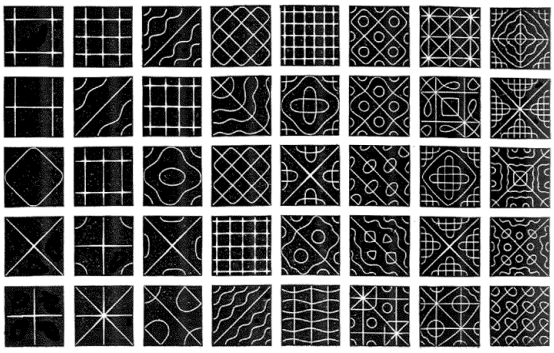
1 Intertactive Visual Pattern

Quanto più a fondo esaminiamo la struttura della Natura, tanti più frattali scopriamo. In effetti la loro onnipresenza nel mondo naturale di cui facciamo farte è una delle ragioni per le quali ci è così facile trovarli attraenti. Essi rappresentano la forma di computer art che ha colto il programma fondamentale di cui si sono servite le entità viventi per stabilire le proprie nicchie distinte nell'intero corso della storia evolutiva: la riproduzione autosimile di uno stesso motivo in differenti dimensioni.
-- John D. Barrow, L'Universo come opera d'arte.

Visual Pattern

"*Ogni viaggio inizia con un primo passo*" diceva Confucio, quindi è arrivato il momento di aprire Flash e intraprendere la scoperta di uno dei linguaggi che stanno ridefinendo i limiti della progettazione di ambienti sensibili. Saliamo su un tappeto volante e dirigiamoci verso la prima meta. Una meta molto vicina a noi. Basta voltare lo sguardo ed osservare il tappeto su cui siamo seduti. A cosa somiglia? Può sembrare un giardino zen, con le linee che si inseguono e si ripetono, oppure un mandala. Riconosciamo subito che ci sono dei muduli (texture, elementi di base) che si ripetono con un certo ritmo. Ho avuto la fortuna di assistere alla tessitura con il telaio a mano, un oggetto intorno alla cui complessità si sono generate leggende e dicerie che lasciano credere che infilare centinaia di fili sia un lavoro che solo in pochi sanno fare e chi lo fa è sempre circondato da un'alone di meraviglia e rispetto.

"*Uno scienziato del del XIX secolo, Ernst Chladni, escogitò un esperimento che era solito eseguire davanti alle corti d'Europa*"[4], l'esperimento consisteva nell'utilizzare una lastra di metallo, ricoperta di sabbia, che faceva vibrare con l'archetto di un violino. Magicamente la sabbia si disponeva in strane forme secondo le vibrazioni che produceva la lastra.



4 Du Sautoy Marcus, L' enigma dei numeri primi

City Scape
Visualizing urban noise

Il primo progetto è scrivere il codice ActionScript per un'installazione temporanea che indaga il rapporto tra il rumore dell'ambiente circostante e la vibrazione di visual pattern proiettati su un'architettuta urbana.

