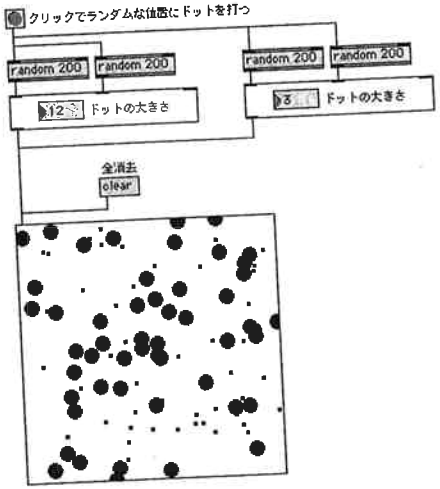
あとは、これまでのパッチで利用していたpaintdotのパッチ・オブジェクトとナンバー・ボックスは削除し、bpatcherオブジェクトを作成する。次に、bpatcherオブジェクトのインスペクターでpaintdotパッチを指定する。また、Borderのチェック・ボックスにチェックを付けて境界線を描くように指定したが、これは好みで選べばよい。そして、bpatcherオブジェクトの大きさやパッチの表示部分を調整する。

最後に、bpatcherオブジェクトと他のオブジェクトをパッチ・コードでつなぐ。これでパッチの修正は完了だ。bpatcherオブジェクト内のナンバー・ボックスをドラッグすれば、ドットの大きさが変わるだろう。

■3-9-32　bpatcherオブジェクトを利用したパッチ



06/5/3

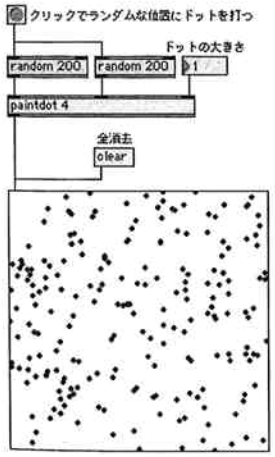
3-10 繰り返しの処理

プログラミングをしていく上で、1つの処理を繰り返して行いたい場合がある。例えば、ランダムな位置に点を100個打ちたい場合には、どのようにすればよいのだろうか？単純にパッチを100個並べることでも実現できるが、それではあまりに効率が悪い。あとになってから何らかの変更をすることも困難だ。そこで、ここでは一定の処理を繰り返して行う手法やオブジェクトについて説明を行うことにしよう。繰り返し処理はプログラミングの基本的な利点の1つであり、さまざまな応用が可能になるだろう。

無限ループによる繰り返し

paintdotオブジェクトを使って、手軽にドット(小円)を描けるようになったが、ランダムな位置にドットを描くには、buttonオブジェクトをクリックしなければならない。このままではlcdオブジェクトの描画領域を埋め尽くすように数多くのドットを描こうとすると手が疲れてしまうだろう。ここでは、自動的に何個ものドットを描く方法を考えてみよう。

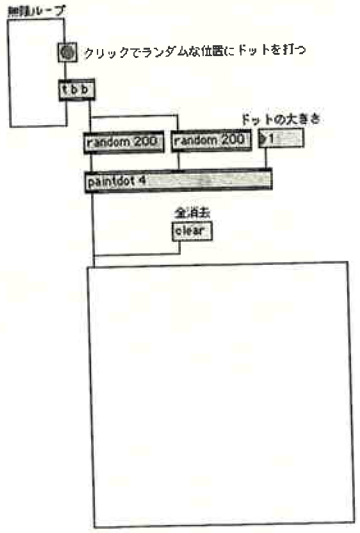
■3-10-1　ランダムな位置に手動でドットを描くパッチ



これまでのパッチでは、buttonオブジェクトをクリックするとbangメッセージが出力され、bangメッセージによってrandomオブジェクトがランダムな数値を出力し、その数値を座標としてpaintdotオブジェクトが描画メッセージをlcdオブジェクトに送っていた。そこで、描画の自動化のためにはbangメッセージを自動的に繰り返して出力するメカニズムを作ればよいことになる。これにはいくつかの方法があるが、まず、間違った、そして危険な、無限ループによる方法を紹介しよう。

