

Art 486: Advanced Interactive Media

mcdo@umbc.edu

Lecture 6, Sept 22

Schedule

- Take Roll
- make the particle system again
- add in mouse input
- add in keyboard input
- add in sound

let's make a particle system

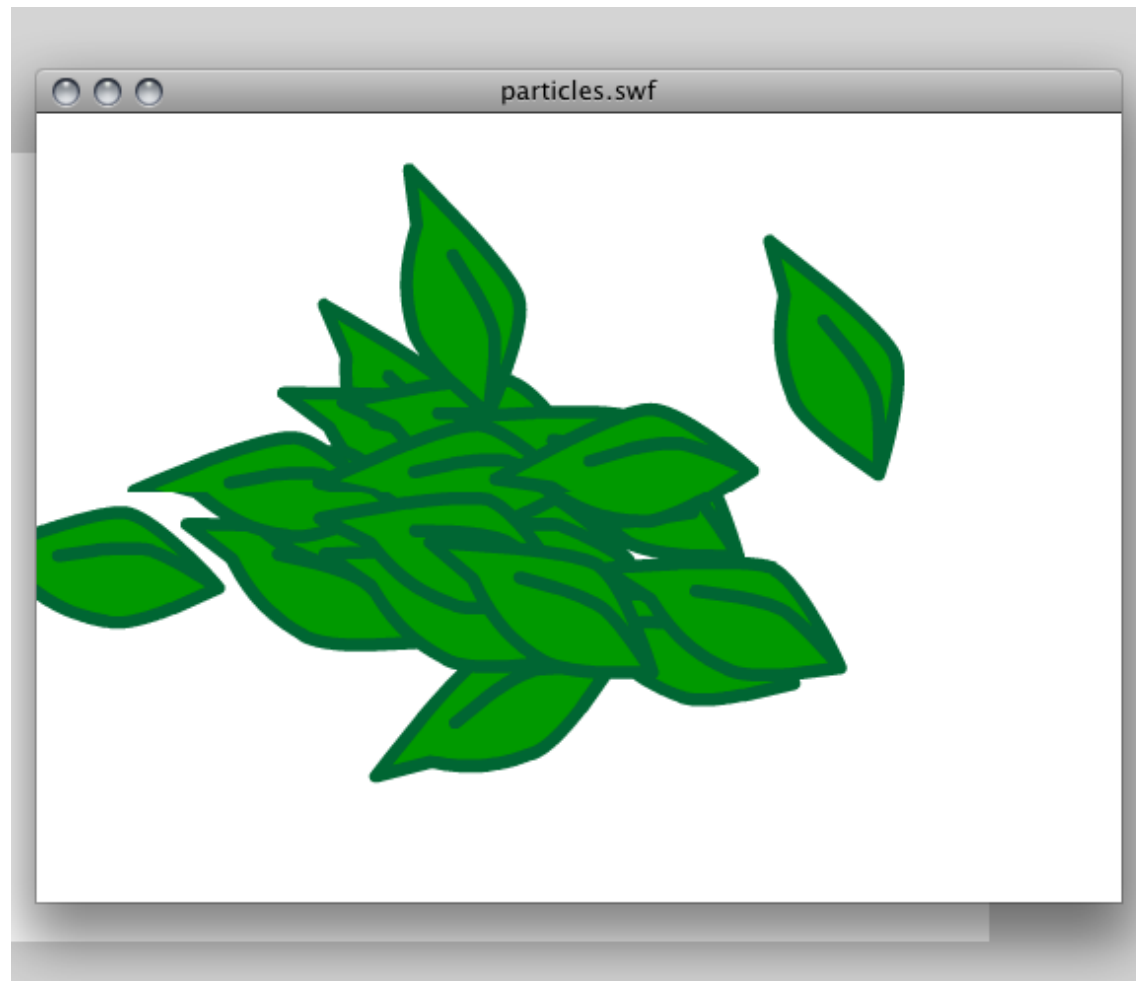
- on frame 1

on frame 3

```
1  
2 var syms: Array;  
3 var i: int;  
4  
5 syms = new Array(20);  
6  
7 for (i=0; i<20; ++i) {  
8     syms[i] = new mySymbol();  
9     syms[i].x = i*10 + 200;  
10    syms[i].y = i*5 +200;  
11    addChild(syms[i]);  
12 }  
13
```

