LIVECODE Community Edition

アニメエンジン日本語ノート

小島健治の LiveCodeアニメエンジン日本語ノート
http://kenjikojima.com/

このノートで書いているAnimationEngine語彙 一覧：

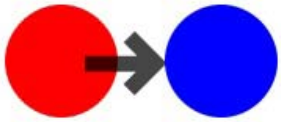
```
circleCollide
circleLineSegmentCollide
closestpointOnLine
constrainElliptical
    constrainEllipticalCallback
    constrainEllipticalExit
    constrainEllipticalInit
constrainLinear
    constrainLinearInit
    constrainLinearExit
    constrainLinearCallback
constrainRectangular
    constrainRectangularInit
    constrainRectangularExit
    constrainRectangularExit
    constrainRectangularCallback
distance
findAngle
imageCollide
intersectRect
moveCircular
moveElliptical
moveLinear
movePolygonal
pointInPoly
rotate3DPoint
rotateIsoPoint
spiral
ae3dconverttoscreen
```

このノートで説明している LiveCode語彙：

```
arcAngle
```

LIVECODE Community Edition

アニメエンジン日本語ノート



circleCollide (the loc of grc “blue”,the loc of grc “red”,threshold)

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/circleCollide.livecode"

「circleCollide」は、ふたつのオブジェクト（Oval) の 距離 が、5番目のパラメータ（threshold) 内にある時、true を返します。
「the loc of オブジェクトネーム」のパラメータは以下のように、 x と y とのふたつの数値でひとつの座標を表します。

基本構文：
circleCollide(正円１のロケーション, 正円２のロケーション, true を返す2点間の距離)
circleCollide(x1,y1,x2,y2,threshold)

AE / constrainRectangularCallback

「constrainRectangularCallback」は、
ハンドラー「mouseMove」と同じ働きをしますが、オブジェクト内に constrainRectangularの範囲を
カスタムプロパティで設定することで、その範囲内から外に出ません。

set the constrainRectangular of grc "blue" to 22,38,440,380
set the constrainRectangular of grc "red" to 22,38,440,380
「 constrainrectangular」は、カスタムプロパティにそのオブジェクトが、マウスダウンでドラックできる
「uAllowConstrainDrag」を作り、ドラッグできる the rect の範囲をカスタムプロパティ「 constrainrectangular」で設定します。
オブジェクト内には「grab me」は必要ありません。

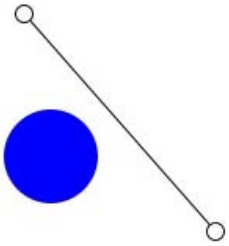
サンプルスクリプトは、直径70 pix の円が、2pix 以上重なった時、backgroundColor をグリーンに変えます。

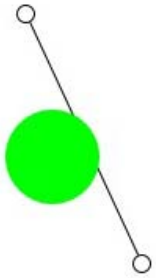
```
-- カード内のスクリプト
on constrainRectangularCallback
 if circleCollide(the loc of grc "blue",the loc of grc "red",68) then
          set the backgroundColor of grc "blue" to 0,255,0
          set the backgroundColor of grc "red" to 0,255,0
        else
          set the backgroundColor of grc "blue" to 0,0,255
          set the backgroundColor of grc "red" to 255,0,0
 end if
end constrainRectangularCallback
```

グラフィック内に、スクリプトは必要ありませんが、カスタムプロパティでconstrainRectangular の範囲を設定しています。

LIVECODE Community Edition

アニメエンジン日本語ノート





circleLineSegmentCollide

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/circleLineSegmentCollide.livecode"

直線と正円の弧が交差した時に「true」を返します。
サンプルスタックでは、交差した時に円の色がグリーンに変わります。

基本構文
circleLineSegmentCollide(直線の1行目のポイント(X1,Y1), 直線の2行目のポイント(X2,Y2), \
　　正円のロケーション, 正円の半径)
circleLineSegmentCollide(line 1 of the points of grc "tLine",line 2 of the points of grc "tLine", \
　　the loc of grc "blueOval", grc "blueOval"の半径)

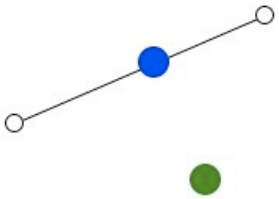
線のグラフィック「tLine」のポイントを求めると、
ラインの始めと終わりの、２行のポイントのロケーションが返されます。
put the points of grc “tLine”

パラメータ1 は。ラインの 1行目のポイント。パタメータ2 はラインの 2行目のポイント。
パラメータ3 は正円のロケーション。パラメータ4は正円の半径です。

```
-- 3つある円はすべて、マウスダウンで動かせるように以下を入れます
on mouseDown
 grab me
end mouseDown

-- 小円の名前は、グラフィック「point1」と、グラフィック「point2」としています
-- ブルーの正円の名前は、グラフィック「blueOval」


-- カード内に入れるスクリプトは
on mouseMove
 -- 白の小円が動かされた時、ラインのポイントが変わる
 set the points of grc "tLine" to the loc of grc "point1" &","& the loc of grc "point2"

 -- ラインのポイントの1行目は、グラフィック「point1」のロケーションに同じ
 -- ラインのポイントの2行目は、グラフィック「point2」のロケーションに同じ
 if circleLineSegmentCollide(the loc of grc "point1",the loc of grc "point2", \
   the loc of grc "blueOval",30) then
 -- true を返す
       set the backgroundColor of grc "blueOval" to green
   else
 -- false を返す
       set the backgroundColor of grc "blueOval" to blue
 end if
end mouseMove
```

LIVECODE Community Edition

アニメエンジン日本語ノート



closestpointOnLine

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/closestPointOnLine.livecode"

基本構文
closestpointOnLine(ラインのポイント１,ラインのポイント2,ドラッグするオブジェクトのロケーション)
closestpointOnLine(tLoc1,tLoc2,tLoc3)

