“Creatures” Tutorial – Part I

Over and over, I have been asked to write a tutorial on how to make the various little creatures that appear on www.bit-101.com. I wasn’t sure where to even begin, as these things just kind of evolve out of a simple idea and slowly take shape. But I’m going to give it a shot. I’ll break it into a couple (or more) different parts to keep it digestible. The first part will go into making a body and legs and making these parts react to each other and the environment. The second part will go into methods of making the creatures as a whole come alive and react to user control, or autonomously. Just writing that sounds intimidating, but it’s really not that bad.

First off, you should be very, very familiar with springs as described in my “Elasticity Tutorial”. If you haven’t mastered that, don’t even bother with this one. These creatures rely almost completely on elasticity for their motion and reaction, and I’m going to assume you have a command over that. You should also be completely familiar with how to make a movie clip move, react to gravity and bounce, as covered in my “Gravity Tutorial”. Again, if you haven’t read that (or at least know those concepts well from some other source) please do so first.

Next, this is going to be written with Flash MX ActionScript. There are so many advantages to this, and I’ve gotten so used to it, I can’t go back to doing things in Flash 5 syntax. If you’re not using MX yet, sorry. If someone feels up to converting this code to Flash 5, feel free.

One last note, some of these lines end up longer than the available width and I had to wrap them around. Hopefully it is obvious when this happens. If you don’t see a semicolon on the end of the line, and the next line is indented, that’s a clue.

OK, here we go…

1. We’re going to take a quick brush up review on springs (called “elasticity” in my last tutorial). My definition of a spring: A force that pulls an object towards a certain point. The further the distance from that object to that point, the stronger the force. Let’s see it in action. Make a small movie clip, just a filled circle or something. Put it on stage, name the instance *foot_mc*.

2. We’ll make the object draggable. This is so cool and easy to do in MX. No more invisible buttons. Just create a *doDrag()* and a *noDrag()* function and set the onPress, on Release, and onReleaseOutside handlers of the object to those functions. The functions just call a startDrag() and stopDrag() and set a variable *dragging* which we will use later. The following code goes right in frame one of the main time line.

   ```
   foot_mc.onPress=doDrag;
   foot_mc.onRelease=noDrag;
   foot_mc.onReleaseOutside=noDrag;
   function doDrag(){
      this.startDrag();
      this.dragging=true;
   }
   function noDrag(){
      stopDrag();
      this.dragging=false;
   }
   ```

   There you have it, the movie clip is now draggable.

3. Now let’s add the spring. First we need a spring factor, a variable we’ll call “k”. We’ll set it to 0.2. We’ll also set a damping or friction factor to slow things down so we don’t spring around forever. Finally, we’ll define a point to spring to, centerX, centerY. This code goes right at the top of the file.

```
k = .2;
damp = .9;
centerX = 270;
centerY = 200;
```

4. Now we’ll add the spring function to the onEnterFrame handler of *foot_mc*.

```
foot_mc.onEnterFrame = spring;
function spring() {
    if (!this.dragging) {
          var dx = centerX-this._x;
          var dy = centerY-this._y;
          this.vx += dx*k;
          this.vy += dy*k;
          this.vx *= damp;
          this.vy *= damp;
          this._x += this.vx;
          this._y += this.vy;
    } else {
          this.vx = this._x-this.oldx;
          this.vy = this._y-this.oldy;
          this.oldx = this._x;
          this.oldy = this._y;
    }
}
```

Now that’s a mouthful, but if you have gone through the gravity and elasticity tutorials (like I told you to!!!) this should all seem pretty familiar.

First we check if we are dragging the clip. If so, we don’t want to apply the spring formula, so we check if(!this.dragging) which means, “if NOT dragging”.

If we’re OK there, we find the distance (in x and y) from the center point to this movie clip (this.dx, this.dy).

Then we add a fraction of that distance to the x and y velocities (this.vx and this.vy). We do that by multiplying the distance by k.

Then we apply our damping by multiplying the velocity by damp.

Then we add the velocity to the current position.

If we ARE dragging the clip, we find out how many pixels it is moving each frame by subtracting its old position from its current position. That becomes its new velocity.

Again, everything here has been covered in the earlier tutorials, but I just wanted to run through it with the new syntax and make sure we're all on the same page. Now you should have your foot springing towards the center point. You can drag it around and throw it as well.

You might notice that dx and dy are defined with "var" This means they will only exist for the duration of the function, and will then be destroyed. That's fine because they will be completely recalculated on each frame. vx and vy, however, are assigned to "this", the object that called the function (foot_mc). That's because their values need to persist. They will be added to or subtracted to on each frame, so we need to preserve their values.