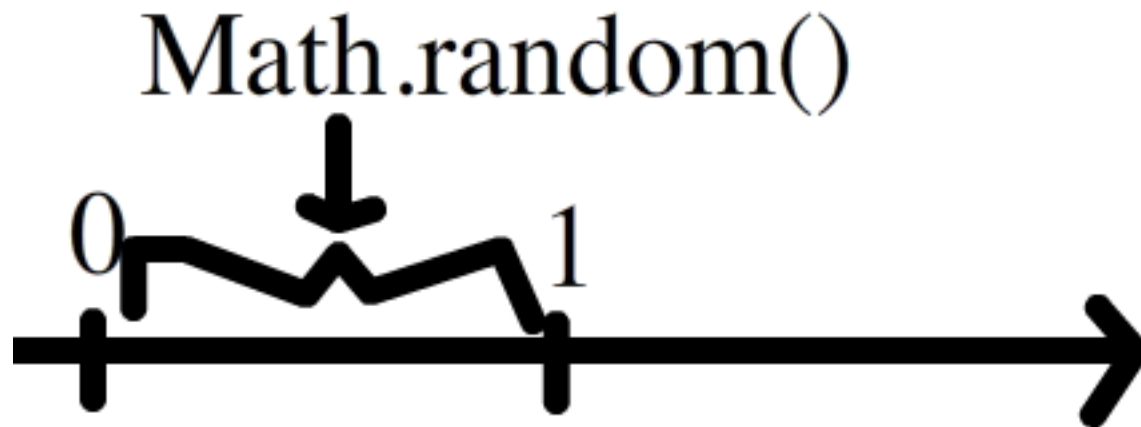
```
2  
3 for (i=0; i<20; i=i+1) {  
4     syms[i].x += Math.random()*10.0-5;  
5     syms[i].y += Math.random()*10.0-5;  
6     syms[i].rotation +=Math.random()*5.0-2.5;  
7 }  
8 gotoAndPlay(2);  
9
```

result: jittery things



looking at randomness

- `Math.random()` returns one number between 0 and 1
- different one every time you call it
 - called 60 times every time we get to frame 3
- equally likely to be anywhere in the range

