contentList - a list of content to append the new property list to
on createContentList myPropList, contentList, theUrl
    -- TODO: find a better ordering for the content
    --
    -- TODO: parses the list of images
    --
    -- parses the list of text
    textList = FindAllSubTextInTags(myPropList, "body", [])
    --
    -- parse the links
    linkList = FindAllLinks(myPropList, [])
    --
    -- parse the images
    imgList = FindAllImages(myPropList, [])

    textListIndex = 1
    linkListIndex = 1
    imgListIndex = 1
    choices = []
    if (textListIndex <= textList.count) then

```

```

    choices.add(#text)
end if
if linkListIndex <= linkList.count then
    choices.add(#link)
end if
if imgListIndex <= imgList.count then
    choices.add(#img)
end if

choice = void

repeat with i = 1 to textList.count + linkList.count + imgList.count then
    content = []
    choice = choices[random(choices.count)]

    if choice = #text then -- the next content is text
        content.AddProp(#type, #text)
        content.AddProp(#text, textList[textListIndex])
        textListIndex = textListIndex + 1
    else if choice = #link then
        content.AddProp(#type, #link)
        content.AddProp(#url, standardizeChildUrl(theUrl,
linkList[linkListIndex]))
        linkListIndex = linkListIndex + 1
    else if choice = #img then
        content.AddProp(#type, #img)
        content.AddProp(#img, standardizeChildUrl(theUrl, imgList[imgListIndex]))
        imgListIndex = imgListIndex + 1
    end if

    contentList.append(content)

    if choice = #text and textListIndex > textList.count then
        choices.deleteOne(#text)
    else if choice = #link and linkListIndex > linkList.count then
        choices.deleteOne(#link)
    else if choice = #img and imgListIndex > imgList.count then
        choices.deleteOne(#img)
    end if

    if i > 20 then
        return contentList
    end if
end repeat

--
-- Return the final content list
return contentList
end

on DeleteTextVisualItem webpage, visualItem
    i = visualItem.getProp(#ModelNumber)
    removeVisualItemFromTree(webpage, visualItem)
    if visualItem.name = "page2.html+isLoading" then
        put "debug"
    end if

    pScene.deleteModel(visualItem.getProp(#Name))
    --visualItem.getProp(#Model).removeFromWorld()
    -- pScene.deleteShader(visualItem.getProp(#Name))
end

on CreateTextVisualItem theText, visualName, visualItem
    -- create the texture from the text
    -- #texture:
    member("TextRef").text = theText
    member("ImageRef").image = member("TextRef").image

    myTexture = sendsprite(1, #createTextureFromText, pScene, visualName,
member("TextRef").text)

    img = member("ImageRef")

```

```

visualItem.addProp(#texture, img)
visualItem.addProp(#Name, visualName)

-- check to see if the modelling resource has been created of a certain
dimension
heightRatio = myTexture.height
widthRatio = myTexture.width

if heightRatio > widthRatio then
    heightRatio = heightRatio / widthRatio
    widthRatio = 1
else
    widthRatio = widthRatio / heightRatio
    heightRatio = 1
end if

planeResource = sendsprite(1, #createOfFindPlaneResource, widthRatio,
heightRatio)

-- create a shader
-- #shaders:
visualItem.addProp(#shader, sendsprite(1, #createStandardShader, img,
visualName, visualName))

-- #modelType:
myModelType = #box
visualItem.addProp(#modelType, myModelType)

-- #model:
-- place a new model into the scene
pScene.newModel(visualName, planeResource)
visualItem.addProp(#modelName, pScene.model.count)

myModel = pScene.model(visualName)
addProp(visualItem, #model, myModel)

-- attach shader to model
sendsprite(1, #attachShaderToModel, myModel, myModelType,
visualItem.getProp(#shader))

return visualItem
end

on DeleteImageVisualItem webpage, visualItem
    removeVisualItemFromTree(webpage, visualItem)
    pScene.deleteModel(visualItem.getProp(#Name))
end

-- simple link content creation
on CreateImageVisualItem img, visualName, visualItem

    -- create the texture from the text
    -- #texture:

    imgPath = the moviePath & img
    downloadNetThing(img, imgPath)