3-10-2の例では、buttonオブジェクトから出力されたbangメッセージによって、triggerオブジェクトから2つのbangメッセージが出力する。1つのbangメッセージはドットを描く処理を行い、もう1つのbangメッセージは元のbuttonオブジェクトに戻る。何らかのメッセージを受け取ったbuttonオブジェクトは、黄色く点滅するとともにbangメッセージを出力する。したがって、1度buttonオブジェクトをクリックしてbangメッセージが出力されれば、ドットを描くと同時にbuttonオブジェクトにbangメッセージが送られ、再びbuttonオブジェクトからbangメッセージが出力される。このことが何度も繰り返して行われることで、ランダムな位置にドットを描くことが繰り返されるはずである。

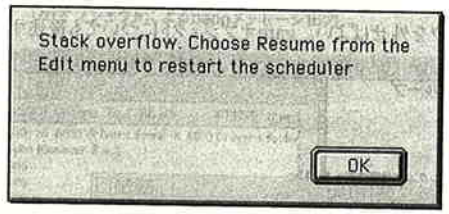
■3-10-2　単純なループによる繰り返し処理



しかし、このパッチを実行すると、何十個かのドットを描くものの、すぐに"スタック・オーバーフロー"というアラートが表示されて、パッチの動作が止まってしまう。あるいは、Maxの異常終了や、コンピューターのハングアップを引き起こすかもしれない。スタック・オーバーフローは、Maxが作業するメモリーを使い果たした状態のことだ。こうなるとOKボタンをクリックしてアラートを閉じても、パッチはそれ以上動作しない。

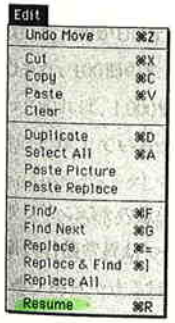
実は、3-10-2のようなパッチは、いわゆるメッセージの無限ループを引き起こすので、極めて危険な処理と言える。

■3-10-3　スタック・オーバーフローのアラート



不幸にしてスタック・オーバーフローになってしまった場合でも、Editメニューの Resumeを選べば、使用中のメモリーが解放され、パッチが動作するようになる。もちろん、もう1度buttonオブジェクトをクリックすれば、同じようにスタック・オーバーフロー状態になるので、何度も試すべきでないのは言うまでもない。

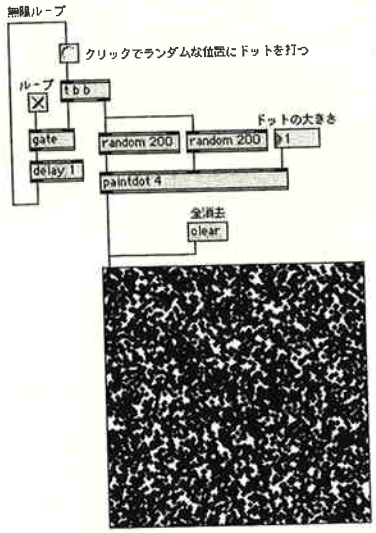
■3-10-4　EditメニューのResume



無限ループによるスタック・オーバーフローを解決するには、後述するdelayオブジェクトを用いて、わずかにbangメッセージの出力を遅らせばよい。

delayオブジェクトは、一定時間後に新しいbangメッセージを発生させ、それは直前のメッセージのつながりとは独立している。つまり、buttonオブジェクトを何度もクリックした場合と同じ処動作になり、スタック・オーバーフローにならない。しかし、動作上は無限ループになるので、どこかで処理を止める仕掛けが必要になる。このためには、gateオブジェクトによって、メッセージの流れを中断できるようにする。gateにつながるtoggleオブジェクトをチェックした上でbuttonオブジェクトをクリックすれば、あっと言う間にlcdオブジェクトがドットで埋め尽くされていくだろう。この動作を止めるには、toggleオブジェクトのチェックを外せばよい。gateオブジェクトについても後述する。