City Scape

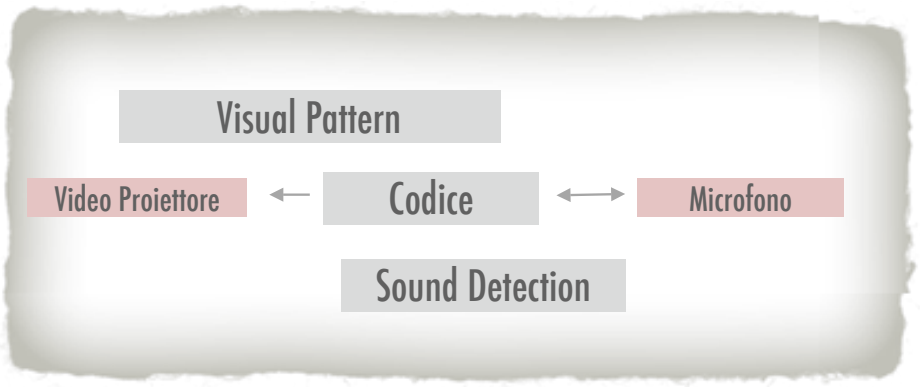
La città contemporanea è l'espressione del processo di esternalizzazione delle immagini grafiche. Le strade sono perimetrate da poster pubblicitari, ovvero pillole iconografiche orientate a farci percepire la necessità di un prodotto. Il processo di esternalizzazione delle immagini inizia con la scrittura e l'architettura. Possiamo facilmente vedere come la struttura e le architettrure di una città siano l'immagine percepita di un sistema sociale. La chiesa, il palazzi, i giardini, le periferie sono icone informative della realtà.
L'arte contemporanea è un grande ventaglio che si è aperto ai nuovi strumenti digitali. Come ogni nuovo strumento, il computer in rapporto con l'arte, ha prodotto una modificazione della realtà e dello spazio sociale. Dall'invenzione della stampa a caratteri mobili che ha visto il proliferare di biblioteche e il sedimentare della conoscenza, fino ai cellulari che hanno ammazzato le cabine telefoniche che abitavano le strade fino alla fine del secolo scorso. Il computer e l'arte ha generato la *computer art* e nuovi spazi in cui fruirne o essere messa in scena. Se oggi le tele abitano musei, gallerie d'arte e fiere, allo stesso modo la computer art ha visto nascere festival, eventi, meeting, nuova conoscenza e nuovi interrogativi sul ruolo della *computer art*.
Una delle peculiarità della computer art è di interrogarsi sull'interazione uomo-macchina, cercare soluzioni, ispirare percorsi.
In questo primo ciclo di esercizi lavoreremo alla creazione del codice per un installazione di *computer art* che utilizza la videoproiezione architetturale per illuminare con una nuova luce interattiva la facciata di un palazzo o il perimetro di una pizza.
City Scape è l'utilizzo delle architetture come nuovi spazi per la computer art. L'installazione **Visualizing Urban Noise** nasce proprio dal voler mostrare il livello di rumore che si genera in una città.

Visualizing Urban Noise

Il primo progetto è scrivere il *Codice* ActionScript per un'*installazione temporanea* che indaga il rapporto tra il flusso sonoro dell'ambiente circostante e la vibrazione di *visual pattern* proiettati su un'architettuta urbana.

Rimando in appendice un approfondimento sui dispositivi tecnologici di proiezione architetturale necessari per l'installazione.

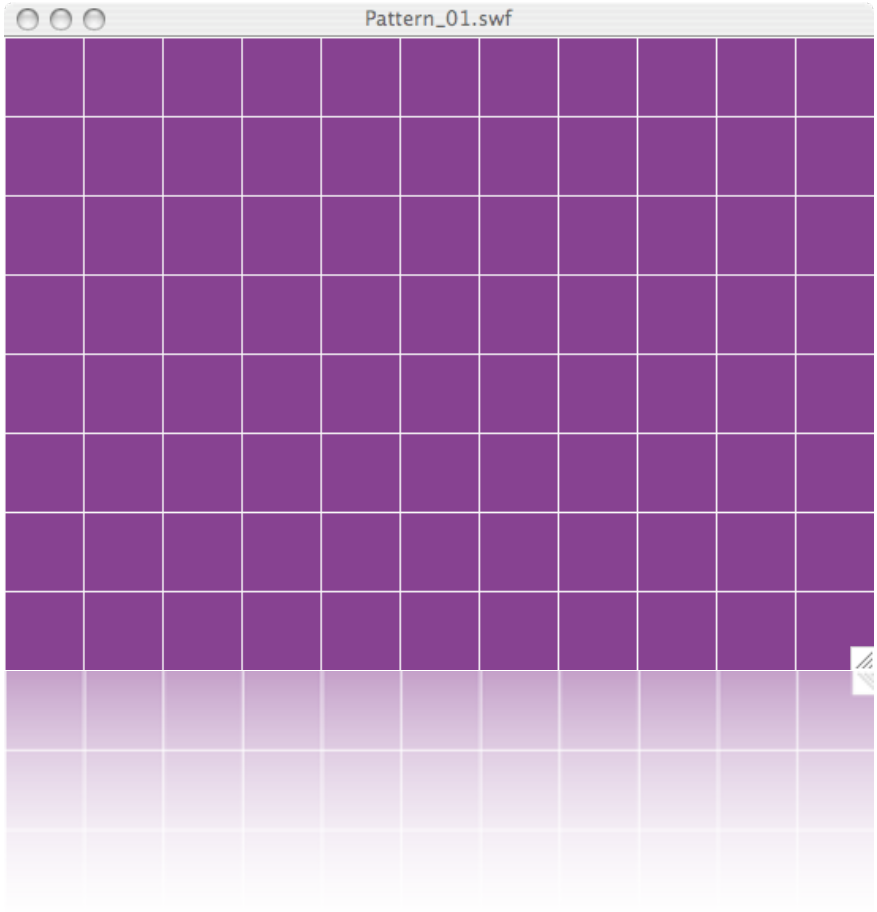
Concentriamoci invece sulla scrittura del Codice che permette di dare vita all'installazione.



