This is a tutorial on how to simulate various gravity forces on an object in a Flash movie. I am releasing this tutorial freely, for the benefit of the Flash Programming community. Feel free to pass it on to anyone. All I ask is that you keep this document in its original form. If you want to add to it, place your comments or additions at the end of the document, with your name, e-mail address and date and send me a copy.

I am writing this tutorial in stages, each stage being a complete working movie. Each next stage will add a few more features to the previous one. This is the way I built the file in the first place, getting one thing to work, then adding a little more…

The general outline will be:

A. Create an object and get it to move in one direction (i.e. horizontally).

B. Get it to bounce off one side of the screen, then two sides.

C. Then get it moving vertically as well as horizontally, and bouncing off all sides of the screen.

D. Adding gravity.

E. Adding friction.

F. Adding a variable level of "bounce".

G. Letting you pick up the ball and throw it.

OK, let's start.

A. CREATING THE OBJECT AND MOVING IT

Here we are just going to create the object and have it move at a steady rate using Action Script alone.

1. Create a new movie. Modify movie to change the background to black and the frame rate to 20. This will make things look pretty smooth. Keep the default size settings of 550x400.
2. Draw a circle, select it, turn it into a movie clip object.
3. Double click the movie clip to edit it.
4. Change the name of the first layer to "ball".
5. Create another layer, labeled "actions".
6. Make 3 key frames in the action layer, and make sure the ball level extends out 3 frames as well.
7. The first key frame will initialize the object. Select the frame, go to the action panel and type in the following (I recommend using Expert Mode):

```
_x=10;
_y=200;
xspeed=5;
```

   It's easy to see that this positions the object at the coordinates 10, 200 and assigns a speed of "5". This step could also be done with a "onClipEvent(load)"action applied to the object itself, but for the sake of clarity, I kept all the code in this timeline.

8. Now go to the second frame in actions, and type in the following:

```
_x=_x+xspeed;
```

   Again, pretty simple. This merely increases the objects x coordinate by 5 (the value of "xspeed").

9. Now, go to frame 3 in actions and type in:

```
gotoAndPlay(2);
```

This will cause the clip to loop continuously between frames 2 and 3, constantly changing the location of the object based on the speed variable.

10. Test the movie. The ball should move from the left to right and go off the screen into oblivion. Try changing the speed variable and see that the ball goes faster or slower. Nothing too exciting yet, but this is the framework for the rest of the movie. It works like we expected, so we can build another piece on top of this safely. That will be adding a "bounce" effect.

B. MAKING THE BALL BOUNCE

Having our ball go off into oblivion wasn't very exciting. So let's have it bounce back when it reaches the edge.

1. First we need to know where the edge is and we need to know when we hit it. So, go into your movie clip, action frame 1, where we initialize everything and add this line:

```
rightedge=550;
```

I got this from the default size of a Flash movie which is 550. So we want to bounce when we hit that.

2. OK, now we know where the edge is, we need to know when we hit it. Go to frame 2, which is where we move the ball. After we add the speed to the position, we need to check if we have gone past the right edge. If we have, then we are going to place the ball ON the right edge instead. Then we want to change the direction it is heading. Here:

```
if(_x+_width/2>rightedge) {
  _x=rightedge-_width/2;
  xspeed=-xspeed;
}
```

What is this "_width/2"? Remember that your object's _x position is the *center* of the object. If you put that at the edge of the screen, the ball will be half on and half off the screen. So, you take half the width (the _width property) and add or subtract that from your position.

In the first line, you are checking if the right edge of your object (_x+_width/2) is more than the right edge of your screen. If so, you are putting the right edge of your object ON the right edge of the screen. That way it looks like it is hitting the edge. This may not be the most precise way of positioning the object at the point of the bounce, particularly when you get into two dimensions, but it is simple and it ended up looking pretty damn good in the final product.

The last line changes the speed from a positive amount to a negative amount, thus reversing the direction of the ball. (As we will soon see, the same line will also change the speed from negative to positive when it hits the opposite wall.)

3. Test the movie now. The ball should hit the right side of the screen and bounce off it, going the same speed in the other direction. Still not too exciting, but we are getting there. Now let's bounce it off the left wall.

4. Add the following to your initialization frame, frame 1 in actions:

```
leftedge=0;
```

I shouldn't have to explain that one.