5. So far, we've just sprung to a stationary point. Now let's slowly build up the complexity. We'll start by springing to a point that can be moved. Make a second movie clip, instance named *body_mc*. We'll make it draggable as well, and have the foot spring to the coordinates of *body_mc* instead of a predetermined center point. Here's the code in full:

```
k = .2;
damp = .9;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
foot_mc.onPress = doDrag;
foot_mc.onRelease = noDrag;
foot_mc.onReleaseOutside = noDrag;
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
foot_mc.onEnterFrame = spring;
function spring() {
   if (!this.dragging) {
         var dx = body_mc._x-this._x;
         var dy = body_mc._y-this._y;
         this.vx += dx*k;
         this.vy += dy*k;
         this.vx *= damp;
         this.vy *= damp;
         this._x += this.vx;
         this._y += this.vy;
   } else {
         this.vx = this._x-this.oldx;
         this.vy = this._y-this.oldy;
         this.oldx = this._x;
         this.oldy = this._y;
   }
}
```

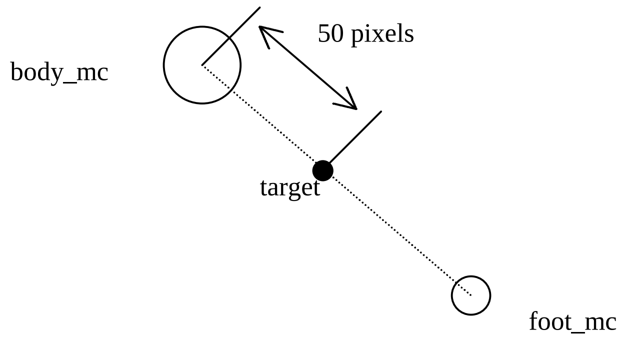
You can now drag around the body movie clip, and the foot will continue to spring towards it, wherever it is. Let's keep adding.

6. One problem you might notice is that the foot will eventually come to a stop directly on top of the body clip. Depending on the shape of your clips and their depths, one might completely cover the other. Here, we'll change the point that the foot is springing to (I call it the target point). Rather than using *body_mc*'s coordinates directly, we'll add a value to them, say 50 pixels on the _x. This will give the foot a target slightly off to the right of the center clip. We'll just change the spring function to the following:

```
function spring() {
  if (!this.dragging) {
       var targetX = body_mc._x+50;
       var targetY = body_mc._y;
       var dx = targetX-this._x;
       var dy = targetY-this._y;
       this.vx += dx*k;
       this.vy += dy*k;
       this.vx *= damp;
       this.vy *= damp;
       this._x += this.vx;
       this._y += this.vy;
  } else {
       this.vx = this._x-this.oldx;
       this.vy = this._y-this.oldy;
       this.oldx = this._x;
       this.oldy = this._y;
  }
}
```

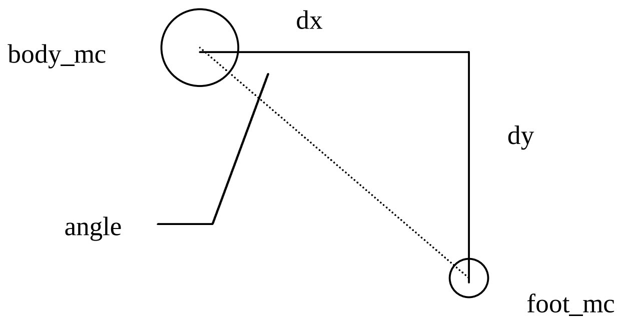
Here we've just created a *targetX* and *targetY* variable rather than springing directly to the body clip's coords. We use the target point to cause the object to spring. What you should see now is the foot springing and coming to rest just to the right of the body.

7. Ready for a quantum leap? OK. It was easy enough to make the target slightly to the right of the body. It would be just as easy to make it to the left by subtracting an amount from _x, or making it above or below by changing the _y. But what I want to do is have the target point 50 pixels from the body clip, *in the direction of the foot itself.* In other words, if the foot is to the right of the body, the target point should be 50 pixels to the right. If it's below, the target should be 50 pixels below the body. And here's the clincher: if the foot is at an angle of say, 45 degrees from the body, the target point should be 50 pixels out from the body, at an angle of 45 degrees. Here, a picture will hopefully show it better than I can say it.



No matter where I move that foot (or no matter where it moves on its own), I want to compute its target point to be 50 pixels from the body, in the direction towards the foot. To do this, we'll need to know a little trigonometry. Hey! Come back here! It's not that bad.