ドラッグするオブジェクト（サンプルではグリーンの円）のロケーションのポイントから、
2点間の直線が常に直角になる線上のポイント（ロケーション）を返します。

直線「tLine」、直線の左右の円「point1」「point2」、直線上の円（ブルー）「pointIndicator」、
ドラッグで移動する円（グリーン）「dragCircle」とします。

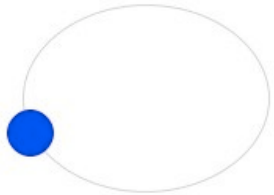
始めにメッセージボックスから「 constrainrectangular」を設定します。
set the constrainrectangular of grc "point1" to the rect of this card
set the constrainrectangular of grc "point2" to the rect of this card
set the constrainrectangular of grc "dragCircle" to the rect of this card

「 constrainrectangular」は、カスタムプロパティにそのオブジェクトが、マウスダウンでドラックできる
「uAllowConstrainDrag」を作り、ドラッグできるthe rect の範囲をカスタムプロパティ「 constrainrectangular」
で設定します。オブジェクト内には「grab me」は必要ありません。

```
-- 直線の両端のオブジェクトには
on constrainRectangularCallback
 put the loc of grc "point1" into tLoc1
 put the loc of grc "point2" into tLoc2
 put the loc of grc "dragCircle" into tLoc3
 set the points of grc "tLine" to tLoc1 & cr & tLoc2
 set the loc of grc "pointIndicator" to closestpointOnLine(tLoc1,tLoc2,tLoc3)
end constrainRectangularCallback


-- ドラッグするオブジェクトには
on constrainRectangularCallback
 put the loc of grc "point1" into tLoc1
 put the loc of grc "point2" into tLoc2
 put the loc of grc "dragCircle" into tLoc3
 set the loc of grc "pointIndicator" to closestpointOnLine(tLoc1,tLoc2,tLoc3)
end constrainRectangularCallback
```

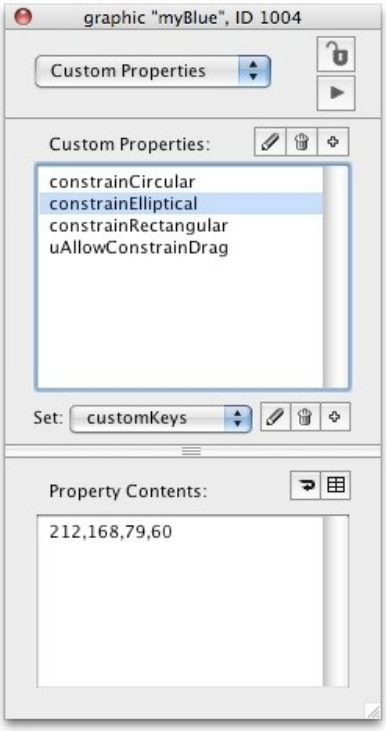
LIVECODE Community Edition

アニメエンジン日本語ノート



constrainElliptical

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/constrainElliptical.livecode"



楕円軌道をオブジェクトがマウスで動きます。ハンドラーの「on mouseMove」ではなく、
constrainElliptical でカスタムプロパティに設定された、 uAllowConstrainDrag　によってドラッグできます。
楕円軌道は同じく、constrainEllipticalで設定されるカスタクプロパティ constrainElliptical に、ロケーション、
楕円の左右の半径、天地の半径で決められます。

基本構文：
set the constrainElliptical of （マウスダウンで動くオブジェクト） to \
　　(楕円のロケーション x, y), (楕円の左右 / 2), (楕円の天地 / 2)

サンプル：
楕円軌道をグラフィック Oval で作って、名前は「myOrbit」。
マウスで動くオブジェクトをグラフィックで作って名前を「myBlue」としたら、メセージボックスから

set the constrainElliptical of grc "myBlue" to \
　　the loc of grc "myOrbit",the width of grc "myOrbit"/2,the height of grc "myOrbit"/2

grc “myBlue” には、右図の4つのカスタムプロパティが作られます。

一旦 constrainElliptical を設定したら、グラフィックの楕円軌道を削除しても、オブジェクトは楕円上を回ります。

ドキュメントによると、「constrainElliptical」を設定した場合、mouseDown, mouseUp, mouseRelease,
mouseMove 等を同時に使用すると問題が起こることがあるので代わりに

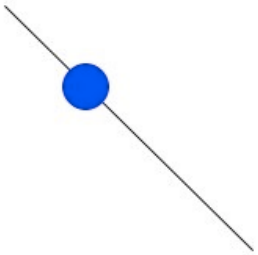
mouseDown - constrainEllipticalInit
mouseUp - constrainEllipticalExit
mouseRelease - constrainEllipticalExit
mouseMove - constrainEllipticalCallback

を使用するようにとあります。

ユーザーのマウスの移動ではなく、 正円軌道をオブジェクトが自動で回転するコマンド moveCircularと、
楕円上をオブジェクトが自動で回転するコマンド moveElliptical があります。

LIVECODE Community Edition

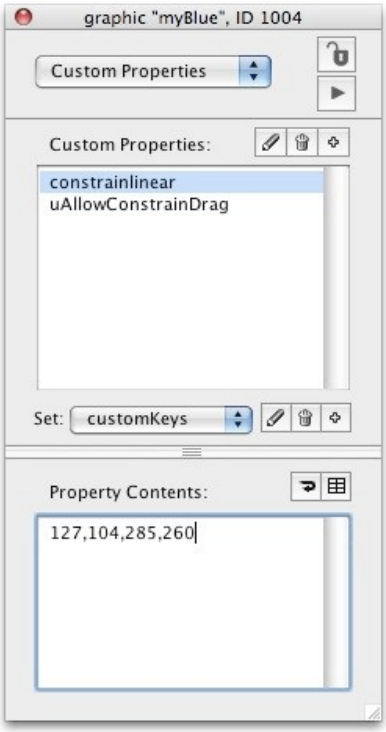
アニメエンジン日本語ノート



constrainLinear

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/constrainLinear.livecode"