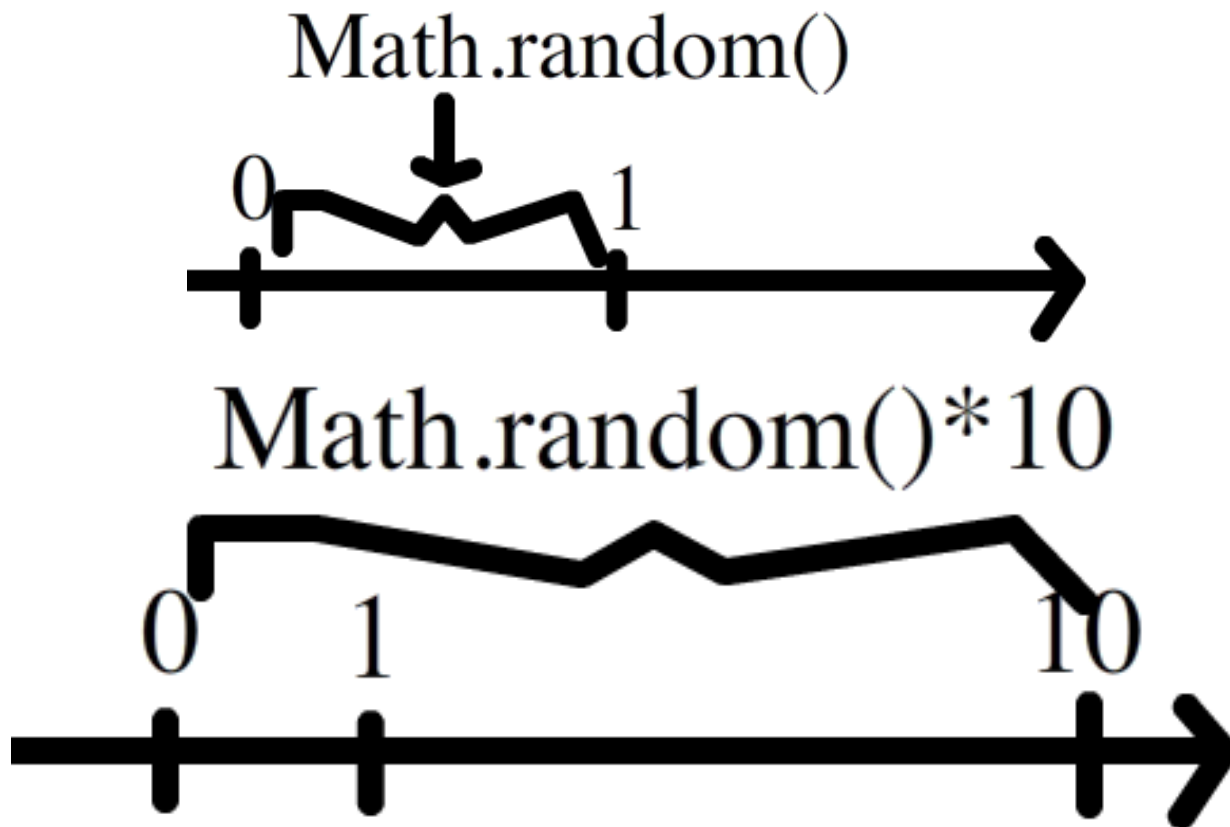


changing the range

- $\text{Math.random()} * 10$
- still random; the behavior of Math.random() doesn't change
- if Math.random() returns .01,
 $\text{Math.random()} * 10 = .1$
- if Math.random() returns .9,
 $\text{Math.random()} * 10 = 9$
- $\text{Math.random()} * 10$ returns a number between 0 and 10

thinking visually

- multiplication stretches the brackets



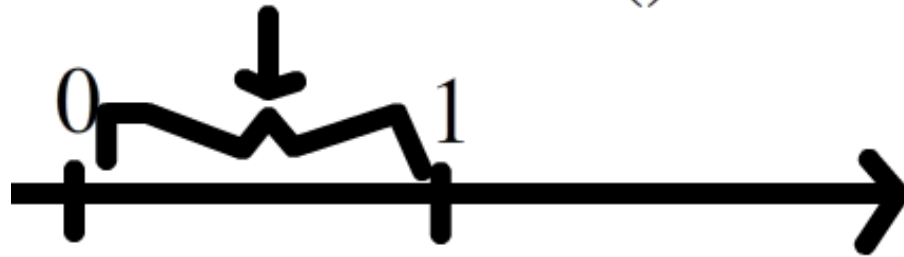
changing the range's position

- `Math.random()-.5`
 - does not change the width of the brackets
 - changes where they are on the number line
 - if `Math.random()=.1`, `Math.random()-.5=-.4`
 - if `Math.random()=.9`, `Math.random()-.5=.4`
 - if `Math.random()=.5`, `Math.random()-.5=0`