First we need to know the angle between body and foot. Here's another nifty diagram:



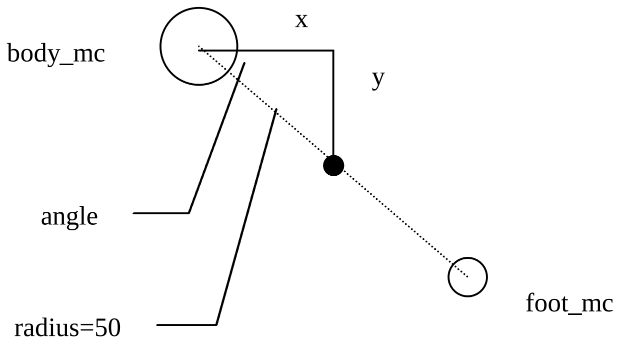
Here we have formed a right triangle between foot and body. To find the angle specified, we can use the Flash trig function, Math.atan2. This function takes two arguments and returns the value of the angle. The arguments are the length of the side *opposite* to the angle, and the length of the side *adjacent*, or next to, the angle. In our diagram, dy represents the length of the side opposite the angle, and dx is the length of the adjacent side. See that? So, to get the angle, we say:

```
this.angle = Math.atan2(dy, dx);
```

(Obviously this would have to go *after* the line in which we calculate dx and dy.)

Also, you should know that atan2 returns an angle expressed in "radians". This is not the same as degrees. Actually, one radian is equal to about 59 point something degrees. At this point we don't need to worry about it, because we will just be using this angle to feed back to more trig functions, which also use radians. We don't even need to see or ever know what this number is. We just store it in a variable and use it in another statement. Later, we will need to use the angle to control the rotation of an object. Unfortunately, the _rotation property of movie clips uses degrees, not radians! So at that point we will have to learn how to convert them.

8. Now that we have an angle, and a radius (50), it is pretty easy trigonometry to get the x and y values of that point using cos and sin. Taking a look at the next diagram:



We can find x and y by saying:

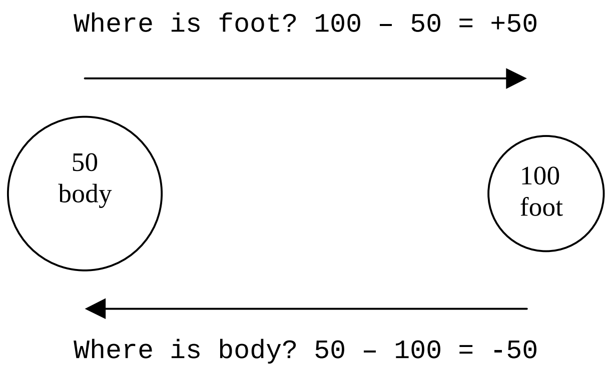
```
x=Math.cos(angle)*radius;
y=Math.sin(angle)*radius;
```

Now, realize that the x and y is only in relation to the body_mc’s coordinates. If we want screen coordinates, we have to add the x and y to the body’s _x and _y.

Here’s the final code for this one:

```
radius = 50;
k = .2;
damp = .9;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
foot_mc.onPress = doDrag;
foot_mc.onRelease = noDrag;
foot_mc.onReleaseOutside = noDrag;
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
foot_mc.onEnterFrame = spring;
function spring() {
   if (!this.dragging) {
        var dx = this._x-body_mc._x;
        var dy = this._y-body_mc._y;
        var angle = Math.atan2(dy, dx);
        var targetX = body_mc._x+Math.cos(angle)*radius;
        var targetY = body_mc._y+Math.sin(angle)*radius;
        this.vx += (targetX-this._x)*k;
        this.vy += (targetY-this._y)*k;
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
```

You see there are only a few changed lines. First I needed to reverse the order of subtraction to get dx and dy. Before we were asking, "Where is the body in relation to the foot?" So we subtracted foot from body. Now we want to know where foot is, in relation to body, so we reverse it. If we didn't do that, the angle would come out 180 backwards and mess everything up.



Next, we declare our radius variable. Then we find the angle using dx and dy with Math.atan2. Then we calculate our targetX and targetY using the sin and cos of angle times the radius, added to body_mc's coords. The next line uses a bit of a short-cut. We just subtract the foot's _x from targetX and multiply by k to get the acceleration factor.