直線軌道をオブジェクトがマウスで動きます。ハンドラーの「on mouseMove」ではなく、constrainLinear でカスタムプロパティに設定された、 uAllowConstrainDrag　によってドラッグできます。直線軌道は同じく、constrainLinear で設定されるカスタクプロパティ constrainLinear の ポイント１ と
ポイント2のロケーション（直線の両端）で決められます。



基本構文：
set the constrainLiner of （マウスダウンで動くオブジェクトの名前） to \
(グラフィック直線のポイント１ x1, y1), (グラフィック直線のポイント２ x2, y2)

サンプル：
グラフィック直線「tLine」、マウスで直線上を移動するグラフィック「myBlue」を作ったら、
メッセージボックスから
set the constrainlinear of grc "myBlue" to \
　　(line 1 of the points of grc "tLine" & "," & line 2 of the points of grc "tLine")

grc “myBlue” に、カスタムプロパティ「constrainLinear」と、「uAllowConstrainDrag」が作られ、
2点間のポイントが設定されます。

ドキュメントによると、「constrainLinear」を設定した場合、mouseDown, mouseUp, mouseRelease, mouseMove 等を同時に使用すると問題が起こることがあるので代わりに

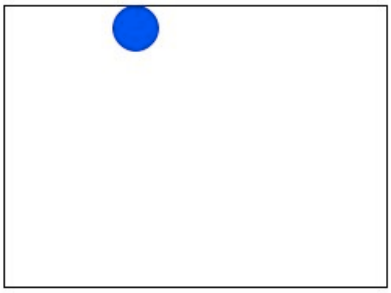
mouseDown - constrainLinearInit
mouseUp - constrainLinearExit
mouseRelease - constrainLinearExit
mouseMove - constrainLinearCallback

を使用するようにとあります。

同じように、直線上を自動でオブジェクトが移動するコマンドに、moveLinear があります。

LIVECODE Community Edition

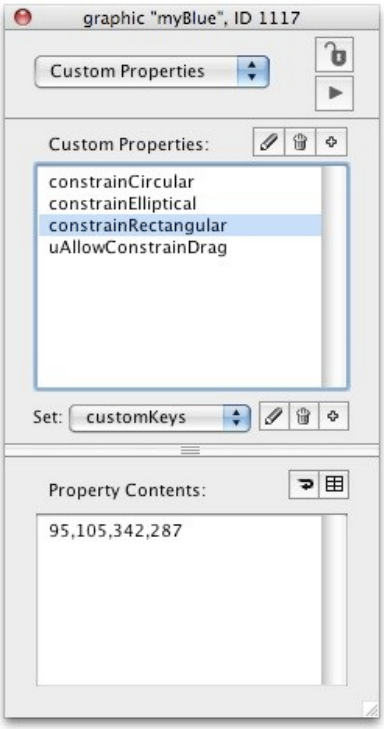
アニメエンジン日本語ノート



constrainRectangular

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/constrainRectangular.livecode"

設定した矩形の範囲内だけ、ドラッグでオブジェクトが移動で来ます。別な言い回しでは、
範囲外にオブジェクトは出る事ができません。範囲さえ設定すれば、矩形のグラフィックは必要ありません。



基本構文：
set the constrainRectangular of （グラフィック・オブジェクトの名前） to (矩形の4点のポイント、 X1, y1, x2, y2)

サンプル：
移動するオブジェクトのグラフィック「myBlue」と、移動範囲グラフィック矩形「tRect」 を作って、
メッセージボックスから

set the constrainRectangular of grc "myBlue" to rect of grc "tRect"

grc “myBlue” のカスタムプロパティに「contrainCircular」「constrainElliptical」「constrainRectangular」
「uAllowConstrainDrag」が作られて、「constrainRectangular」に移動できる矩形の範囲のポイントが設定されます。

ドキュメントによると、「constrainRectangular」を設定した場合、mouseDown, mouseUp, mouseRelease,
mouseMove を同時に使用すると、問題が起こることがあるので代わりに

mouseDown - constrainRectangularInit
mouseUp - constrainRectangularExit
mouseRelease - constrainRectangularExit
mouseMove - constrainRectangularCallback

を使うようにとあります。

具体的には、マウスダウンしたときに「myBlue」の色をグリーンに変更して、
マウスアップではまたブルーに戻すには

-- カード内に以下のスクリプトを入れます

```
on constrainRectangularInit　　-- mouseDown に代わって
 set the backgroundColor of grc "myBlue" to green
end constrainRectangularInit

on constrainRectangularExit　　-- mouseUp に代わって
 set the backgroundColor of grc "myBlue" to blue
end constrainRectangularExit
```

LIVECODE Community Edition

アニメエンジン日本語ノート

blueRed:	68
blueGreen:	85
GreenRed:	73

distance(the loc of grc "blue",the loc of grc "Red")

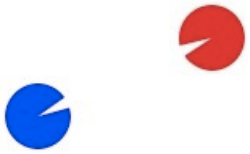
始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/distance.livecode"

ふたつのオブジェクトのロケーションの距離を、ピクセルで返します。

```
-- カード内のスクリプト
on mouseMove
 put distance(the loc of grc "blue",the loc of grc "red") into fld "blueRed"
 put distance(the loc of grc "blue",the loc of grc "green") into fld "bluegreen"
 put distance(the loc of grc "green",the loc of grc "red") into fld "greenRed"
 pass mouseMove
end mouseMove

-- それぞれのグラフィック内のスクリプト
on mouseDown
 grab me
end mouseDown
```

LIVECODE Community Edition

アニメエンジン日本語ノート



findAngle(the loc of grc "blue",the loc of grc "Red")