5. Now in frame 2 add:

```
if(_x-_width/2<leftedge) {
  _x=leftedge+_width/2;
  xspeed=-xspeed;
}
```

This is essentially the same code as in number 2 above, but you are checking if the ball has traveled past the right edge, and if so, putting it ON the right edge and again changing the direction.

6. Test it again. It should bounce off both sides now, back and forth. Still not super realistic, but it is getting better. And we have a solid foundation to add to.

C. VERTICAL MOTION

Now we need to make the ball move in two dimensions. This is going to be easier than you might think. Basically, we just need to duplicate everything we did so far and apply it to the "y" dimension.

1. First we need some vertical speed. Go to the initialization frame (1) and enter:

```
yspeed=20;
```

I also changed xspeed to 20 just to speed things up a bit.

2. Now we need to know where the top and the bottom of the screen are, instead of the left and right. So add:

```
topedge=0;
bottomedge=400;
```

3. Now go to frame 2. Copy everything there, and paste another copy just below the existing text. Change the following:
Change _x to _y
Change xspeed to yspeed
Change leftedge to topedge
Change rightedge to bottomedge
Change _width to _height

I almost hate to say it, but that's all for this step! Test it. It should bounce around the screen, rebounding off of each edge. Although it is about as realistic as a game of "Pong", we will soon fix that with the next few steps.

D. ADDING GRAVITY

This step will go a long way to make the ball look more realistic. To figure this one out, we have to look at what gravity is or does. In practical terms, it is a force that is always pulling something down. So in our flash file, that means a piece of code that constantly adds a positive "y" value. (Remember that the y value increases as you go down toward the bottom of the screen.) If you threw an object down from a tall building, it would start out with a certain speed (a positive y value), but as gravity acted on it, it would go faster and faster until it landed. In other words, its yspeed would increase continuously. If you throw a ball up in the air, you are giving it a negative yspeed. Gravity will pull it in a positive y direction – adds a positive y value to it – until its yspeed reaches 0, then it starts falling down again, its yspeed becoming positive. So in terms of our code, gravity is just a number that we add to the yspeed on each loop. Again, this is going to seem too simple.

1. Go to frame 1 and give gravity a value of 2:

```
gravity=2;
```

2. Then go to frame 2 and put in the following, right at the bottom:

```
yspeed=yspeed+gravity;
```

3. Again, I hate to say it, but that's all for this step! Test the movie. The ball should bounce around looking much more realistic. Play around with the gravity and speed settings to see what happens. Now at this point, it doesn't look so much like "Pong" but it looks like a tennis match with invisible opponents. That's because the ball is never losing any momentum. It might eventually be dragged down by gravity, but it is bouncing around damn near perpetually. It's like it is constantly being hit so as to maintain its speed. In the real world, two things would slow it down. One would be friction – the air molecules it is passing through would put a small amount of drag on it. The other is the fact that an object that bounces loses some force when it hits the ground. It rises up with a bit less speed than it had on the way down. We will address those two factors next.

E. FRICTION

If you shot a bullet in outer space, it would theoretically go on at the same speed forever. Nothing would slow it down. Here on earth, though, any moving object experiences friction from the air, water, or other medium it is traveling in. If you loosely crumple up a piece of paper and try to throw it, you will notice that it does not go very far. The air stops it. We are now going to model that force in our Flash universe. We are going to create some "drag". Drag is a factor that reduces any speed by a small percent – xspeed and yspeed in any direction. So all we have to do is multiply our speed variables by, let's say 0.98 on each loop.

1. Back in frame 1, create a drag variable:

```
drag=.98;
```

Also, for the sake of keeping things interesting, change the xspeed and yspeed lines to read:

```
xspeed=Math.random()*30;
yspeed=Math.random()*30;
```

Math.random() creates a random number between 0 and 1. Multiplying that times 30 gives us an xspeed and yspeed between 0 and 30. It will be different each time we run the program.

2. In frame 2 change the line that reads:

```
yspeed=yspeed+gravity;
```

to:

```
yspeed=yspeed*drag+gravity;
```

and add this line:

```
xspeed=xspeed*drag;
```

3. Test the movie. Pretty cool, huh? Now we are looking very realistic. Amazing what a couple of lines of code can do. Again, play with the values and see what happens when you make more or less friction, gravity, etc.