Test this out. You can drag and throw as before and the same springing action is there, but the foot will always spring to a point 50 pixels away from the body. It doesn't care what direction, just that it's 50 pixels. Please take your time and make sure you really understand what's going on here. This is so vital to making the parts of a system react to each other. If you don't get this, you'll be completely lost when we double the complexity next, and quadruple it a little later.

9. OK, so far we have one relatively stationary object and one springing object. Now we are pretty much following the steps of how I originally developed this stuff originally. (Not that I'm saying I am the first one to ever to this stuff, but these were the ideas I came up with by myself while experimenting.) My next thought when I had gotten this far was, "Well, how about if object 1 springs to a point 50 pixels away from object 2, and object 2 springs to a point 50 pixels away from object 1?" Actually, that kind of concept had been going on in my mind for a few months, but I always considered it way too complex to attempt coding. But when I discovered springs and Hooke's Law in Jobe Makar's chapter of *Macromedia Flash Super Samurai* (if you don't have this book, rush out and buy it now), and then figured out how to do the offset thing as above, I realized that it would be as simple as giving the same formula to both objects. We just take the actions we applied to foot – having it spring towards body, and apply that to body – having it spring toward foot. Here's the code:

```
radius = 50;
k = .2;
damp = .9;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
foot_mc.onPress = doDrag;
foot_mc.onRelease = noDrag;
foot_mc.onReleaseOutside = noDrag;
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
foot_mc.onEnterFrame = footSpring;
body_mc.onEnterFrame = bodySpring;
function footSpring() {
   if (!this.dragging) {
         var dx = this._x-body_mc._x;
         var dy = this._y-body_mc._y;
         var angle = Math.atan2(dy, dx);
         var targetX = body_mc._x+Math.cos(angle)*radius;
         var targetY = body_mc._y+Math.sin(angle)*radius;
         this.vx += (targetX-this._x)*k;
         this.vy += (targetY-this._y)*k;
         this.vx *= damp;
         this.vy *= damp;
         this._x += this.vx;
         this._y += this.vy;
   } else {
         this.vx = this._x-this.oldx;
         this.vy = this._y-this.oldy;
         this.oldx = this._x;
         this.oldy = this._y;
   }
}
function bodySpring() {
   if (!this.dragging) {
         var dx = this._x-foot_mc._x;
         var dy = this._y-foot_mc._y;
         var angle = Math.atan2(dy, dx);
         var targetX = foot_mc._x+Math.cos(angle)*radius;
         var targetY = foot_mc._y+Math.sin(angle)*radius;
         this.vx += (targetX-this._x)*k;
         this.vy += (targetY-this._y)*k;
         this.vx *= damp;
         this.vy *= damp;
         this._x += this.vx;
         this._y += this.vy;
   } else {
         this.vx = this._x-this.oldx;
         this.vy = this._y-this.oldy;
         this.oldx = this._x;
         this.oldy = this._y;
   }
}
```

Here, instead of having just a spring function, we have two functions, bodySpring and footSpring. Each one is assigned to the onEnterFrame handler of its corresponding movie clip. The only difference between the two is that footSpring specifies body_mc as its target, and bodySpring specifies foot_mc as a target.

Now I know, this is horrible coding practice, to have two separate functions which are 99% identical. I just wanted to do it this way for clarity. You can plainly see (hopefully) that foot is springing to body, and at the same time, body is springing to foot. Later, these functions will evolve anyway and be more distinct.

What you should have now is a free-floating entity. You can drag either piece and the other will spring towards it. When you let go, they will spring towards and away from each other until they find their own equilibrium. I was pretty amazed when I first made this and it actually worked. I really didn't expect it to.

One note. There is nothing to stop these body parts from flying off stage just yet. If their original placement is far away from each other, they may snap towards each other with a lot of force and eventually drive each other off stage. Either place them relatively close to begin with, or reduce your damping factor to add enough friction to keep them from going far.

10. Next, we'll just add a minor visual enhancement. We have a body and a foot, so where's the leg??? Just throw this block of code at the end of the file to use the drawing API to draw a line between the two:

    ```
    _root.onEnterFrame = function() {
       _root.clear();
       _root.lineStyle(1, 0, 100);
       _root.moveTo(body_mc._x, body_mc._y);
       _root.lineTo(foot_mc._x, foot_mc._y);
    };
    ```

    This doesn't give any additional functionality, merely makes the connection between the two objects more visible.

11. OK. I said we were going to double, then quadruple the complexity. So, let's add another leg! First of all, change your foot movie clip's instance name to *foot0_mc*. Now copy it, paste another instance of it on stage, and name the new instance, *foot1_mc*.