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
```
go stack \
    url "http://kenjikojima.com/livecode/ae/findAngle.livecode"
```

ふたつのオブジェクトのロケーションのアングルを返します。
ふたつのオブジェクトが上下にある場合： 360, 80 または、 180, 0
ふたつのオブジェクトが左右にある場合： 90, 270 または、 270, 90

例では、arcAngle = 340 の、ブルーとレッドのグラフィック「oval」を、
fineAngle によって startAngle を変化させて、互いに向かい合っているようにしています。

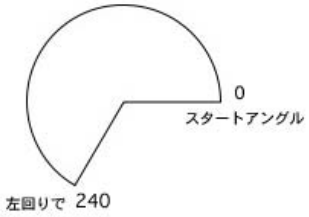
```
-- カード内のスクリプト
on openCard
 set the arcAngle of grc "blue" to 340
 set the arcAngle of grc "red" to 340
end openCard


-- arcAngle でグラフィックの開いた部分は、 startAngle が起点となっているので
-- アングルの中間で互いが向かい合うように調整する
function adjustAngle
 return (360 - the arcAngle of grc "blue") div 2
end adjustAngle


command moveAngle
 put findangle(the loc of grc "blue",the loc of grc "Red") into test1
 put findangle(the loc of grc "Red",the loc of grc "blue") into test2
 if test1=0 then put 360 into test1
 if test2=0 then put 360 into test2
 set the startAngle of grc "blue" to 90 -test1 +adjustAngle()
 set the startAngle of grc "Red" to 90 -test2 +adjustAngle()
end moveAngle


-- それぞれのグラフィック内のスクリプト
on mouseMove
 moveAngle
 pass mouseMove
end mouseMove


on mouseDown
 grab me
end mouseDown
```

LiveCode の arcAngle (楕円の開いた角度を設定) の言語説明

LIVECODE Community Edition

アニメエンジン日本語ノート

arcAngle



種類：LiveCode の arcAngle (楕円の開いた角度を設定) の言語説明
property (graphic オブジェクトのプロパティ)

基本的な構文：
set the arcAngle [of graphic] to angleInDegrees

arcAngle は、グラフィックオブジェクトのプロパティのひとつで、楕円の開いた角度を設定します。

「arcAngle」はグラフィック・オブジェクトで、チャートなどで使う楕円の開いた角度を作成する時に設定します。
円がまったく開いてない状態では「360」です。ツールパレットの Oval ツールで楕円を描くと、
インスペクターに「startAngle」と「arcAngle」をコントロールするスライダーが現れて、値をコントールできます。

グラフィック・オブジェクトが作られた状態（デフォルト）では、「arcAngle」は 360 で、「startAngle」は 0 です。
「arcAngle」の数値は 0 から 360 の間の整数が設定できます。

スクリプト例：
-- グラフィック1を作って、 arcAngle を240に設定
create graphic -- グラフィックの基本形を作成
set the style of grc 1 to oval -- グラフィックのタイプ（スタイル）を oval に
set the height of grc 1 to the width of grc 1 -- オーバルを正円に
set the arcAngle of grc 1 to 240 -- arcAngle を240に設定

-- 上のスクリプトで作られた「arcAngle を240に設定」したグラフィックは、
-- 「startAngle」が 0 ですから、時計の分針で言えば、 15分から35分の間が空いた円になります。

set the startAngle of grc 1 to 90
--さらに startAngle を90に設定すると、0分から15分（4分の 1 の円）になります。

LIVECODE Community Edition

アニメエンジン日本語ノート

imageCollide

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/imageCollide.livecode"

PING イメージなど、透明部分のある（アルファチャンネル）イメージの、
不透明な部分が重なった時に「true」を返します。
ファンクションが有効な最大イメージサイズは 80 x 80。

PING イメージ

false

基本構文
imageCollide(イメージ名１, イメージ名２, しきい値ピクセル数)
imageCollide(the name of image 1, the name of image 2, threshold)

パラメータ 「threshold」日本語にすると限界点とか、しきい値という言葉になりますが、
イメージの半透明な影とか滲みの部分が重なった時にでも、「true」を返えさない範囲が設定できます。
別な言葉で言えば、実際のイメージの実質的な部分の衝突で、「true」を返す設定にできます。
どういう数値にすればその範囲が定められるのか、サンプル作成時点では不明でしたが、
とにかく設定すれば機能するようです。ドキュメントによると、「threshold」の設定は 0 から 255 となっています。

true

サンプル2 のイメージサイズは、50x50 ピクセル。 完全に不透明な円の直径は 26 ピクセルなので、
「threshold」 を 12 としてみた処、範囲が遠過ぎてあまり効果が見えませんでした。少しづつ数値を上げて、
最終的に 90 にしたところ完全な不透明部分の衝突で、trueが返されました。

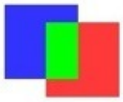
false
パラメータ threshhold 設定

```
-- カード内のスクリプト(サンプル2)
on mouseMove
 if imageCollide(the name of img 1,the name of img 2,90) then
        put "Collide!" into fld "collide"
 else
        put "" into fld "collide"
 end if
end mouseMove
```

true
パラメータ threshhold 設定

LIVECODE Community Edition

アニメエンジン日本語ノート



The rect of grc "green" is 143,201,165,239

intersectRect(the rect of grc "blue",the rect of grc "red")

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/intersectRect.livecode"

