キーボードを使う（1）

コンピュータを使う際に、キーボードは重要な入力装置であるが、これをピアノの鍵盤のように、演奏のための装置として用いることもできる。

まず、ここではＧのキーを押すと 330Hz の音が出るようにしよう。

ポイントは、GUI にキーの働きを埋め込む、ということである。キーボードの操作は、GUI の機能の一部として記述される。なぜそのようにしなければならないか、理由の説明は割愛するが、キーボードを使えるようにするためには、ボタンやスライダーがないにせよ、最低限 GUI のウィンドウが必要となる。

ウィンドウだけでは愛想がないので、ボタンで音色を選び、スライダーで音量を調節できるようにしよう。

まず、音色の違う音を３種類用意する。

```
SynthDef.new( \sineTone, { arg freq=440, amp=0.1, pos=0;
	var src, env, out;
	src = SinOsc.ar( freq );
	env = EnvGen.kr(
			Env.new(
				[0, 1, 0.2, 0],
				[0.01, 0.2, 0.1]
				),
			levelScale: amp,
			doneAction: 2
			);
	out = Pan2.ar( src*env, 0 );
	Out.ar( 0, out );
} ).send( s );

SynthDef.new( \sawtoothTone, { arg freq=440, amp=0.1, pos=0;
	var src, env, out;
	src = LFSaw.ar( freq );
	env = EnvGen.kr(
			Env.new(
				[0, 1, 0.2, 0],
				[0.01, 0.2, 0.1]
				),
			levelScale: amp,
			doneAction: 2
			);
	out = Pan2.ar( src*env, 0 );
	Out.ar( 0, out );
} ).send( s );

SynthDef.new( \pulseTone, { arg freq=440, amp=0.1, pos=0;
	var src, env, out;
	src = LFPulse.ar( freq );
	env = EnvGen.kr(
			Env.new(
				[0, 1, 0.2, 0],
				[0.01, 0.2, 0.1]
				),
			levelScale: amp,
			doneAction: 2
```

```
                        );
        out = Pan2.ar( src*env, 0 );
        Out.ar( 0, out );
} ).send( s );
```

では、GUI の作成に移ろう。ウィンドウ上に設けるのは、ボタン３つとスライダー１つである。

```
var window, sine, saw, pulse, volume;

window = Window.new( "330Hz", Rect.new( 300, 300, 290, 90 ) );
window.view.decorator = FlowLayout.new( window.view.bounds );
sine = Button.new( window, 90@30 );
sine.states = [
                        [ \SINE, Color.black, Color.blue ],
                        [ \SINE, Color.red, Color.blue ]
                ];
sine.action = { arg butt;
                if ( butt.value==1, {
                        ~def = \sineTone;
                        saw.value = 0;
                        pulse.value = 0;
                        }, {                    // 同じボタンを２度クリックしても、
                        butt.value = 1;         // 状態を保つようにする。
                        }
                );
        };

saw = Button.new( window, 90@30 );
saw.states = [
                        [ \SAW, Color.black, Color.yellow ],
                        [ \SAW, Color.red, Color.yellow ]
                ];
saw.action = { arg butt;
                if ( butt.value==1, {
                        ~def = \sawtoothTone;
                        sine.value = 0;
                        pulse.value = 0;
                        }, {
                        butt.value = 1;
                        }
                );
        };

pulse = Button.new( window, 90@30 );
pulse.states = [
                        [ \PULSE, Color.black, Color.green ],
                        [ \PULSE, Color.red, Color.green ]
                ];
pulse.action = { arg butt;
                if ( butt.value==1, {
                        ~def = \pulseTone;
                        saw.value = 0;
                        sine.value = 0;
                        }, {
```

```
                        butt.value = 1;
                        }
                );
        };

window.view.decorator.nextLine;

volume = EZSlider.new(
                        window,
                        280@30,
                        \volume,
                        ControlSpec.new( 0.0, 1.0, \lin ),
                        { arg ez;
                                ~volume = ez.value;
                                },
                        0.1
                        );

window.view.keyDownAction = { arg view, char, modifiers, keycode, unicode;
                                // 引数のうち、char は文字を、
                                // keycode はその文字のキーに与えられた数値を表す
        if ( char==$g, {        // キーの文字は頭に$マークを付けて表す
                s.sendMsg( \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 330, \amp, ~volume );
        } );
};

window.front;           // ウィンドウを表示
~volume = 0.1;          // ~volume の値を初期化
sine.valueAction = 1;   // .valueAction は、ボタンの見た目の状態を変えるだけでなく、
                        // それに伴う関数も実行する
                        // これにより、~def の値は\sineTone に初期化される。
```

[問題 1]