    Now, go back to the line that reads:

    ```
    foot_mc.onEnterFrame = footSpring;
    ```

    and change it to assign the footSpring function to both of our feet.

    ```
    foot0_mc.onEnterFrame = footSpring;
    foot1_mc.onEnterFrame = footSpring;
    ```

    Then, take the lines:

    ```
    foot_mc.onPress = doDrag;
    foot_mc.onRelease = noDrag;
    foot_mc.onReleaseOutside = noDrag;
    ```

    and make them:

```
foot0_mc.onPress = doDrag;
foot0_mc.onRelease = noDrag;
foot0_mc.onReleaseOutside = noDrag;
foot1_mc.onPress = doDrag;
foot1_mc.onRelease = noDrag;
foot1_mc.onReleaseOutside = noDrag;
```

That takes care of the feet. They both have the footSpring function as an onEnterFrame handler, so they will both spring to within 50 pixels of the body. And they are both draggable.

12. Now for the body. We want it to try to position itself so it is 50 pixels away from foot 0 and from foot 1. The key code for this is in these lines from the bodySpring function:

```
        var dx = this._x-foot_mc._x;
        var dy = this._y-foot_mc._y;
        var angle = Math.atan2(dy, dx);
        var targetX = foot_mc._x+Math.cos(angle)*radius;
        var targetY = foot_mc._y+Math.sin(angle)*radius;
        this.vx += (targetX-this._x)*k;
        this.vy += (targetY-this._y)*k;
```

All we need to do is run this code twice. Once with foot0_mc as the target and once with foot1_mc. We could just copy and paste this code again and change the names. But we might end up with even more feet. So rather than end up with a gargantuan function filled with almost duplicate code, we'll use a loop and dynamically generate the names.

```
for (i=0; i<2; i++) {
  var dx = this._x-_root["foot"+i+"_mc"]._x;
  var dy = this._y-_root["foot"+i+"_mc"]._y;
  var angle = Math.atan2(dy, dx);
  var targetX = _root["foot"+i+"_mc"]._x
                +Math.cos(angle)*radius;
  var targetY = _root["foot"+i+"_mc"]._y
                +Math.sin(angle)*radius;
  this.vx += (targetX-this._x)*k;
  this.vy += (targetY-this._y)*k;
}
```

We use 2 in the for loop, so it will loop once for each leg. The first time, *i* will be 0, the next time it will be 1. We then use that to dynamically construct the movie clip name: _root["foot"+i+"_mc"].

Note that there is no special formula for adding multiple forces to an overall velocity. You simply calculate the force, add it to the velocity, calculate the next force, add that to the velocity, and so on. I was pretty happy to discover that! After we finish the loop, we jump back out and apply the damping and add the final velocity figures to the coordinates.

13. Finally, we just alter our line drawing code to draw line to both legs.

```
_root.onEnterFrame = function() {
   _root.clear();
   _root.lineStyle(1, 0, 100);
   for (i=0; i<2; i++) {
         _root.moveTo(body_mc._x, body_mc._y);
         _root.lineTo(_root["foot"+i+"_mc"]._x,
                           _root["foot"+i+"_mc"]._y);
   }
};
```

I also highly suggest dropping down the damp value to at least 0.85, to prevent things from flying off the screen.

Now, to me, this is getting pretty damn cool.

Here’s the final code so far, in case you are getting confused:

```
radius = 50;
k = .2;
damp = .85;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
foot0_mc.onPress = doDrag;
foot0_mc.onRelease = noDrag;
foot0_mc.onReleaseOutside = noDrag;
foot1_mc.onPress = doDrag;
foot1_mc.onRelease = noDrag;
foot1_mc.onReleaseOutside = noDrag;
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
foot0_mc.onEnterFrame = footSpring;
foot1_mc.onEnterFrame = footSpring;
body_mc.onEnterFrame = bodySpring;
function footSpring() {
   if (!this.dragging) {
         var dx = this._x-body_mc._x;
         var dy = this._y-body_mc._y;
         var angle = Math.atan2(dy, dx);
         var targetX = body_mc._x+Math.cos(angle)*radius;
         var targetY = body_mc._y+Math.sin(angle)*radius;
         this.vx += (targetX-this._x)*k;
         this.vy += (targetY-this._y)*k;
```

```
          this.vx *= damp;
          this.vy *= damp;
          this._x += this.vx;
          this._y += this.vy;
     } else {
          this.vx = this._x-this.oldx;
          this.vy = this._y-this.oldy;
          this.oldx = this._x;
          this.oldy = this._y;
     }
}
function bodySpring() {
     if (!this.dragging) {
          for (i=0; i<2; i++) {
               var dx = this._x-_root["foot"+i+"_mc"]._x;
               var dy = this._y-_root["foot"+i+"_mc"]._y;
               var angle = Math.atan2(dy, dx);
               var targetX = _root["foot"+i+"_mc"]._x
                    +Math.cos(angle)*radius;
               var targetY = _root["foot"+i+"_mc"]._y
                    +Math.sin(angle)*radius;
               this.vx += (targetX-this._x)*k;
               this.vy += (targetY-this._y)*k;
          }
          this.vx *= damp;
          this.vy *= damp;
          this._x += this.vx;
          this._y += this.vy;
     } else {
          this.vx = this._x-this.oldx;
          this.vy = this._y-this.oldy;
          this.oldx = this._x;
          this.oldy = this._y;
     }
}
_root.onEnterFrame = function() {
     _root.clear();
     _root.lineStyle(1, 0, 100);
     for (i=0; i<2; i++) {
          _root.moveTo(body_mc._x, body_mc._y);
          _root.lineTo(_root["foot"+i+"_mc"]._x,
                    _root["foot"+i+"_mc"]._y);
     }
};
```