■3-10-5　正しく動作する無限ループ

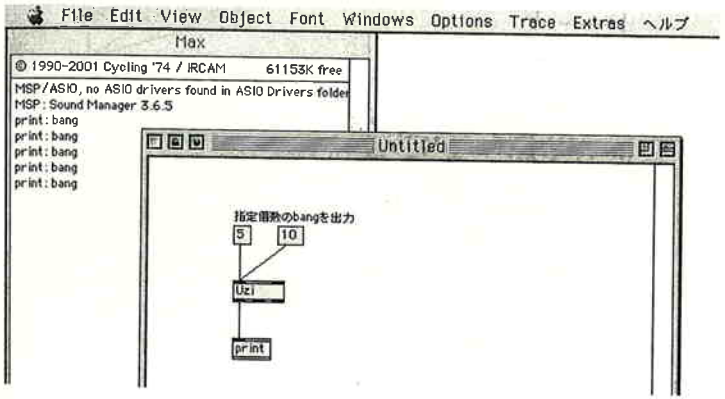


このようにすれば、スタック・オーバーフローを起こさず、任意のタイミングで処理を中断できるループ処理を行うことは可能だ。しかし、無限ループは異常動作の原因になりやすいので、特別な事情がない限り、このような処理は避ける方が賢明である。

Uziオブジェクトによる有限回数の繰り返し

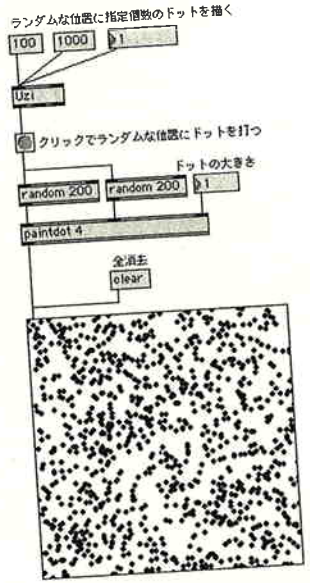
次に、100個や1,000個など指定した個数のドットをランダムに描くパッチを作成しよう。このように、繰り返す回数が有限回であれば、Uziオブジェクトを利用することができる。Uziオブジェクトは、受け取った整数の数だけbangメッセージを出力する。例えば5というメッセージをUziオブジェクトに送ると、5個のbangメッセージが出力される。これは瞬間的に行われるので判別できないが、printオブジェクトによってMaxウィンドウに表示させると、確かに5個のbangメッセージが出力されたことが分かる。

■3-10-6　Uziオブジェクトによるbangメッセージ出力

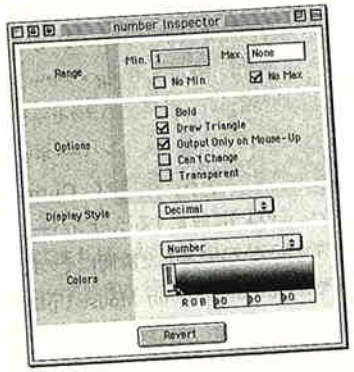


この仕組みを利用して作成した3-10-7のパッチでは、100のメッセージ・ボックスをクリックすれば、100がUziオブジェクトに送られるので、100個のbangメッセージが連続的に出力され、100個のドットがランダムな位置に描かれる。1000のメッセージ・ボックスをクリックすれば、1,000個のドットが描かれる。ドットの個数はナンバー・ボックスでも指定できるようにしているが、この場合、最小値を1にするだけでなく、Output Only on Mouse-Upをチェックしておいた方がよい。このように設定すると、ナンバー・ボックスをドラッグしている間の数値は出力せず、マウス・ボタンを離した瞬間にその時点の数値を出力することになる。なお、デフォルトではOutput Only on Mouse-Upはチェックされていないので、ドラッグして数値が変化するにつれて、逐次数値が出力される。

■3-10-7 指定個数のドットを描く

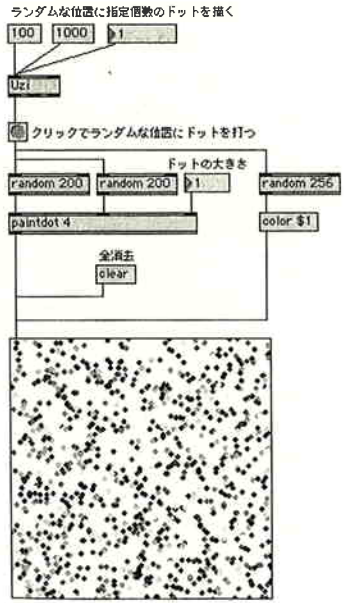


■3-10-8 ナンバー・ボックスのインスペクター



繰り返してドットを描き続けると、やがてはlcdオブジェクトの全領域を黒く塗りつぶしてしまう。そこで、ランダムに色を付けてドットを描くことにしよう。ドットを描く色は、colorメッセージとして0から255までの範囲の整数を指定する。この整数は、Maxが持つカラー・パレットのインデックス番号に相当するので、0から255まで間のランダムな数値を発生させるためには、新たにrandom 256というオブジェクトを作る必要がある。そして、その出力をcolor $1というメッセージ・ボックスを通してlcdオブジェクトに送れば、ランダムな配色でドットが描かれるようになる。