La struttura base dell'installazione ci mostra che il codice è composto da due componenti: il Visual Pattern e il Sound Detection.
Iniziamo a scrivere il codice ActionScript che genera un Visual Pattern.

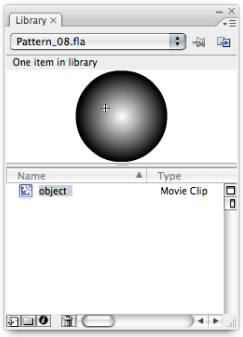
VisualPattern_01

regolarità



Iniziamo con *regolarità*. Un Visual Pattern si costruisce come nella realtà a partire da un modulo di base, che chiamiamo *object*. Il problema è come ripeterlo sullo *Stage* in maniera ordinata, in modo da riempire interamente lo spazio delimitato delle dimensioni espresse nelle Proprietà del Filmato, in maniera tale che mandato in esecuzione disegni un *pattern visivo* composto da moduli di base affiancati.
E' buona prassi suddividere il problema, in questo caso della *regolarità,* in passi successivi espressi in linguaggio naturale, che verranno poi tradotti in Actionscript.

Il problema Visual Pattern



Disegniamo un modulo di dimensioni 50x50px e lo convertiamo in un simbolo MovieClip, istanziandolo con il nome *object*.
La nostra funzione *makePattern* dovrà *copiare* ripetutamente il modulo *object* sullo schermo, a partire dal vertice il alto a sinistra (lo zero delle coordinate dello *Stage*). Quante righe e quante colonne saranno disegnate? Dipende dalle dimensioni dello stage e da quelle del modulo. Come faccio a dire con Actionscript *ripeti per un certo numero di volte delle azioni?*
La risposta sta nell'istruzione di base *for*, che come vedremo nella traduzione in Actionscript assolve proprio a questo compito.

Vediamo come appare la funzione *makePattern* in linguaggio naturale:

1. **Prendi** un nuovo oggetto dalla Libreria
2. **Copialo e posizionalo** al fianco del precente, *se è il primo inizia dal vertice in alto a sinistra dello Stage* (0,0).
3. Quando una riga è stata completata, **ripeti** dalla successiva
4. Quando lo Stage è completo, **termina**.

In quattro passi siamo riusciti a descrivere l'azione di copiare ripetutamente un oggetto in maniera ordinata e regolare sullo *Stage*.

Let's code!

Utilizziamo 6 variabili globali, ovvero valori visibili in tutto l'ambiente di programmazione.

```
var stagex:Number = Stage.width; // la misura delle x dello Stage
var stagey:Number = Stage.height; // la misura delle y dello Stage
```

Le dimensioni del modulo di base che verrà duplicato su tutto lo schermo in modo da formare una griglia ordinata.

```
var incx:Number = 50;
var incy:Number = 50;
```

Infine il numero di moduli sulle asse della x (*maxx*) e della y (*maxy*). Queste due variabili ci serviranno per controllare il doppio ciclo for che come la navetta del telaio farà correre i numeri di una variabile da un valore di partenza, fino ad uno di arrivo.

```
var maxx:Number = stagex/incx; // numero di colonne
var maxy:Number = stagey/incy; // numero di righe
```

Definiamo la funzione makePattern.

```
function makePattern() {
```

La funzione *makePattern()* utilizza una variabile locale *i* che tiene il conto degli oggetti e assegna ad ognuno un identificatore unico.

```
    var i = 0; // conta il numero totale di oggetti
```

Possiamo immaginare il *ciclo for* come un cronometro che parte ad un determinato momento e inizia a contare finchè non arriva ad un altro preciso momento.

```
for (var iy = 0; iy<=maxy; iy++) {
```

Leggiamo il primo ciclo for come la varibile *iy* che inizia a contare da zero e si ferma quando raggiunge il numero totale di righe sullo schermo (la variabile *maxy*). Ad ogni ciclo il valore di *iy* si incrementa di uno (*iy*++).
Quindi al primo ciclo *i* vale zero, al secondo uno, ecc...

```
for (var ix = 0; ix<=maxx; ix++) {
```

Il secondo *ciclo for* posiziona un nuovo oggetto al fianco del precedente, a partire dalla colonna zero.
Buona parte del lavoro ormai è stato fatto, non ci resta che utilizzare la funzione Actionscript *attachMovie* per **copiare** sullo *Stage* un oggetto dalla Libreria e controllare le sue proprietà di posizione (_x, _y), e il gioco è fatto!

```
attachMovie("object", "object_"+i,_root.getNextHighestDepth());
this["object_"+i]._x = ix*incx;
this["object_"+i]._y = iy*incy;
i++;
}
}
}
```

Non dimentichiamoci di chiamare la funzione alla fine del nostro codice altrimenti non succede nulla!