14. Now, you can add as many feet as you want. You just have to continue to name them sequentially, foot2_mc, foot3_mc, etc. and change the numbers in the for loops (one in bodySpring, one in the final line drawing code. You'd also need to add in the event handlers for them. We might as well throw those in a loop too! And rather than plugging hard coded numbers into all those loops, we'll set a variable, *numFeet* up at the top, and use that in the loops instead. Here's what it looks like:

```
numFeet = 5;
radius = 50;
k = .2;
damp = .85;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
for (i=0; i<numFeet; i++) {
   _root["foot"+i+"_mc"].onPress = doDrag;
   _root["foot"+i+"_mc"].onRelease = noDrag;
   _root["foot"+i+"_mc"].onReleaseOutside = noDrag;
   _root["foot"+i+"_mc"].onEnterFrame = footSpring;
}
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
body_mc.onEnterFrame = bodySpring;
function footSpring() {
   if (!this.dragging) {
        var dx = this._x-body_mc._x;
        var dy = this._y-body_mc._y;
        var angle = Math.atan2(dy, dx);
        var targetX = body_mc._x+Math.cos(angle)*radius;
        var targetY = body_mc._y+Math.sin(angle)*radius;
        this.vx += (targetX-this._x)*k;
        this.vy += (targetY-this._y)*k;
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
function bodySpring() {
   if (!this.dragging) {
        for (i=0; i<numFeet; i++) {
             var dx = this._x-_root["foot"+i+"_mc"]._x;
             var dy = this._y-_root["foot"+i+"_mc"]._y;
             var angle = Math.atan2(dy, dx);
             var targetX = _root["foot"+i+"_mc"]._x
                  +Math.cos(angle)*radius;
             var targetY = _root["foot"+i+"_mc"]._y
                  +Math.sin(angle)*radius;
             this.vx += (targetX-this._x)*k;
             this.vy += (targetY-this._y)*k;
        }
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
_root.onEnterFrame = function() {
   _root.clear();
   _root.lineStyle(1, 0, 100);
   for (i=0; i<numFeet; i++) {
        _root.moveTo(body_mc._x, body_mc._y);
        _root.lineTo(_root["foot"+i+"_mc"]._x,
                  _root["foot"+i+"_mc"]._y);
   }
};
```

15. Heck, since we are getting all automated here, why not automate our foot creation too, rather than manually placing them and naming them?

    First, delete all the feet from the stage. Leave the body there.

    Now, go into the library and right click (or whatever you Mac users do) on the foot symbol and choose “linkage”. Choose “Export for ActionScript” and give it the identifier “foot”. This will allow us to dynamically place as many feet as we choose on stage.

    Now, back in our script, go to the first for loop, where we set our foot event handlers, and change it to read as follows:

```
for (i=0; i<numFeet; i++) {
   attachMovie("foot", "foot"+i+"_mc", i);
   _root["foot"+i+"_mc"]._x = body_mc._x
                  +Math.random()*100-50;
   _root["foot"+i+"_mc"]._y = body_mc._y
                  +Math.random()*100-50;
   _root["foot"+i+"_mc"].onPress = doDrag;
   _root["foot"+i+"_mc"].onRelease = noDrag;
   _root["foot"+i+"_mc"].onReleaseOutside = noDrag;
   _root["foot"+i+"_mc"].onEnterFrame = footSpring;
}
```

    This just attaches an instance of foot, and places it somewhere nearby the body. Now we can instantly decide on 1 foot or 100 by changing one variable – numFeet! (I strongly recommend you don’t go much more than a dozen though. Each additional foot adds a whole bunch of calculations to the file, and generally makes things look messy. Around three to six look pretty good to me.)