ふたつの矩形のオブジェクトの重なった the rect を返します。
サンプルでは、重なった部分に、グリーンのグラフィックが現れます。

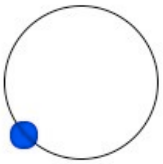
基本構文：
intersectrect(矩形 1 の the rect , 矩形2 の the rect)
intersectrect(the rect of grc "blue", the rect of grc "red")

```
-- カード内のスクリプト
command setIntersect
 get intersectrect(the rect of grc "blue",the rect of grc "red")
 if it <> false then
        set the rect of grc "green" to it
        show grc "green"
        put it into fld "intersect"
 else
        hide grc "green"
        put empty into fld "intersect"
 end if
end setIntersect
```

```
-- それぞれのグラフィック内のスクリプト
on mouseMove
 setIntersect
 pass mouseMove
end mouseMove

on mouseDown
 grab me
end mouseDown
```

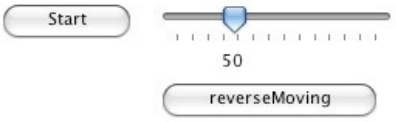
LIVECODE Community Edition

アニメエンジン日本語ノート



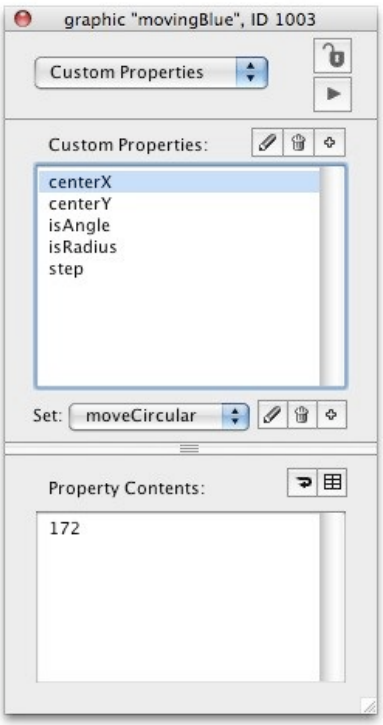
moveCircular

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/moveCircular.livecode"



正円の軌跡で移動するオブジェクトのカスタム・キー「moveCircular」を作って、必要なプロパティを設定します。
これは正円上の軌跡で移動と言う事で、必ずしもグラフィックの円を描かなくても、
円のセンターと半径を設定すれば動くようです。

-- グラフィック「movingBlue」を作ったら、メッセージボックスから
send moveCircular to grc "movingBlue"



を送ると、カスタムキー「moveCircular」が grc "movingBlue" に作られて、
プロパティのデフォルト値が以下のように書き込まれます。カッコ内はデフォルトの値
centerX　　軌跡になる円のセンターX (grc "movingBlue"のあった X)
centerY　　軌跡になる円のセンターY (grc "movingBlue"のあった Y)
isAngle　　軌跡になる円の arcAngle (1)
isRadius　　軌跡になる円の半径 (50)
step　　移動するオブジェクトのスピード (1)

移動する軌跡がわかりやすいように、上の値を元に円を作ります。
width, height ともに 100 ピクセルの円（半径 50）、ロケーションは grc "movingBlue"のあったX,Y で
円を作ると（サンプルは 200,160)、 grc "movingBlue"は 1 の位置に移動します。
これで grc "movingBlue" に、moveCircular を送るとこの円上の軌跡で移動します。

-- ボタン「startStop」を作って、円上で移動するスクリプトを書き込みます。
-- movePoligonal のスタートストップとまったく同じです。
```
on mouseUp
 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me
 if the flag of me then hereWeGo
end mouseUp

on hereWeGo
 send moveCircular to grc "movingBlue"
 if the flag of me then send hereWeGo to me in 20 milliseconds
end hereWeGo
```

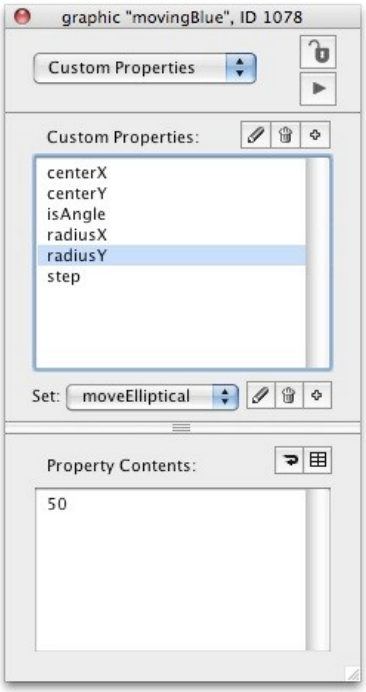
これでボタンをクリックすると、step 1 のスピードで時計回りにオブジェクトが移動します。

-- スピードを上げるには “step” の数を増やします
set the moveCircular["step"] of grc "movingBlue" to 3

-- 反対回り（左回り）にするには、“step” の数をマイナスにします
set the moveCircular["step"] of grc "movingBlue" to -4

LIVECODE Community Edition

アニメエンジン日本語ノート



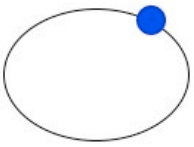
moveElliptical

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/moveElliptical.livecode"

楕円の軌跡で移動するオブジェクトのカスタム・キー「moveElliptical」を作って、必要なプロパティを設定します。
これは楕円上の軌跡で移動と言う事で、必ずしもグラフィックの楕円を描かなくても、
楕円のロケーションと半径を設定すれば動くようです。

-- グラフィック「movingBlue」を作ったら、メッセージボックスから
send moveElliptical to grc "movingBlue"
を送ると、カスタムキー「 moveElliptical」が grc "movingBlue" に作られて、
プロパティのデフォルト値が書き込まれます。

カッコ内はデフォルトの値
centerX　　軌跡になる楕円のセンターX (grc "movingBlue"のあった X)
centerY　　軌跡になる楕円のセンターY (grc "movingBlue"のあった Y)
isAngle　　軌跡になる楕円の arcAngle (0)
radiusX　　軌跡になる楕円のヨコ半径 (80)
radiusY　　軌跡になる楕円のタテ半径 (50)
step　　移動するオブジェクトのスピード (1)