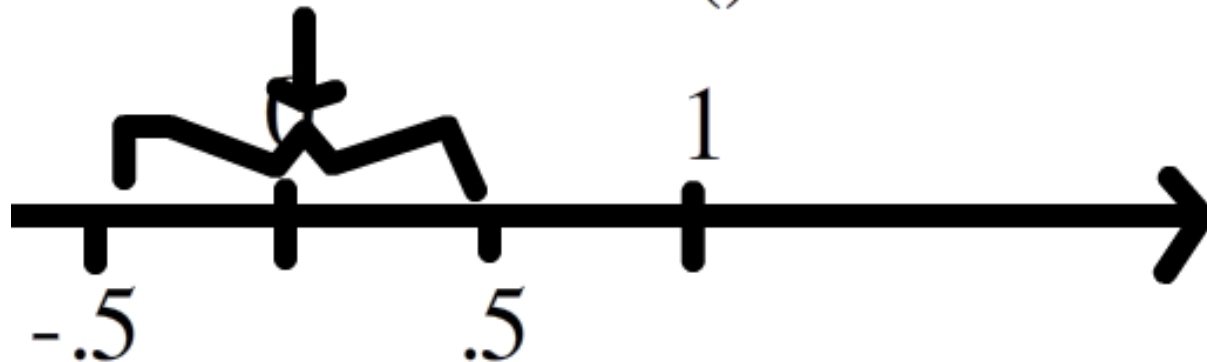
thinking visually

- addition slides the brackets around

`Math.random()`

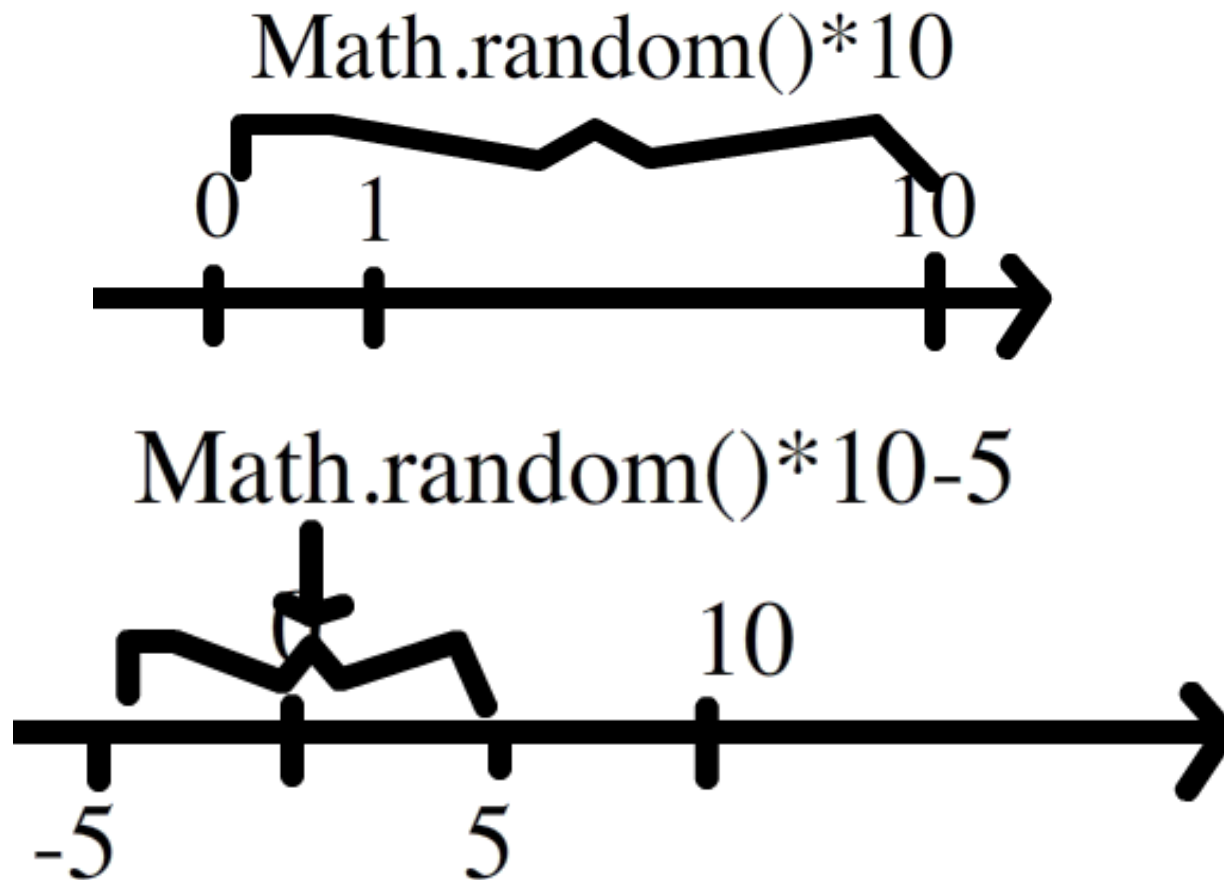


`Math.random()-.5`



combine the two!