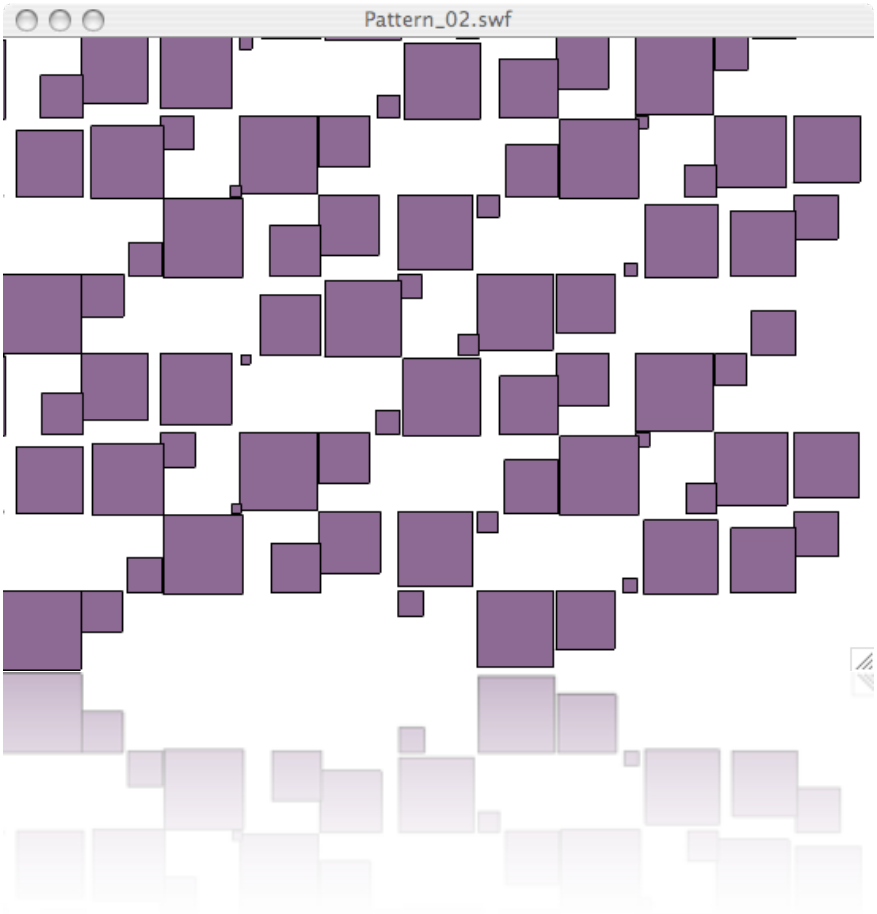
```
makePattern();
```

Esercizio per la mente

Sperimentare diverse forme dell'oggetto e spiegare cosa succede.

Pattern_02

regolarità + irregolarità



Abbiamo visto come scrivere la funzione **makePattern** che implementa un semplice algoritmo per esplorare lo spazio dello schermo in maniera ordinata. Se volessimo realizzare qualcosa di disordinato? In fondo la natura nel suo intimo mantiene un ordine che ai nostri sensi appare piuttosto caotico. Basti immaginare le coste frastagliate, oppure la jungla con il suo groviglio di fiori, piante ed alberi, il canto delle cicale in estate. Nel 1931, all'alba del computer, Alan Turing nel celebre saggio *Sui numeri calcolabili* immagina "che il calcolatore digitale contenga un generatore di numeri a caso" e ci accompagna in territori del calcolo che portano fino alla percezione extrasensoriale. Quella che nel 1931 era solo un'ipotesi di generatore di numeri casuali, ben presto nei primi computer si concretizzò nella funzione random. Utilizziamo la funzione random per lasciare la scelta al computer, per farci sorprendere. Scopriremo come la casualità inserita nel codice infonde vita ai nostri progetti.