移動する軌跡がわかりやすいように、上の値を元に楕円を作ります。
width = 160（半径 80）, height = 100（半径 50）、
ロケーションは grc "movingBlue"のあった X,Y (サンプルでは200,160) で楕円を作ると、
grc "movingBlue"はarcAngle 0 の位置に動きます。
これで grc "movingBlue" に、moveElliptical を送るとこの円上の軌跡で移動します。

-- ボタン「startStop」を作って、楕円上で移動するスクリプトを書き込みます。
-- movePoligonal、moveCircle のスタートストップとまったく同じです。

```
on mouseUp
 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me
 if the flag of me then hereWeGo
end mouseUp

on hereWeGo
 send moveElliptical to grc "movingBlue"
 if the flag of me then send hereWeGo to me in 20 milliseconds
end hereWeGo
```

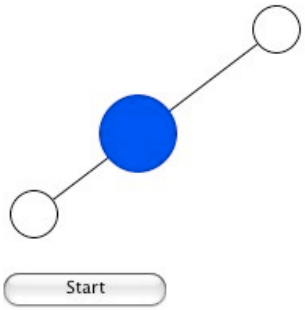
これでボタンをクリックすると、step 1 のスピードで時計回りにオブジェクトが移動します。

-- スピードを上げるには “step” の数を増やします
set the moveElliptical["step"] of grc "movingBlue" to 3

-- 反対回り（左回り）にするには、“step” の数をマイナスにします
set the moveElliptical["step"] of grc "movingBlue" to -3

LIVECODE Community Edition

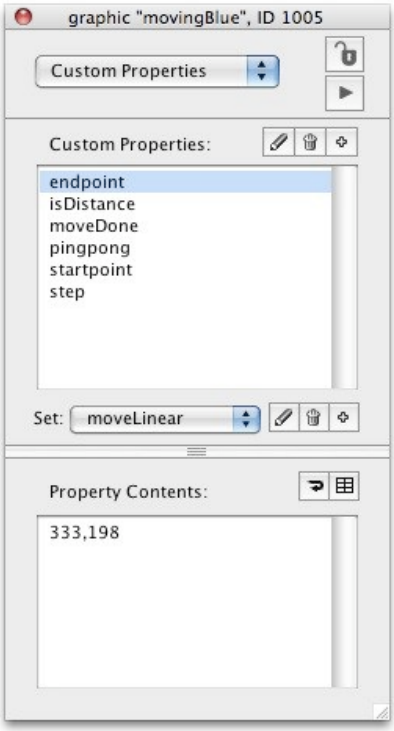
アニメエンジン日本語ノート



moveLinear

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/moveLinear.livecode"

オブジェクトが直線で移動するプロパティを、ターゲットとなるオブジェクトのカスタム・キーに、
「**moveLinear**」と言う名前で設定します。サンプルでは向きと長さの違う直線を作り出す為に、
左右にあるグラフィックの白い円をドラッグすることで、直線の向きと長さのが変化できるようになっています。
移動するポイントの、2点のX値Y値を設定すれば、 グラフィックの直線は描く必要はありません。




```
-- 直線の長さ角度を変化させるスクリプト：カード内
-- 白い Oval のグラフィックは、「 startOval」と「 endOval」と言う名前にします
on mouseMove
 set the points of grc "tLine" to the loc of grc "startOval" &","& the loc of grc "endOval"
end mouseMove
```

右図のブルーの円が、直線上を移動するターゲットで、名前を「movingBlue」としました。
カスタムキー「 **moveLinear**」にセットするプロパティは、図２にある通り６種類がセットされます。

テストのために、「**startpoint**」と「**endpoint**」だけを設定してみます。

```
-- ブルーの円(グラフィック）を、ロケーション 100,100 と 200,100 の間を行き来させる
set the moveLinear["startpoint"] of grc "movingBlue" to 100,100
set the moveLinear["endpoint"] of grc "movingBlue" to 200,100
```

これで grc “movingBlue” に、カスタムキーの「 **moveLinear**」が作られて、
「**startpoint**」と「**endpoint**」にそれぞれの座標がセットされます。

次に X軸の100 と200 との間を行き来させるボタン「StopStart」を作って、
そのボタンの「flag」と言うカスタムプロパティが true なら、ブルーの円が移動、
false なら止まるように、ボタン内にスクリプトを書きますが、その前に

```
-- クリックの度に、カスタムプロパティ「flag」は、「true」と「false」が交互にセットされる、
-- トリガーのスクリプトが必要です。
if the flag of me is empty then set the flag of me to false
set the flag of me to not the flag of me
```

```
-- この後、true ならば(= the flag of me) 移動する条件を書いて、hereWeGo と言うハンドラーに渡します。
if the flag of me then hereWeGo
-- もし the flag of me が true でない場合は、 mouseUp を抜けて grc "movingBlue"はストップします。
```

```
-- hereWeGo では、the flag of me がtrue である間
--「 moveLinear」を 20 ミリセカンド毎に grc "movingBlue" 送り続けます
on hereWeGo
 send moveLinear to grc "movingBlue"
 if the flag of me then send hereWeGo to me in 20 milliseconds
end hereWeGo
```

「 **moveLinear**」を grc "movingBlue" 送ると、カスタムキー「 **moveLinear**」には、
「**startpoint**」と「**endpoint**」以外のカスタムプロパティ「**isDistance**」、
「**moveDone**」「**step**」がセットされます。

isDistance は、 「**startpoint**」からターゲットの移動しているオブジェクトまでのピクセル数**moveDone** は、
true であればストップ。そうでない場合は移動中**step** は、直線上を移動するオブジェクトのスピード。デフォルトでは １ 。
数値を上げると高速になります

もうひとつセットできるプロパティの「**pingpong**」は、「true」であれば移動するオブジェクトが折り返されます。

ブルーの円(グラフィック）を、ロケーション 100,100 と 200,100 の間を行き来させる、

ボタンのスクリプトを通しで書いてみます。実際には、ボタンのレイベルが「Stop」と「Start」とに切り替わるスクリプト等が必要ですが、テスト用にシンプルにしています。