F. ADDING A LEVEL OF BOUNCE

So our ball is looking pretty realistic right now. It is a nice, bouncy rubber ball. But say you wanted a wooden ball, or a steel ball, etc. These would not be quite so bouncy. They would hit the ground (or wall, or ceiling) with a certain speed and rebound with a much lower speed, if any at all. So we are again talking about a factor that will reduce the speed by a certain percentage, very similar to drag. But this time it will only reduce the speed when it hits a surface. We will call this factor "bounce". A theoretically perfect "superball" would have a bounce factor of "1". It would bounce up with the exact same speed it landed with (although our "drag" would reduce its speed). A lead ball would have a bounce factor pretty close to zero. You would expect it to hit the ground and pretty much stay there. Most other substances are somewhere in between.

1. You can probably guess that we are going to go into frame one and add a "bounce" variable.

```
bounce=.9;
```

2. Now we need to go into frame 2 and add this factor in the same way we did with "drag", but only when the ball bounces. We have four "if" statements there. Each one has a statement of

```
xspeed=-xspeed;
```

or

```
yspeed=-yspeed;
```

That is the point of the bounce. We just need to add our "bounce" factor here. Change these four statements to read:

```
xspeed=-xspeed*bounce;
```

and

```
yspeed=-yspeed*bounce;
```

3. Now test your file. Not much difference, right? That's because we have a high bounce factor, .9. Change that to something smaller, like .3 and see the difference. This allows you to change the material you are working with. That pretty much wraps up the modeling of the world and its forces. In the next section we will learn how to interactively drag and throw the ball, which is more complicated than the last few steps, but I was pretty surprised how easy it actually was when I sat down to figure out how to do it. Before we go on to that, here is the final code we have for our ball so far:

In frame 1:

```
_x=10;
_y=200;
xspeed=Math.random()*60-30;
yspeed=Math.random()*60-30;
rightedge=550;
leftedge=0;
topedge=0;
bottomedge=400;
gravity=2;
drag=.98;
bounce=.9;
```

In frame 2:

```
_x=_x+xspeed;
if(_x+_width/2>rightedge) {
  _x=rightedge-_width/2;
  xspeed=-xspeed*bounce;
}
if(_x-_width/2<leftedge) {
  _x=leftedge+_width/2;
  xspeed=-xspeed*bounce;
}
_y=_y+yspeed;
if(_y+_height/2>bottomedge) {
  _y=bottomedge-_height/2;
  yspeed=-yspeed*bounce;
}
if(_y-_height/2<topedge) {
  _y=topedge+_height/2;
  yspeed=-yspeed*bounce;
}
yspeed=yspeed*drag+gravity;
xspeed=xspeed*drag;
```

G. DRAGGING AND THROWING THE BALL

So far, the ball appears, moves in a random direction, bounces to a stop and that's the end of the show. Now we will get interactive and learn how to pick up the ball and throw it. We want to be able to throw the ball in any direction, and have the speed of our throw determine the speed of the ball. No problem. To do this, we need to make the ball movie clip "draggable". I'm hoping you know a little bit about dragging movie clips, but if not, you will learn a little here.

1. First we need an invisible button. Create a new symbol, name it "invisibleButton" and make sure it has button behavior checked.
2. An invisible button is one that only has a "hit" state. You can't see it, but you can react with it. Create a key frame in the hit state, leave the up, over and down states blank, and draw a circle in the hit state. The circle just needs to be a fill, no outline, and it should be perfectly centered on the crosshair in the middle of the stage. This is done most easily with the grid and snap to grid.
3. Go back you your ball movie clip and create a new layer. Name it "button".
4. Open your library and drag a copy of "invisibleButton" onto the stage. Make sure it is in the "button" layer. You will see a light blue outline of the button. This is just to let you know it is there and where the hit state is. This will not show up when you play the movie.
5. Position it so it is directly over the ball, and scale it if need be to cover the ball exactly.
6. Making sure the button is selected, open the Actions panel. Type in the following:

```
on(press){
      startDrag(this);
      dragging=true;
}
on(release, releaseOutside){
      stopDrag();
      dragging=false;
}
```

In short, what this does is, when you press the button, it directs the movie clip to start following the mouse around. It also sets a variable "dragging" to be true. When you release the mouse, it stops dragging the clip and sets "dragging" to false. You will see how we use this variable next.