■3-10-9 ランダムな色で円を描く



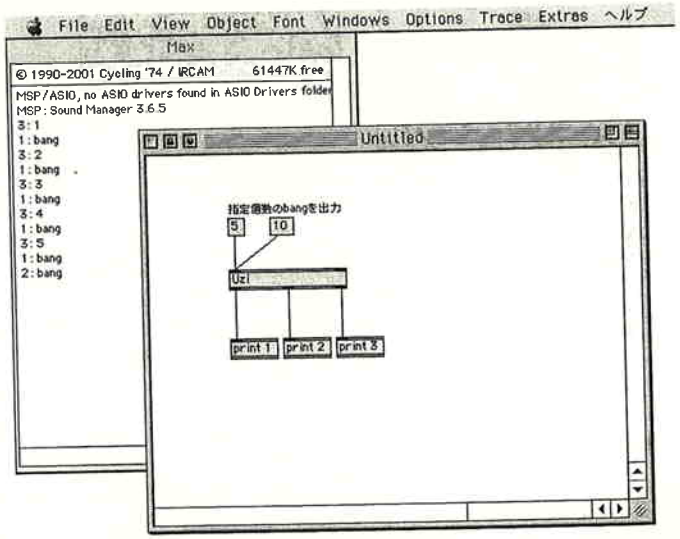
Uziオブジェクトのインデックス番号

ある処理を繰り返す場合に、それが何回目の繰り返しであるかを知りたいことがあ

る。Uziオブジェクトを使って繰り返し処理を行う場合は、Uziオブジェクトの第3アウトレットから何回目であるかのインデックス番号が出力されるので、これを利用すればよい。100というメッセージをUziオブジェクトに送れば、第1アウトレットから100個のbangメッセージが出力されるが、同時に第3アウトレットから1から100までの整数が出力されることになる。もちろん、right-to-left orderのルールに従って、第3アウトレットから1を出力後に第1アウトレットからbangメッセージを出力し、その次に第3アウトレットから2を出力したあとに第1アウトレットからbangメッセージを出力する、ということを100回繰り返す。ちなみに、100回の出力が終われば、次の瞬間に第2アウトレットからbangメッセージが出力される。

3-10-10のパッチでは、Uziオブジェクトの各アウトレットの出力を、printオブジェクトを使ってMaxウィンドウに表示している。printオブジェクトにアーギュメントを指定した場合は、そのアーギュメントを先に伴って表示が行われるので、どのprintオブジェクトが表示したかを示すことができる。ここでは、対応するアウトレット番号をprintオブジェクトのアーギュメントにしている。先の説明をMaxウィンドウの表示で確かめておこう。

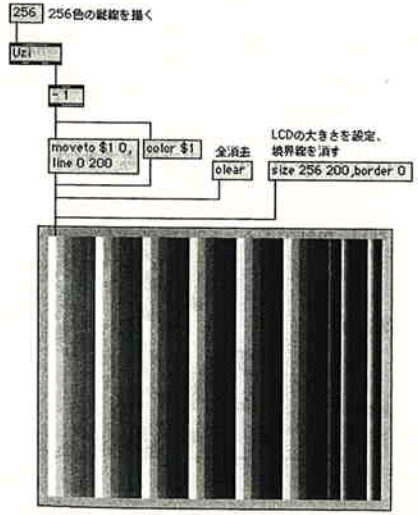
■3-10-10　Uziオブジェクトの出力



ではカラー・パレットの256色全色を描いてみよう。ここではドットではなく、縦の直線として描画する。このためには256ピクセルが必要になるので、lcdオブジェクトにsize 256 200,border 0というメッセージを送り、lcdオブジェクトの横幅を256ピクセルにする。border 0はlcdオブジェクトの境界線を消すメッセージであり、border 1なら境界線が描かれる。lcdオブジェクトでは目に見えない仮想のペンを動かすことで線を描く。ペンをlcdオブジェクトから離して動かすにはmoveメッセージとmovetoメッセージがある。いずれもアーギュメントの2つの数値によってX座標とY座標を指定するが、moveは現在のペンの位置からの相対座標、movetoはlcdオブジェクトの左上原点からの絶対座標になる。これらのメッセージでは線は描かれず、ペンが移動するだけだ。