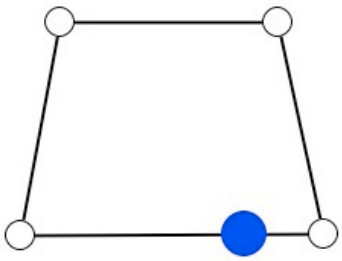
```
-- button “StartStop”のスクリプト
on mouseUp
 if the flag of me is empty then set the flag of me to false
        set the flag of me to not the flag of me
 if the flag of me then
        hereWeGo
 end if
end mouseUp

on hereWeGo
 send moveLinear to grc "movingBlue"
 if the flag of me then send hereWeGo to me in 20 milliseconds
end hereWeGo
```

これで基本的な動きはわかりましたが、まだ変化する直線上ではオブジェクトが動いていません。
その前に「**step**」のスピードを 3 に、「**pingpong**」を true にしておきます。

```
set the moveLinear["step"] of grc "movingBlue" to 3 set the moveLinear["pinpong"] of grc "movingBlue" to true

-- カード内に書き込むスクリプト 1on mouseMove
-- 直線を変化させるスクリプト
-- 上記テストでは固定した数値でしたが、直線が変更されるにしたがって、スタートとエンドが書き替えられます
 set the points of grc "tLine" to the loc of grc "startOval" &","& the loc of grc "endOval"

 -- 移動するターゲットオブジェクト内に、カスタムキー「moveLinear」の「startpoint」と「endpoint」がセットされる
 set the moveLinear["startpoint"] of grc "movingBlue" to  \
        the loc of grc "startOval" set the moveLinear["endPoint"] of grc "movingBlue" to the loc of grc "endOval"
end mouseMove


-- カード内に書き込むスクリプト 2
-- カード上にあるオブジェクトが共通に使えるコマンド。
-- このサンプルでは、button “StartStop”, grc “startOval”, grc “endOval”で使っています
-- 上のベーシックテストで、 ボタン “StartStop”に書き込んだのと、ほとんど同じです
command hereWeGo
 send moveLinear to grc "movingBlue"
 -- 各オブジェクトに共通に使えるよう、 ボタン“StartStop” では me としていたのを「the target」にしています
 if the flag of the target then send hereWeGo to the target in 20 milliseconds
end hereWeGo

-- ボタン「startEnd」に書き込むスクリプトon mouseUp -- the flag of me のプロパティを true と false に交互に切り替える
 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me
 if the flag of me then
        -- the flag of me が  true ならば、コマンド hereWeGo に渡す
         hereWeGo -- ボタンのレイベルを「Stop」にする
         set the label of me to "Stop"
 else
        -- false の場合、オブジェクトの移動はストップされるので -- ボタンのレイベルを「Start」にする
         set the label of me to "Start"
 end if
end mouseUp


-- グラフィック「startOval」と「endOval」に、共通に書き込むスクリプト
on mouseDown
 set the flag of me to true
 hereWeGo
 grab me
end mouseDown

on mouseUp
 set the flag of me to false
end mouseUp
```

LIVECODE Community Edition

アニメエンジン日本語ノート

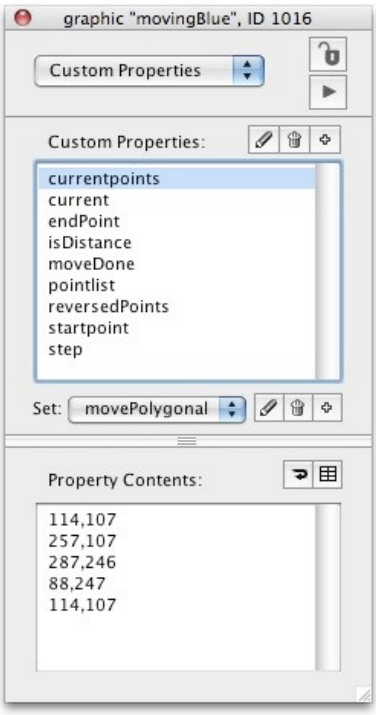


movePolygonal

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/movePolygonal.livecode"

多角形で直線移動するオブジェクトのカスタム・キー「movePolygonal」を作って、必要なプロパティを設定します。
これは多角形で移動すると言う事で、必ずしもグラフィックの線を描かなくても、
ポイントさえ指示すれば、そのように動くようです。
サンプルは白い丸をドラッグすれば、多角形が変形して移動する直線がわかりやすくなっています。

多角形を変形させるスクリプト（サンプルスタックの為で、movePolygonal とは直接関係ありません）
-- 「myPath」と言う名前のポリゴンのポイントを得るスクリプトは
put the points of grc “myPath”
をメッセージボックスで叩くと、以下のような５行のポイントが得られます。



114,107
257,107
287,246
88,247
114,107

１行目と５行目の数値は同じですから、この多角形は始めと最後のポイントが閉じられています。
それぞれのポイントに白い円をグラフックで作ります。
左上「ovalTL」、右上「ovalTR」、右下「ovalBR」、左下「ovalBL」

次に移動するブルーの円 (“movingBlue”)のグラフィックを作って、「movePoligonal」の “pointlist” を、
メッセージボックスから予めセットしておきます。

```
set the movePolygonal["pointlist"] of grc "movingBlue" to the points of grc "myPath"
```