16. There’s just a couple more things I want to do before I wrap up Part I of this tutorial. First, I want to add some gravity. But before I do that, I should make some barriers – minimally a floor, but I’ll throw in a couple of walls at no extra charge.

    Up top, we’ll create some values for these.

    ```
    right = 20;
    left = 520;
    bottom = 380;
    ```

    I always make my movies for BIT-101 as 540x400, with a 20 pixel frame, which is why I chose those figures. If you want, you can also draw in some lines on the stage –vertical lines at 20 y and 520 y, and a horizontal one at 380 x.

17. Now, we need to check if the body or any of the feet has gone past these barriers. As an example, we’d check if the object’s_x is greater than left. But we should also take into account the width of the object. So the if statement becomes:

    ```
    if(this._x>left-this._width/2)
    ```

    If it turns out that it did cross the line, we just move it back so it is resting right on the edge of the wall. Then we reverse its x velocity to simulate a bounce. Here’s how this looks all together:

    ```
    if (this._x>left-this._width/2) {
       this._x = left-this._width/2;
       this.vx *= -1;
    }
    ```

18. Then we do the same thing for right and bottom.

    ```
    if (this._x>left-this._width/2) {
       this._x = left-this._width/2;
       this.vx *= -1;
    }
    if (this._x<right+this._width/2) {
       this._x = right+this._width/2;
       this.vx *= -1;
    }
    if (this._y>bottom-this._height/2) {
       this._y = bottom-this._height/2;
       this.vy *= -1;
    }
    ```

    These 12 lines go in the bodySpring function and the footSpring function, right after the lines that add the velocity to the position. Here’s the code that shows how those functions look now:

    ```
    function footSpring() {
       if (!this.dragging) {
            var dx = this._x-body_mc._x;
            var dy = this._y-body_mc._y;
    ```

```
        var angle = Math.atan2(dy, dx);
        var targetX = body_mc._x+Math.cos(angle)*radius;
        var targetY = body_mc._y+Math.sin(angle)*radius;
        this.vx += (targetX-this._x)*k;
        this.vy += (targetY-this._y)*k;
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
        if (this._x>left-this._width/2) {
             this._x = left-this._width/2;
             this.vx *= -1;
        }
        if (this._x<right+this._width/2) {
             this._x = right+this._width/2;
             this.vx *= -1;
        }
        if (this._y>bottom-this._height/2) {
             this._y = bottom-this._height/2;
             this.vy *= -1;
        }
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
function bodySpring() {
   if (!this.dragging) {
        for (i=0; i<numFeet; i++) {
             var dx = this._x-_root["foot"+i+"_mc"]._x;
             var dy = this._y-_root["foot"+i+"_mc"]._y;
             var angle = Math.atan2(dy, dx);
             var targetX = _root["foot"+i+"_mc"]._x
                  +Math.cos(angle)*radius;
             var targetY = _root["foot"+i+"_mc"]._y
                  +Math.sin(angle)*radius;
             this.vx += (targetX-this._x)*k;
             this.vy += (targetY-this._y)*k;
        }
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
        if (this._x>left-this._width/2) {
             this._x = left-this._width/2;
             this.vx *= -1;
        }
        if (this._x<right+this._width/2) {
             this._x = right+this._width/2;
             this.vx *= -1;
        }
        if (this._y>bottom-this._height/2) {
             this._y = bottom-this._height/2;
             this.vy *= -1;
        }
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
_root.onEnterFrame = function() {
   _root.clear();
   _root.lineStyle(1, 0, 100);
   for (i=0; i<numFeet; i++) {
        _root.moveTo(body_mc._x, body_mc._y);
        _root.lineTo(_root["foot"+i+"_mc"]._x,
                  _root["foot"+i+"_mc"]._y);
   }
};
```

Now at least we have some barriers in place. Try bouncing things around off the walls and floor.

19. Last, but not least, we add in some gravity. Gravity is simply another force that will pull the objects downward. In other words, will add to their y velocity.