一方、ペンをlcdオブジェクトに付けて線を描くには、lineメッセージとlinetoメッセージを用いる。これらもアーギュメントでX座標とY座標を指定し、lineは相対座標、linetoは絶対座標で表す。実際のパッチとしては3-10-11のようになる。まず、Uziオブジェクトに256メッセージを送り、第3アウトレットから出力される1から256までの整数を利用する。この整数は、−1オブジェクトによって減算が行われ、0から255までの数値にな

■3-10-11　カラー・パレット256色の縦線を描く



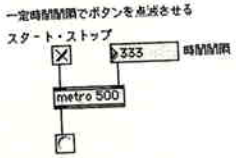
る。この数値をcolor $1のメッセージ・ボックスに送り、ペンの色を指定する。同じ整数をmoveto $1 0,line 0 200のメッセージ・ボックスに送り、縦線を描く。つまり、カラー・パレットのインデックス番号として0から255まで変化するにつれて、X座標は0から255まで変化しながら、繰り返して縦線を描くことになる。ちなみに、インデックス番号0の色は白なので、パッチ・ウィンドウの色と区別がつかない。これを解消するために、lcdオブジェクトの背後に、ひとまわり大きなパネル・オブジェクトを配置している。

metroオブジェクトによる一定時間間隔の繰り返し

一定の時間ごとに処理を繰り返すには、metroオブジェクトが便利だ。metroオブジェクトは、アーギュメントまたは第2インレットに受け取る数値をmsec単位の間隔として、一定時間ごとにbangメッセージを出力する。第1インレットはmetroオブジェクトの動作を決め、0以外の数値を受け取れば動作を開始し、0なら動作を止める。したがって、metroオブジェクトの第1インレットにはtoggleオブジェクトを接続するのがよいだろう。こうすることでtoggleオブジェクトをクリックしてチェック・マークを付ければ1を出力し、もう1度クリックしてチェック・マークを外せば0を出力するようになっている。

次のような簡単なパッチで試してみよう。toggleオブジェクトをクリックしてチェック・マークを付ければ、metroオブジェクトが動作を始め、500msec間隔でbangメッセージを出力するので、それを受け取ったbuttonオブジェクトが点滅する。ナンバー・ボックスの値を変えれば、時間間隔が変化する。200msec以下の短い時間間隔になれば、buttonオブジェクトは常に点灯しているように見えるので確認できないが、それでもmetroオブジェクトからは短い時間間隔でbangメッセージが出力されている。toggleオブジェクトをもう1度クリックすれば、metroオブジェクトの動作が止まり、buttonオブジェクトの点滅も止まる。

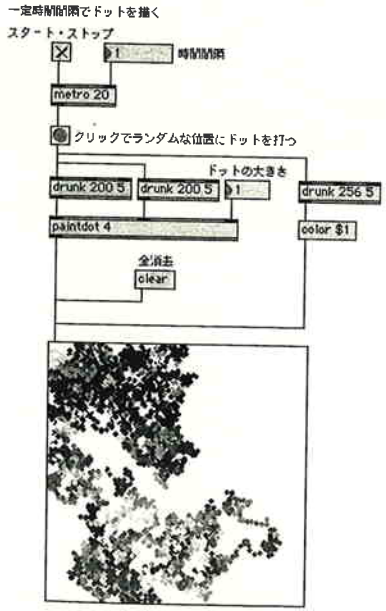
■3-10-12　metroオブジェクトの基本動作



では、一定時間間隔で繰り返してドットを描くパッチを作ってみよう。今回は円の位置をrandomオブジェクトで決めるのではなく、drunkオブジェクトを用いる。また、色もdrunkオブジェクトを使って設定する。

