```
// The locations of the various files
// algorithmically generated 'melody':
Document.open("/Users/richdpl/Documents/Music/Melodia/MelodiaSyr

// the programming for SC's interface with the unit
Document.open("/Users/richdpl/Documents/Music/Melodia/MelodiaSyr


// Load functions and synths
(
s.waitForBoot(
{ (
"~/Documents/Music/Melodia/MelodiaSynthFunc/SynthDefs/*".loadPat
"~/Documents/Music/Melodia/MelodiaSynthFunc/Functions/*".loadPat
) })
);

// add some reverb and test
(
~rev = Synth.new("reverb", [\inBus, ~effect]);
~myModNote.value( 3.rand, 880.rand, 2.0, 5.4, 0.4, 2.0*(2.0.ranc
)

////////////////////////////////////////////////////////////////
// Intro
////////////////////////////////////////////////////////////////
// Where to pitch this: originally it was mechanism, metaphor, m
interfaces, the hardware/software metaphor and how these compone
sprinkled in) to form an interesting musical performance/composi

//  Research focuses
1.   The physical interface: design, construction and use;
2.   The algorithmic control of music;
3.   The links between these two focuses;
4.   Different in acoustic and electronic music;
5.   These focuses lead to the following questions...

// Research questions
1.   What is the meaning of physical gesture as a cause of musi
2.   What is the relationship between physical gesture and musi
     metaphorical relationships?
```

3.   What does it mean to be skilled musically?
4.   What are the differences in aims and approaches between mus
5.   What are the importance of the motor skills involved in mus

// Other systems
1.   Nintendo Wii, Microsoft Natal, Sony Move and the Apple iPho
2.   G-Speak, Vicon systems, Gypsy MIDI, etc.;
3.   BBC News - Sony shows off its motion controller PlayStatio
4.   Natal Demo.

```
(
y = Window("Sony Move").front;
y.bounds = Rect(0, 0, 740, 500);
m = MovieView(y, Rect(10, 10, 720, 480));
m.path_("/Users/richdpl/Documents/Music/Canterbury/SonyMove.mp4"
)

(
y = Window("Microsoft Natal").front;
y.bounds = Rect(0, 0, 660, 380);
m = MovieView(y, Rect(10, 10, 640, 360));
m.path_("/Users/richdpl/Documents/Music/Canterbury/NatalDemo.mp4
)

(
y = Window("iPad").front;
y.bounds = Rect(0, 0, 870, 500);
m = MovieView(y, Rect(10, 10, 852, 480));
m.path_("/Users/richdpl/Documents/Music/Canterbury/iPadEdit.mp4"
)
```

// Linking real world data to software
1.   These systems now commonly allow significantly more physica
2.   these processes (musical or not) are usually highly limite
3.   If used musically, these limitations reflect the quality ar
4.   In general, they are analogical systems in that they seek t
      it creatively and/or metaphorically.