- $\text{Math.random()} * 10 - 5$
- first stretch, then slide!



an aside: Jared Tarbell

- many, many flavors of random numbers
- <http://www.levitated.net/daily/levPolychrome.html>
- <http://www.levitated.net/bones/blueBoxes/index.html>
- lots of mouse input, too. hmm!

mouse input

- to get input, you must “process events” or “handle events”
- most of the time, you are not giving input
 - you can click maybe twice a second
 - flash redraws 30 times a second
 - most frames, you’re not giving input
 - when you click, it’s special! it’s an “event”

event types

- input is an event
- you can click the mouse, or push a key
- `MouseEvent.CLICK` – a mouse click
 - press and release, within a short time (<.3sec)
- Generated by Flash whenever you click the mouse
 - events are messages
 - how do you receive them?

```
stage.addEventListener(event, handler);
```

- a function
- call it once
- give it the type of the event you want to listen for
- give it the name of a function
 - a special kind of function
 - an “event handler”
 - it will be called by Flash when the event happens

stage.XXX

- this function is part of whatever class “stage” belongs to
- “stage” is the background of the application
- one stage per Flash application
- sets frame rate, resolution, background color
- when you click on nothing, you click on the background– the stage

hidden complexity!

- how does Flash know how the mouse works?
 - don't care; it does— lots of work for Adobe
- where is the code that calls our function?
 - don't care; it just does
 - for the example of the last slide, somewhere in “stage”'s class
- Flash and ActionScript work together to let us handle events

event listeners

- Flash maintains a list of functions to call and events that cause them to be called
- `addEventListener(event, fn)`
 - adds an item to that list
 - (the event to watch for, the function to call)
- every frame, flash looks for events.
 - if there are any, it looks at the list
 - if it finds a function for that event, it calls the fn
 - if it doesn't, the event is ignored

declaring an event handler

```
// for a mouse event
```

```
addEventListener(MouseEvent.CLICK,  
    clicker);
```

```
// this function definition will work
```

```
function clicker(e:MouseEvent):void {  
    // ... do stuff  
}
```

inside MouseEvent

- the only input to a mouse event handler (the function that gets called) is the mouse event
- why?
 - we don't get to see the code where flash calls our mouse event handler
 - we can't modify that code
 - we can't control what input our function gets
 - fortunately, we get lots of input

an example mouse event handler

```
1  
2 stage.addEventListener(MouseEvent.CLICK,  
3     drawSpot);  
4 function drawSpot(e:MouseEvent) {  
5     graphics.lineStyle(2, 0xff0000, 1.0);  
6     graphics.drawCircle(e.stageX, e.stageY, 50);  
7 }  
8
```

MouseEvent.MOUSE_MOVE

- Flash makes one of these every time the mouse moves
- this event listener will call its event handler every time you move the mouse

```
stage.addEventListener(MouseEvent.MOUSE_MOVE,  
                                         drawSpot);
```

- an event listener is attached to the stage
- it's called every time the mouse is moved
- it calls the function named “drawSpot”

```
function drawSpot(e:MouseEvent) {
```

- here's the definition of the event handler function
- only one input allowed, e, which is a MouseEvent