任意の配列に対して、上の例では単純なサイン波、ノコギリ波、矩形波の音をそれぞれ 3 つのボタンに当てはめたが、代わりに、

•倍音の加算合成による 330Hz 音

•矩形波に周波数変調を施した 330Hz 音

•ノイズに減算合成（フィルター加工）を施して作った 330Hz 音

の 3 つを当てはめなさい。

"

条件分岐（2）

前回、条件分岐として if 構文を学んだが、今回は switch 構文を紹介しよう。

```
switch ( 値を判断するもの,
        値 A, { 値が A の場合の処理 },
        値 B, { 値が B の場合の処理 },
        ...
        );
```

これは、値の違いによって処理を２種類以上に分けたい場合に有効である。

例えば、次のように使うことができる。

```
switch ( rrand( 0, 3 ),
	0, { "Hello.".postln },
	1, { "Thank you.".postln },
	2, { "Excuse me.".postln },
	3, { "I'm sorry.".postln }
	);
```

rrand(0, 3)によって得られる数値は、0, 1, 2, 3 のいずれかである。そこで、その結果にしたがって、それぞれ表示する文字列を変えてみたのが、上の例である。

[問題 2]

任意の配列に対して、.choose メソッドを行なうと、中から 1 つの要素がランダムに選び出される。また、2 つの自然数 X と Y について、

X%Y

とすると、X を Y で割った余りが算出される。例えば、11%4 は 3 となり、14%7 は 0、5%7 は 5 となる。

そこで、

a = [6, 7, 8, 9, 10];

と

b = [2, 3, 5];

という 2 つの配列からそれぞれ要素を 1 つずつランダムに選び出し、a から選んだものを b から選んだもので割り、その余りの数値によって"Hello."、"Thank you."、"Excuse me."、"I'm sorry."のいずれかを区別して表示させるようなプログラムを記述しなさい。

キーボードを使う（2）

上の switch を使って、複数のキーにそれぞれ違った音を当てはめることができる。例えば、先の GUI の例で、window.view.keyDownAction = ... の部分を、

```
window.view.keyDownAction = { arg view, char, modifiers, keycode, unicode;
	switch ( char,
		$a, {
		s.sendMsg(
			\s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 57.midicps, \amp, ~volume
			);
		},
		$s, {
		s.sendMsg(
			\s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 59.midicps, \amp, ~volume
			);
		},
		$d, {
		s.sendMsg(
			\s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 61.midicps, \amp, ~volume
			);
		},
```

```
$f, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 63.midicps, \amp, ~volume
        );
},
$g, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 65.midicps, \amp, ~volume
        );
},
$h, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 67.midicps, \amp, ~volume
        );
},
$j, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 69.midicps, \amp, ~volume
        );
},
$k, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 71.midicps, \amp, ~volume
        );
},
$l, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 73.midicps, \amp, ~volume
        );
},
$q, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 56.midicps, \amp, ~volume
        );
},
$w, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 58.midicps, \amp, ~volume
        );
},
$e, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 60.midicps, \amp, ~volume
        );
},
$r, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 62.midicps, \amp, ~volume
        );
},
$t, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 64.midicps, \amp, ~volume
        );
},
$y, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 66.midicps, \amp, ~volume
        );
},
$u, {
s.sendMsg(
        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 68.midicps, \amp, ~volume
```

```
                        );
                },
                $i, {
                s.sendMsg(
                        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 70.midicps, \amp, ~volume
                        );
                },
                $o, {
                s.sendMsg(
                        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 72.midicps, \amp, ~volume
                        );
                },
                $p, {
                s.sendMsg(
                        \s_new, ~def, n=s.nextNodeID, 0, 1, \freq, 74.midicps, \amp, ~volume
                        );
                }
        );
};
```

という具合に変更すれば、ピアノの鍵盤のように、音の高さを変えることができる。

ただし、スライダーを動かした直後に、例えばＲのキーを押すと、Ｒに当てはめた音が出るかわりに、スライダーがランダムに動いてしまう。

これを防ぐために、ここではスライダーにフォーカスが当たらないように設定する。「フォーカスが当たらない」とは、スライダーとしての機能が損なわれることを意味しない。マウスを使ってスライダーを動かすことはできるし、それによって何らかの制御を行なうことは依然として可能だが、スライダーをキーボードで操作することのみができなくなる。

そのためには、

```
volume.sliderView.canFocus = false;
```

とする。

[問題 3]

上の例では、キーボード上のＱからＰまでの列とＡからＬまでの列のキーのみを扱っているが、さらに１から０までの列のキーとＺからＭまでの列のキーも加え、音の高さを適切に当てはめ、鍵盤を完成させなさい。

[問題 4]

[問題 3]で完成させた GUI に、さらにディレイのエフェクトを加えられるようにスイッチとなるボタンを付け、エコーの間隔とディレイの持続時間を調節するためのスライダーを計２本付け足しなさい。なお、SynthDef にも変更を加えること。