// Mechanism, metaphor, magic and music

1. What do I mean by metaphor?
2. Relationship of the physical and the algorithmic is the nat
   between physical action and software-driven reaction;
3. Current technology emphasises convincing metaphorical links
   and in music, physical modelling) - see the iPad;
4. Magical properties (also see Natal clip, about 3/4 of the w
5. The synthesiser - is it a musical instrument? Is a laptop

```
////////////////////////////////////////////////////////////////
// Demonstration
////////////////////////////////////////////////////////////////
/*
  •   Ultrasound sensor;
  •   Is instructed by the Arduino board to send out a ping;
  •   Processing on the chip detects its return and sends this do
  •   In this case the data is then used by me within SuperCollic
  •   Other devices about which I may have a chance to demonstrat
  •   This is a deliberately simple device looking at the impleme
      sensor.
  •   I'll be demonstrating some basic functionality, particularl
      the musical algorithms.
  •   It's likely that two of these units will be used to constru
      controlled by movement, will interlink...
*/
```

```
////////////////////////////////////////////////////////////////
// activate the arduino (and check pots)
////////////////////////////////////////////////////////////////
~ardMelodiaPingPot.value;

// close the connection if necessary
~ardMel.close; // close just the port you've just opened
SerialPort.closeAll; // close all ports.

// This starts the principal algorithm and sets sudden movement
(
~melodia.value;
~suddenMove = 1800;
)

// If there is no object inside the set distance, by default 20

~myDist = 4000;
~myDist = 1000;
~myDist = 500;
~myDist = 2000; // default

// So one simply holds one's hand in range and the algorithm wil
// One of the things that many people did when first introduced
the unit.  In general, with

~suddenMove = 1800;

// the unit only occasionally responded (when the melodic algori

~suddenMove = 400; // this works much better...
```

```
// ...and is a specific algorithm included to provide this respo
unit is more responsive.  If the value is too low, the unit resp
the task itself:

~suddenMove = 10;

~suddenMove = 400; // the default value;



///////////////////////////////////////////////////////////////
/*
This configuration of values is a clear example of how interacti
immediately apparent;

Almost invariably one's initial ideas have to be modified (and c

The current configuration isn't ideal, but it generally works ir

With the sudden movement algorithm set, this may well be trigger
movements of objects within range that one is not even consideri
where the movement of its back can, unbeknownst, trigger a respo
*/
///////////////////////////////////////////////////////////////
```

```
///////////////////////////////////////////////////////////////
// More detailed investigation of the melodic and rhythmic aspec
///////////////////////////////////////////////////////////////

// The melodia algorithm is not particularly sophisticated and i
there's a tendency for these configurations to reach a point at
forever incomplete.

// Here's the actual file if necessary:
Document.open("/Users/richdpl/Documents/Music/Melodia/MelodiaSyr

// Start the algorithm with default settings
(
~suddenMove = 1800; // so we know it's the algorithm
~melodiaTask.stop; ~melodiaTask.reset; // if it's going already
~melodia.value;
)

// again, nothing will happen if nothing's in range

// Frequencies are in fact derived from a row:
~pPatt = [ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ];
```

```
// taken originally from an earlier composition of mine.

// However, for a variety of reasons, and not least because the
make a huge difference.  If I change the row:
~pPatt = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ];
~pPatt = [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ];
~pPatt = [ 0, 0, 0 ];

// In addition to the pitches, there is a rhythmic element contc

~myDurArray = [ 4.0, 3.0, 2.0, 1.0, 1.33, 0.25, 0.25, 0.125, 0.0

// From this array, at each iteration of the melodic loop, one v
creates a fairly liberal result.  However, it's easy to create c
indicator more clear:

(
~melodiaTask.stop; ~melodiaTask.reset;
~melodia.value;
~density = 32.0;
~myDurArray = [ 0.25, 0.125 ];
)

// Now I can close the unit to hear the rhythm more precisely (t

~ardMel.close;
SerialPort.closeAll; // might need to do this too.

// so one can set different rhythmic tendencies
~myDurArray = [ 0.5, 0.25, 0.125 ];

~myDurArray = [ 0.5, 0.25, 0.0001 ];

~myDurArray = [ 0.75, 0.001 ];
~myDurArray = [ 0.5, 0.25 ];
~myDurArray = [ 0.75, 0.5, 0.25 ];
~myDurArray = [ 0.75, 0.5, 0.25, 0.125, 0.125 ];
~myDurArray = [ 0.2 ];
~myDurArray = [ 0.2, 0.25 ];
~myDurArray = [ 0.2, 0.25, 0.33, 0.4, 0.5 ];
~myDurArray = [ 0.125 ];
```

```
~myDurArray = [ 0.25, 0.125 ];
~myDurArray = [ 0.375 ];



// or change them algorithmically
(
~rhythmTask = Task({
    100.do({
        ~myDurArray = [ [ 0.25, 0.125 ], [ 0.375, 0.35 ] ].choos
        2.0.rand.wait;
    });
}).play;
)

// this stops that last task
~rhythmTask.stop; ~rhythmTask.reset;

// this brings us back to normal
(
~melodiaTask.stop; ~melodiaTask.reset;
~melodia.value;
)



~rev = Synth.new("reverb", [\inBus, ~effect]); // just in case

// if I start it again you immediately hear the effect:

~ardMelodiaPingPot.value;

(
~melodiaTask.stop; ~melodiaTask.reset;