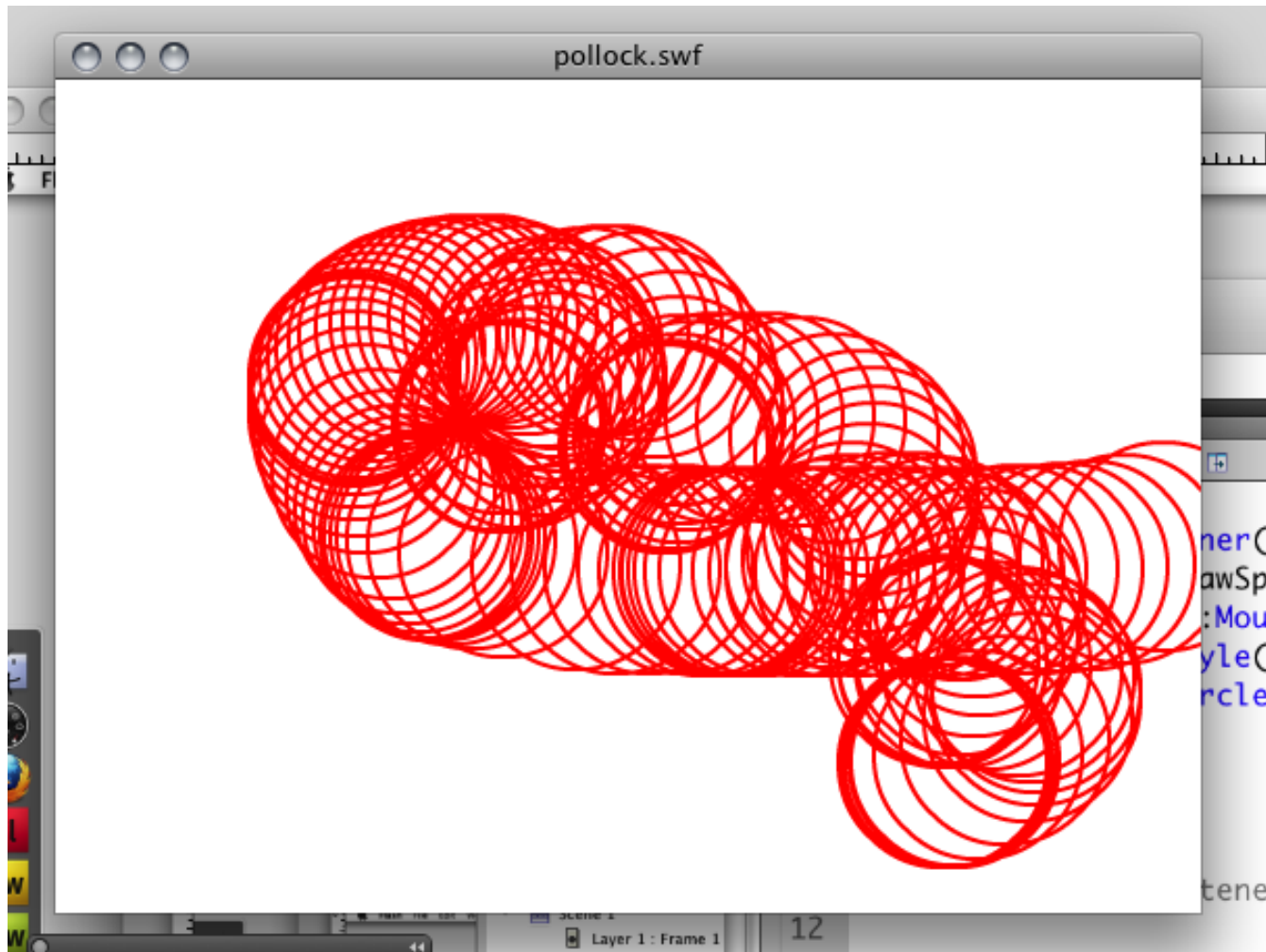
```
graphics.lineStyle(2, 0xff0000, 1.0);
```

- set the drawing style to be
 - line width 2
 - color red
 - alpha = 1.0 => opaque line

```
graphics.drawCircle(e.stageX, e.stageY, 50);
```

- draw a circle with the current line style
- “r” sets the radius of the circle
 - the radius is between 0 and 50 pixels
- e is the event– the mouse event given to the function
 - e.stageX is the x-coordinate of the mouse on the stage
 - e.stageY is the y-coord

when the mouse moves,
draw a red circle around it



types of mouse events

- CLICK
- DOUBLE_CLICK
- MOUSE_DOWN
- MOUSE_MOVE
- MOUSE_OUT
- MOUSE_OVER
- MOUSE_WHEEL

keyboard events are similar

```
addEventListener(KeyEvent.KEY_UP, keyUpper);
```

```
function keyUpper (event:KeyEvent):void { ... }
```

the types of keyboard events are only KEY_UP and
KEY_DOWN

saving the mouse position

- this program only draws when you move the mouse

```
2 var msx, msy, dx, dy: int;
3 graphics.lineStyle(2, 0xff0000, 1.0);
4
5 stage.addEventListener(MouseEvent.CLICK,
6     saver);
7
8 function saver(e:MouseEvent) {
9     msx = e.stageX;
10    msy = e.stageY;
11 }
12
13 dx = Math.random()*10-5;
14 dy = Math.random()*10-5;
15
16 graphics.drawCircle(msx+dx, msy+dy, 10);
17 gotoAndPlay(2);|
```

change to click handler

- instead of drawing, it sets “msx” and “msy”
 - to be the mouse’s position
 - this is the only code that sets msx and msy
 - msx is always the mouse’s x position

frame 3 code

- make dx and dy some random small distance
- draw a circle near the mouse

```
3 dx = Math.random()*10-5;  
4 dy = Math.random()*10-5;  
5  
6 graphics.drawCircle(msx+dx, msy+dy, 10);  
7 gotoAndPlay(2);|
```