    -- import into the imageref member and rename it back to imageref
    m = member("ImageRef")
    member("ImageRef").importFileInto(imgPath)
    member("ImageRef2").duplicate(m.number)
    m.name = "ImageRef"

    myTexture = sendsprite(1, #createTextureFromImage, pScene, visualName)

    img = member("ImageRef")
    visualItem.addProp(#texture, img)
    visualItem.addProp(#Name, visualName)

```

```

-- check to see if the modelling resource has been created of a certain
dimension
heightRatio = myTexture.height
widthRatio = myTexture.width

if heightRatio > widthRatio then
    heightRatio = heightRatio / widthRatio
    widthRatio = 1
else
    widthRatio = widthRatio / heightRatio
    heightRatio = 1
end if

planeResource = sendsprite(1, #createOfFindPlaneResource, widthRatio,
heightRatio)

-- create a shader
-- #shaders:
visualItem.addProp(#shader, sendsprite(1, #createStandardShader, img,
visualName, visualName))

-- #modelType:
myModelType = #box
visualItem.addProp(#modelType, myModelType)

-- #model:
-- place a new model into the scene
pScene.newModel(visualName, planeResource)
myModel = pScene.model(visualName)
addProp(visualItem, #model, myModel)

-- attach shader to model
sendsprite(1, #attachShaderToModel, myModel, myModelType,
visualItem.getProp(#shader))

return visualItem

end

on DeleteVisualSpring springName
    pScene.deleteModel(springName)
end

on CreateVisualSpring springName
    springResource = sendsprite(1, #createOrFindSpringResource, springName)

    myShader = sendsprite(1, #createPlainShader, springName)

    pScene.newModel(springName, springResource)
    myModel = pScene.model(springName)
    sendsprite(1, #attachShaderToModel, myModel, #spring, myShader)

end

-----
-- (Xml Property List -> Result) Handlers
-----

-- Finds all text inside of a property list form XML node
-- will give all sub text
on FindAllSubText propList, childList
    repeat with i = 1 to propList.count then
        -- if the prop is a text property, copy the text
        if getPropAt(propList, i) = pTextProp then
            childList.append(propList[i])
        -- if the property is not an attribute, recurse in it
        else if getPropAt(propList, i) <> pAttributeProp then
            childList = FindAllSubText(propList[i], childList)
        end if
    end repeat
end

```

```

    end if
  end repeat
  return childList
end

-- finds all subtext inside of the tags specified
on FindAllSubTextInTags propList, tag, results
  -- find all parent tags of name tag
  L = FindAllParentElements(propList, tag, [])

  -- go through each resulting property list and search it for text
  repeat with i = 1 to L.count then
    results = FindAllSubText(L[i], results)
  end repeat

  return results
end

-- find all images in the property list
on FindAllImages propList, results
  imgList = FindAllParentElements(propList, "img", [])

  repeat with img in imgList then
    results = FindAttribute(img, "src", results)
  end repeat

  return results
end

-- find all links in the property list
on FindAllLinks propList, results
  -- find all parent tags of name tag
  linkList = FindAllParentElements(propList, "a", [])

  repeat with link in linkList then
    results = FindAttribute(link, "href", results)
  end repeat

  return results
end

-----
-- Property List algorithm notes
-- List of key adjectives for functions:
-- * Element = Xml Element
-- * Parent = upper node possible
-- * Sub = recursively down to text
-----

-----
-- Node Algorithms - property list -> property list
-----

-- Finds all elements of a certain tag
-- Xml represented in property list form
--
-- i.e. s = <html><body><p>test1</p><p>test2</p></body></html>
-- FindAllChildElements(s, "p", []) = [test1, test2]
-- FindAllChildElements(s, "body", []) = [<p>test1</p>, <p>test2</p>]
on FindAllParentElements propList, tag, childList
  -- for each child in the propList
  repeat with i = 1 to propList.count then
    if getPropAt(propList, i) = tag then
      childList.append(propList[i])
      -- otherwise, see if we want to run recursively
    else if listP(propList[i]) then
      childList = FindAllParentElements(propList[i], tag, childList)
    end if
  end repeat
end repeat

```

```
    return childList
end

-- finds a given attribute in a property list
on FindAttribute propList, attribute, results
  repeat with i = 1 to propList.count then
    if getPropAt(propList, i) = pAttributeProp then
      attribList = propList[i]

      repeat with j = 1 to attribList.count
        if getPropAt(attribList, j) = attribute then
          results.append(attribList[j])
        end if
      end repeat

    end if
  end repeat

  return results
end
```