7. If you test the movie now, you'll see that you can pick up the ball and drag it. But, all of our other forces are still active on it, so it keeps slipping from your hand. We need to turn off our gravity, friction, bounce and speed while we are dragging. This is what the "dragging" variable is for. It lets us know when we are dragging and when we stop dragging. Go into frame 2, and add the following line to the very top of the code there:

   ```
   if(!dragging){
   ```

   and, on the very bottom line, close the bracket:

   ```
   }
   ```

   The "!" means "not". So we are saying, if "dragging" is not true (false) then we will apply gravity, etc. But if we are dragging the clip, then "dragging" will be true, and this whole block of code will be ignored. Test the movie. You will now see that you can pick the ball up and drop it. We are getting there…

8. Now we want to throw the ball, and determine its speed and direction. As I said before, I was surprised at how easy this turned out to be. All we need to do is keep track of the clip's position while we are dragging it. We will store this in variables called "oldx" and "oldy". On the next loop, we subtract "oldx" from the current value of _x and we will have the xspeed at which you are dragging. I'll give you an example:

   When you start dragging, _x is 100. We store that in "oldx". The next time the frame loops around, you have moved the ball to the right, so that _x now equals 110.
   110 – 100 = 10 or _x – oldx = xspeed.
   Say you moved it faster, so the ball was at 130 on the second loop.
   130 - 100 = 30, a faster xspeed.
   Or, say you moved it to the right instead. _x now equals 90. 90 – 100 = -10. That's a negative xspeed, or a motion to the left.
   We do the same thing with the _y values, "oldy" and yspeed.

   We only want to check this when we are dragging. We already have an "if" statement checking on this, and code to run if we are *not* dragging. By adding on an "else" statement, we can tell it what to do if we *are* dragging. Add the "else" right at the end of the existing closing bracket:

   ```
   } else {
   xspeed=_x-oldx;
   yspeed=_y-oldy;
   oldx=_x;
   oldy=_y;
   }
   ```

   And there we have it! I told you it was surprisingly simple.

In my final movie, I set up sliders to interactively change the gravity, friction and bounce. I'll leave that up to you. I also moved some of the properties out of the movie clip, and into "_root". This was a move toward more pure object oriented coding. Things like position, speed and bounce are properties of the ball, and belong with the ball. Things like friction, gravity and position of the walls, floor and ceiling are not really properties of the ball, but of the surrounding world or environment. So they belong outside of the ball. So you could have several different balls bouncing around. Each one would have it's own bounce factor (material), position and speed, but they should all share common gravity, friction and barriers.

Once again, here is the final code for each section:

Frame 1:

```
_x=10;
_y=200;
xspeed=Math.random()*60-30;
yspeed=Math.random()*60-30;
rightedge=550;
leftedge=0;
topedge=0;
bottomedge=400;
gravity=2;
drag=.98;
bounce=.9;
```

Frame 2:

```
if (!dragging) {
      _x = _x+xspeed;
      if (_x+_width/2>rightedge) {
            _x = rightedge-_width/2;
            xspeed = -xspeed*bounce;
      }
      if (_x-_width/2<leftedge) {
            _x = leftedge+_width/2;
            xspeed = -xspeed*bounce;
      }
      _y = _y+yspeed;
      if (_y+_height/2>bottomedge) {
            _y = bottomedge-_height/2;
            yspeed = -yspeed*bounce;
      }
      if (_y-_height/2<topedge) {
            _y = topedge+_height/2;
            yspeed = -yspeed*bounce;
      }
      yspeed = yspeed*drag+gravity;
      xspeed = xspeed*drag;
} else {
      xspeed=_x-oldx;
      yspeed=_y-oldy;
      oldx=_x;
      oldy=_y;
}
```

Frame 3:

```
gotoAndPlay(2);
```

and the actions on the invisible button:

```
on(press){
      startDrag(this);
      dragging=true;
}
on(release, releaseOutside){
      stopDrag();
      dragging=false;
}
```

I hope you can take what I did here and evolve it into something even more interesting. If you do, please show me. Good luck. If you add any comments or additions to the end of this document, please send me a copy too. Thanks.

Keith Peters

kp@bit-101.com

Comments or additions by others:

(please include name, e-mail and date)