~melodia.value;
~density = 32.0;
~myDurArray = [ 0.5, 0.25 ];
)

// Now, if I move in front of the unit, you'll hear a very clear
generically with the variable
```

```
~density = 8.0;

// Here, the lower the value the more potential movement there i

~density = 160.0;

// the density value is acting as a 'gear'.

~density = 16.0;
```

```
////////////////////////////////////////////////////////////////
// Movement algorithm
// Most technical part
// NB the posting codes below can be commented in melodia after
////////////////////////////////////////////////////////////////
Document.open("/Users/richdpl/Documents/Music/Melodia/MelodiaSyr
// The quantity of movement is chosen through the following algo
emerging from the unit

~postOrigVal = true;
~postOrigVal = false;

// is put into pairs with a simple switch

~postPairs = true;
```

```
~postPairs = false;

// and the second value of each pair is subtracted from the firs

if ( myDiff == true , { myDiffVal1 = pingVal; myDiff = false; }
    myDiffVal = ((myDiffVal1 - myDiffVal2).abs);

~postDiffVal = true;
~postDiffVal = false;

// As is to be expected, the values are rather jumpy as, even wi
plenty of quantisation taking place.  For this reason, all value

~postav = true;

// whose size can be varied (so creating smoother results or a f
~mySize = 100;
~mySize = 4;
~mySize = 1;
~mySize = 10;

~postav = false;

// which gives rise to a single value:
~postavmean = true;
~postavmean = false;


// this might need restarting to get a clearer value in the winc
~ardMel.close;
SerialPort.closeAll;
~ardMelodiaPingPot.value;

// which is graphically illustrated in the window at the top of
with the variable density above, to control the 'tempo' of the c

(
~melodiaTask.stop; ~melodiaTask.reset;
~melodia.value;
~density = 16.0;
~myDurArray = [ 0.25 ];
```

```
)

// the original duration array
~myDurArray = [ 4.0, 3.0, 2.0, 1.0, 1.33, 0.25, 0.25, 0.125, 0.0

// and pitch array
~pPatt = [ 0, 3, 1, 0, 9, 10, 11, 1, 7, 0, 8, 2, 4, 6, 5 ];
```

```
////////////////////////////////////////////////////////////////
// Bonus bits
// Melismas
////////////////////////////////////////////////////////////////

// clearer to begin with without the unit
```

```
~ardMel.close;
SerialPort.closeAll; // might need to do this too.

// Included in the melodia algorithm is a value that represents
is

~myTupletCoin = 0.001
// which is effectively off.

// 50/50 chance of a melisma with each loop
~myTupletCoin = 0.5

// easier to hear the rhythms
~myDurFactor = 0.2

// only melismas
~myTupletCoin = 1.0

// 'semiquavers'
~myTupletArray = [ 4.0 ]

// no pauses from the 'melody' and readjust the duration
~myDurArray = [ 0.01 ]
~myDurFactor = 10.2

// 'semiquavers' and 'triplets'
~myTupletArray = [ 4.0, 3.0 ]

// and so on
~myTupletArray = [ 3.0, 4.0, 5.0 ]

~myTupletArray = [ 0.51, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.
~myTupletArray = [ 4.0, 5.0, 7.0, 13.0 ]
~myTupletArray = [ 5.0, 6.0, 100 ]
~myTupletArray = [ 3.0, 4.0 ]
~myTupletArray = [ 0.51 ]


// back to the original
(
~melodiaTask.stop; ~melodiaTask.reset;
```

```
~melodia.value;
~density = 16.0;
~myDurArray = [ 4.0, 3.0, 2.0, 1.0, 1.33, 0.25, 0.25, 0.125, 0.0
~myTupletCoin = 0.5;
~myDurFactor = 0.1;
)


~ardMel.close
SerialPort.closeAll // might need to do this too.

// turn unit on again
~ardMelodiaPingPot.value
// then you must do this again
// and remember the pots are now effective
(
~melodiaTask.stop; ~melodiaTask.reset;
~melodia.value;
~density = 32.0;
~myTupletArray = [ 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0
~myDurArray = [ 4.0, 3.0, 2.0, 1.0, 1.33, 0.25, 0.25, 0.125, 0.0
~myTupletCoin = 0.5;
~myDurFactor = 0.1;
)
```

```
////////////////////////////////////////////////////////////////
// Pots and Performance
////////////////////////////////////////////////////////////////
// You'll notice how inspite of the unit, there's plenty of code
intentions, I ended up including a couple of potentiometers to c
four!).

// (Potential area for more work here: pros and cons of developi
composition/performance).

// I wanted originally to use a ribbon potentiometer (like a fac
overestimated reality and it didn't work in the way I expected.

// In this case, the two I implemented set the variable ~myDurFc

// Demonstrate with the above.

// Finally, I'm very aware of two issues:  one is that use of th
'synthesiser', something of which I'm very suspicious.  Also, th
physical action and coding/evaluating - still a physical action,

// A good example happened at a performance earlier this month,
a function three times, thinking it was a function that created
```

```
~h3.value
```

```
// In fact, I evaluated this three times:
```

~h2.value

```
// This function, unfortunately generated 2 minutes of sustainec
to wait for over two minutes while the functions played out as I
evaluated functions.

// In the real world, objects that created such different sounds
probably colour: a bass drum and piccolo. In code, they differec
future, I am considering making the triggering of functions such
objects whose aspect metaphorically relates to their effect.

// everything off
~ardMel.close
SerialPort.closeAll // might need to do this too.




// Gaggle: something I prepared earlier
(
y = Window("Gaggle").front;
y.bounds = Rect(0, 0, 888, 500);
m = MovieView(y, Rect(10, 10, 868, 480));
m.path_("/Users/richdpl/Documents/Music/Canterbury/hci2009_p2g_1
)
```