3-10-13のパッチでは、toggleオブジェクトをチェックすれば20msec、すなわち0.02秒間隔でmetroオブジェクトがbangメッセージを出力するので、それに合わせてbuttonオブジェクトが点滅し、ドットが描かれる。ナンバー・ボックスを使って、時間間隔を指定してもよい。これによって、円が千鳥足で歩いているようなアニメーションが描かれる。これは、metroとdrunkオブジェクトによってドットの位置を徐々に変化させながら描いているためだ。誌面では時間経過による変化の様子は分からないが、randomによる描画とは雰囲気がまったく異なっているのは分かるだろう。

■3-10-13　一定時間間隔でドットを描く



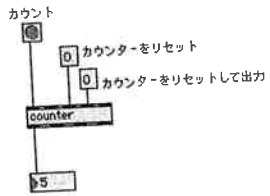
counterオブジェクトによるカウント

Uziオブジェクトでは、第3アウトレットによって何回目の繰り返しであるかが分かった。これに対して、metroオブジェクトは単にbangメッセージを出力するだけなので、何回目であるかが分からない。そこで何回目であるかは、counterオブジェクトを使って調べることができる。

counterオブジェクトは、内部に数を数えるカウンターを持っている。カウンターの初期値は0だ。そして、bangメッセージを受け取れば、その値を第1アウトレットから出力し、カウンターの値を1増やす。その結果、bangメッセージを送る度に、0、1、2、3……という具合に数値が出力される。3-10-14のパッチで、buttonオブジェクトを何度かクリックすれば、ナンバー・ボックスに表示される値が1ずつ増えていくのが確認できる。

counterオブジェクトの第3インレットに整数を受け取った場合は、カウンターの値を設定するだけで出力は行わない。第4インレットに整数を受け取った場合は、カウンターの値を設定するとともに、その数値を出力する。したがって、第3インレットに0を送れば、ナンバー・ボックスの表示は変わらず、次にbuttonオブジェクトをクリックすれば0が表示される。これに対して、第4インレットに0を送れば、すぐにナンバー・ボックスの表示が0になる。そして、buttonオブジェクトをクリックすれば1が表示される。

■3-10-14 counterの基本動作

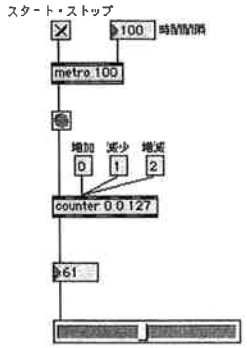


counterオブジェクトのカウンターは、特に指定しない限り、どんどん大きくなっていく。そこで、アーギュメントなどによって、一定の範囲で増減するように指定することができる。counterオブジェクトのアーギュメントは3つあり、順に増減の方向、最小値、最大値を整数で指定する。増減の方向は、0なら増加、1なら減少、2なら増減になる。これは、第2インレットで変更することもできる。

3-10-15の例では、counter 0 0 127とアーギュメントを指定し、metroオブジェクトと組み合わせている。これでtoggleオブジェクトをクリックするとmetroが100msec間隔でbangメッセージを出力し、それを受け取ったcounterがカウンターの値を出力し、ナンバー・ボックスに表示する。その様子が視覚的に分かるようにhsliderオブジェクトも付けている。このようにすれば、自動的にカウンターの値が増加していくのが分かるだろう。そして、カウンターの値が127になると、次の値は0に戻って、0から127まで増加することを繰り返す。

また、counterオブジェクトの第2インレットに1を送れば、カウンターの値が減少するようになる。この場合は、カウンターの値が0になれば、次は127になり、127から0へと減少していく。第2インレットに2を送った場合は、カウンターの値が増加して127になれば、今度はカウンターの値が減少するように変化する。そして、カウンターの値が0になると、再びカウンターの値が増加するようになる。この結果、hsliderオブジェクトは、左右を往復するような動きになるわけだ。

■3-10-15 metroオブジェクトによるカウンター駆動とcounterオブジェクトの増減方向



次の例では、lcdオブジェクトに大きな円を描き、次第に小さくなるように同心円を連続的に描いている。また、円の色はグレースケールの明暗として増減させている。大きな円はドットと言うには語弊があるが、このような描画もpaintdotオブジェクトを利用することができる。lcdオブジェクトの大きさを200ピクセル×200ピクセルとすると、円の