La prima parte del codice rimane identico a quello del primo esercizio tranne la chiamata alla nuova funzione *randomProperties()* all'interno di *makeProperties()*.

```
function makePattern() {
          var i = 0;
          for (var iy = 0; iy<maxy; iy++) {
                 for (var ix = 0; ix<maxx; ix++) {
                 attachMovie("object","object_"+i,_root.getNextHighestDepth());
                        this["object_"+i]._x = ix*incx;
                        this["object_"+i]._y = iy*incy;
                        randomProperties(i);
                        i++;
                 }
          }
}
```

Dopo che l'oggetto è stato posizionato viene chiamata la funzione randomProperties che agisce sui valori di scala, trasparenza e rotazione delle rispettive proprietà *_xscale* e *_yscale*, *_alpha*, *_rotation*. La funzione *random()* di Actionscript viene chiamata su un valore che ne definisce l'intervallo di scelta.
Ad esempio *random(10)* restituisce un valore a caso compreso tra 0 e 9.

```
function randomProperties(i) {
```

Inizialmente assegniamo alle variabili xscale e yscale due diversi valori generati a caso e compresi tra 0 e 199, quindi il nostro oggetto verrà disegnato tra un massimo al doppio delle sue dimensioni originali e un minimo a zero, ovvero scompare.

```
        // scale
        var xscale = random(200);
        var yscale = random(200);
```

Nota che ad ogni chiamata di random viene restituito un nuovo numero a caso! Adesso assegniamo il valore alla proprietà di scala.

```
        this["object_"+i]._xscale = xscale;
        this["object_"+i]._yscale = yscale;
```

Allo stesso modo le variabili a ed r servono per generare la trasparenza dell'oggetto e la sua rotazione.

```
        // alpha
        var a = random(100);
        this["object_"+i]._alpha = a;

        // rotation
        var r = random(360);
        this["object_"+i]._rotation = r;
}
```

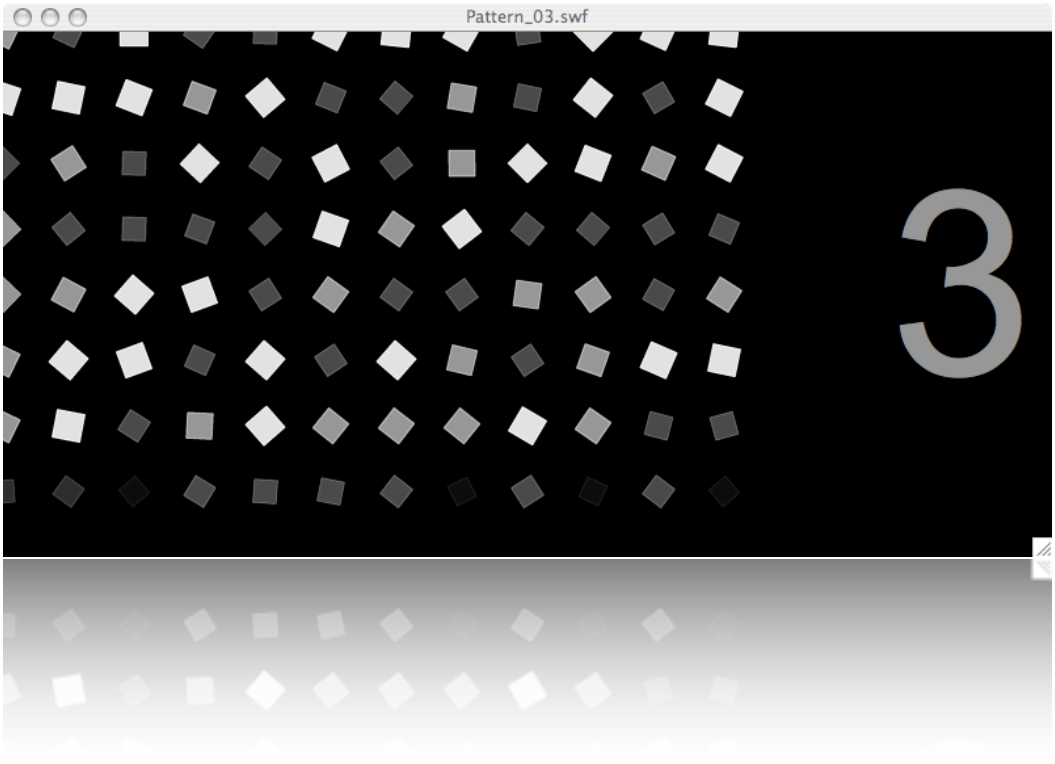
Il risultato è un pattern di oggetti che contina ad essere costruito su una griglia ordinata ma ogni oggetto decide che forma prendere.