    First, define gravity up top:

    ```
    grav = .5;
    ```

    Then we simply go into bodySpring and footSpring and add grav to the current y velocity, just before we apply damping. Here is the final, final, final code for this tutorial.

```
grav = .5;
right = 20;
left = 520;
bottom = 380;
numFeet = 4;
radius = 50;
k = .2;
damp = .85;
body_mc.onPress = doDrag;
body_mc.onRelease = noDrag;
body_mc.onReleaseOutside = noDrag;
for (i=0; i<numFeet; i++) {
   attachMovie("foot", "foot"+i+"_mc", i);
   _root["foot"+i+"_mc"]._x = body_mc._x
                  +Math.random()*100-50;
   _root["foot"+i+"_mc"]._y = body_mc._y
                  +Math.random()*100-50;
   _root["foot"+i+"_mc"].onPress = doDrag;
```

```
   _root["foot"+i+"_mc"].onRelease = noDrag;
   _root["foot"+i+"_mc"].onReleaseOutside = noDrag;
   _root["foot"+i+"_mc"].onEnterFrame = footSpring;
}
function doDrag() {
   this.startDrag();
   this.dragging = true;
}
function noDrag() {
   stopDrag();
   this.dragging = false;
}
body_mc.onEnterFrame = bodySpring;
function footSpring() {
   if (!this.dragging) {
        var dx = this._x-body_mc._x;
        var dy = this._y-body_mc._y;
        var angle = Math.atan2(dy, dx);
        var targetX = body_mc._x+Math.cos(angle)*radius;
        var targetY = body_mc._y+Math.sin(angle)*radius;
        this.vx += (targetX-this._x)*k;
        this.vy += (targetY-this._y)*k;
        this.vy += grav;
        this.vx *= damp;
        this.vy *= damp;
        this._x += this.vx;
        this._y += this.vy;
        if (this._x>left-this._width/2) {
             this._x = left-this._width/2;
             this.vx *= -1;
        }
        if (this._x<right+this._width/2) {
             this._x = right+this._width/2;
             this.vx *= -1;
        }
        if (this._y>bottom-this._height/2) {
             this._y = bottom-this._height/2;
             this.vy *= -1;
        }
   } else {
        this.vx = this._x-this.oldx;
        this.vy = this._y-this.oldy;
        this.oldx = this._x;
        this.oldy = this._y;
   }
}
function bodySpring() {
   if (!this.dragging) {
        for (i=0; i<numFeet; i++) {
             var dx = this._x-_root["foot"+i+"_mc"]._x;
             var dy = this._y-_root["foot"+i+"_mc"]._y;
             var angle = Math.atan2(dy, dx);
```

```
            var targetX = _root["foot"+i+"_mc"]._x
                 +Math.cos(angle)*radius;
            var targetY = _root["foot"+i+"_mc"]._y
                 +Math.sin(angle)*radius;
            this.vx += (targetX-this._x)*k;
            this.vy += (targetY-this._y)*k;
       }
       this.vy += grav;
       this.vx *= damp;
       this.vy *= damp;
       this._x += this.vx;
       this._y += this.vy;
       if (this._x>left-this._width/2) {
            this._x = left-this._width/2;
            this.vx *= -1;
       }
       if (this._x<right+this._width/2) {
            this._x = right+this._width/2;
            this.vx *= -1;
       }
       if (this._y>bottom-this._height/2) {
            this._y = bottom-this._height/2;
            this.vy *= -1;
       }
  } else {
       this.vx = this._x-this.oldx;
       this.vy = this._y-this.oldy;
       this.oldx = this._x;
       this.oldy = this._y;
  }
}
_root.onEnterFrame = function() {
  _root.clear();
  _root.lineStyle(1, 0, 100);
  for (i=0; i<numFeet; i++) {
       _root.moveTo(body_mc._x, body_mc._y);
       _root.lineTo(_root["foot"+i+"_mc"]._x,
                 _root["foot"+i+"_mc"]._y);
  }
};
```

There you have it, a rudimentary BIT-101 creature. Yes, he's pretty spineless right now, and other than manually throwing him around, pretty lifeless too. But we'll handle that in Part II and maybe Part III if necessary. Anyway, there's enough material there to keep you busy playing for a while.

By the way, the file is available for download (just to prove it works!) at:

http://www.bit-101.com/flafiles/020628.fla
http://www.bit-101.com/content/020628.swf

My last two words of advice:

1. Don’t simply copy and paste this code. Try to fully understand what is going on, and why different things are there. If you have any specific questions about any parts of the code or explanations, feel free to email me directly. If something isn’t written clearly, or even incorrectly, I’m very interested in fixing it up.

2. Experiment with it! This code isn’t sacred. It works, but mess with it, break it, push it and mold it into something completely unheard of. And when you get something interesting, send it my way!

Keith Peters
BIT-101
kp@bit-101.com