Esercizio per la mente

Sperimentare diversi valori per gli intervalli della funzione random per ognuna delle proprietà e descriverne i comportamenti.

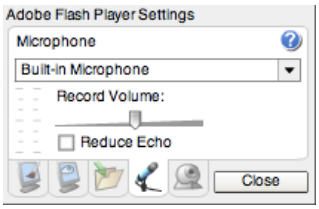
Pattern_03

Sound Detect



ActionScript ha una classe di funzioni che ci permette di controllare un **microfono** collegato al nostro computer (sia esso interno o esterno).

Viene visualizzata una finestra di dialogo relativa alla riservatezza che consente all'utente di scegliere se consentire o negare l'accesso al microfono. Assicurarsi che le dimensioni dello stage siano almeno di 215 x 138 pixel, ovvero le dimensioni minime richieste da Flash per visualizzare la finestra di dialogo.5



Una volta **agganciato** il microfono, l'unico valore che possiamo utilizzare in real time con ActionSctipt è il livello del suono in input dal microfono. Questo è comunque sufficiente per completare il codice dell'installazione **Interactive Visual Pattern**.

La funzione *makePattern()* rimane sostazialmente invariata, eccetto che contare gli oggetti che vengono disegnati. Il totale degli oggetti salvato nella variabile ci servirà successivamente per modificare le proprietà di ogni oggetto.

```
var i = 0; // conta quanti oggetti sono stati disegnati

function makePattern() {
          for (var iy = 0; iy<maxy; iy++) {
                for (var ix = 0; ix<maxx; ix++) {
                attachMovie("object","object_"+i,_root.getNextHighestDepth());
                      this["object_"+i]._x = ix*incx;
                      this["object_"+i]._y = iy*incy;
                      i++;
                }
          }
}
```

5 Manuale di Flash Ascionscript 8

Il problema SoundDetect

Per **agganciare** il microfono del computer ad ActionScript è sufficente scrivere queste poche righe di codice.

Inizialmente definiamo una variabile *my_mic* che contiene il *flusso sonoro* proveniente dal microfono del computer.

```
var my_mic = Microphone.get();
```

Con la funzione AS *attachAudio(variable)* rendiamo disponibile il flusso sonoro all'ambiente di programmazione.

```
_root.attachAudio(my_mic);
```

La funzione *setGain()* ci permette di variare l'*intensità del flusso sonoro* come se avessimo a disposizione un mixer audio.

```
my_mic.setGain(50);
```

Definiamo la variabile sensibile *level* uguale al valore dell'intensità del flusso sonoro del microfono in ogni istante. Per catturare questo valore AS mette a disposizione la funzione *activityLevel*.

```
var level = 0;
```

Per monitorare a *run time* una variabile, o più in generale per scrivere un codice che viene eseguito alla velocità del filmato - fps, AS mette a disposizione la funzione *onEnterFrame*.

Questa funzione esegue le istruzioni in essa contenute (tra parentesi graffe) al numero di frame per secondo impostata nelle proprietà generali del filmato.

```
this.onEnterFrame = function() {
          level = my_mic.activityLevel;
          soundDetect();
};
```

La funzione *soundDetect()* si comporta come la precente *randomProperties* con l'eccezione che viene chiamata a *run time* e i valori delle proprietà di ogni oggetto sono messe in relazione alla variabile sensibile *level.*

```
function soundDetect() {
```

La variabile *power* viene generata casualmente a *run time* ed è un valore di moltiplicazione del livello del microfono per amplificare l'effetto visivo sulle proprietà di ogni modulo. Il ciclo for di soundDetect modifica a *run time* tutti gli *i - numero totale -* oggetti sullo *Stage.*

```
          var power:Number;
          for (var k = 0; k<i; k++) {
                power = random(6)+1;
                this["object_"+k]._xscale = (level*power);
                this["object_"+k]._yscale = (level*power);
                this["object_"+k]._rotation += (level*power);
                this["object_"+k]._alpha = (level*power);
                }
}
```

Esercizio per la mente

Sperimentare diversi valori per gli intervalli di casualità della variabile *power*. Sperimentare diverse forme e grandezze dell'oggetto di base.
Utilizzare il videoproiettore.


NO DIOSSINA