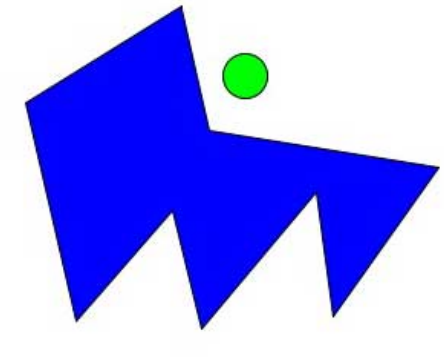
右図のインスペクターの、“pointlist”だけがセットされます。上記のポイントリストになっているはずです。

```
-- ポイントを変化させるスクリプト (ポイント４カ所に同じものを書き込みます)
-- setPoligon のパラメタがポイントによって変わってきます
on mouseDown
 grab me
end mouseDown

on mouseMove
 -- 右上グラフィック Oval “ovalTR” の
 -- ポリゴンのポイント 2（右上） が移動した時に、ポリゴンが変化する。パラメータは 2
 setPoligon 2
end mouseMove

on mouseUp
 -- ポイントリストを最新のデータに書き替える（上記のメッセージボックスからと同じ）
 setTpoints
end mouseUp


-- カード内に書き込む ポリゴンのポイントが移動した時のコマンド
-- パラメータ pline はポイントの順位。pline2 は始めと最後のポイント
command setPoligon pline, pline2
 put the loc of the target into tLoc
 get the points of grc "myPath"
 put tLoc into line pline of it
 if pline2 <> empty then put tLoc into line pline2 of it
 set the points of grc "myPath" to it
 setTpoints
end setPoligon
```

```
 -- ポイントリストを最新のデータに書き替える（上記のメッセージボックスからと同じ）
command setTpoints
 set the movePolygonal["pointlist"] of grc "movingBlue" to the points of grc "myPath"
end setTpoints


-- スタート、ストップのボタンを作ります。右上の図では左下のボタン
on mouseUp
 -- カスタムプロパティに flag を作って、true, false でスタート、ストップを指示します
 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me
 -- the flag of me がtrue だったら、 hereWeGo に渡す
 if the flag of me then hereWeGo
end mouseUp

on hereWeGo
       -- movePolygonal を移動するオブジェクトに送ると、図のプロパティがセットされます
 send movePolygonal to grc "movingBlue"
 if the flag of me then send hereWeGo to me in 20 milliseconds
end hereWeGo


currentPoints  移動中のポリゴンのポイントリスト
current  現在移動中のポイントの endPoint の順位
startPoint    2点間の始めのポイント x,y
endPoint     2点間の終わりのポイント x,y
isDistance     移動中の2点間の始めのポイントから、現在のポイントまでピクセル
moveDone       移動が終われば true
pointList       ポリゴンのポイントリスト
reversedPoints     反対回りのポイントリスト


-- 反対向きに方向を変えるボタン（reverseMoving)
on mouseUp
 send movePolygonalReversePath to grc "movingBlue"
end mouseUp


これでベーシックな動きに関しては一通りですが、サンプルではブルーの円がストップしている時に、
ポイントを動かすとそれに従って、２点間の直線上に移動するように、書き替えています。
```

LIVECODE Community Edition

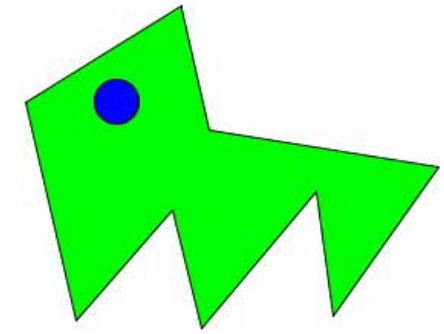
アニメエンジン日本語ノート



pointInPoly

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/pointInPoly.livecode"

基本構文：
pointInPoly(移動するオブジェクトのロケーション, 多角形のポイントリスト)
pointInPoly(the loc of grc "pipBut", the points of grc "pipPoly")



多角形（poligon) の不透明部分に、
他のオブジェクトのロケーション（センター部分）が入った時に、true を返します。

ポリゴンの名前「pipPoly」、円の名前「"pipBut」にしてメッセージボックスから
set the constrainRectangular of grc "pipPoly" to the rect of this card
set the constrainRectangular of grc "pipBut" to the rect of this card

constrainRectangular は、オブジェクトが移動できる範囲を設定して、
オブジェクト内の on mouseMove に grab me が書かれているのと、同じような働きをします。
constrainRectangular が設定してあれば、オブジェクト内に grab me は必要ありません。

```
-- カード内に入れるスクリプト。 true になった時に色を変える
on constrainRectangularCallback
 if pointInPoly(the loc of grc "pipBut", the points of grc "pipPoly") then
        set the backgroundColor of grc "pipBut" to blue
        set the backgroundColor of grc "pipPoly" to green
 else
        set the backgroundColor of grc "pipBut" to green
         set the backgroundColor of grc "pipPoly" to blue
 end if
end constrainRectangularCallback
```

LIVECODE Community Edition

アニメエンジン日本語ノート

rotateIsoPoint

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/rotateIsoPoint.livecode"

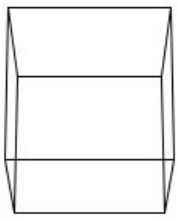
rotate3DPointとほぼ同じ機能ですが、パースペクティブのないアイソメトリック(isometric)図法で表現された、ポリゴンのポイントを返します。図参照。
基本形体の XYZ座標は、右の表のようにae3dConvertToScreenで変換した立体を描くポリゴンのグループです。
基本形体に回転のアングルをパラメータで与えて、それぞれのポイントを得ます。

基本構文：
rotateIsoPoint(基本形体のX座標, 基本形体のY座標, 基本形体のZ座標, \
 Xの回転アングル,Yの回転アングル,Zの回転アングル)

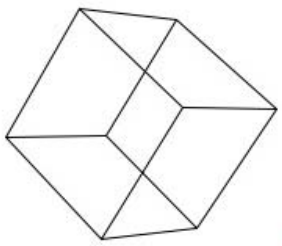
rotateIsoPoint(x,y,z,XRotation,Yrotation,Zrotation)

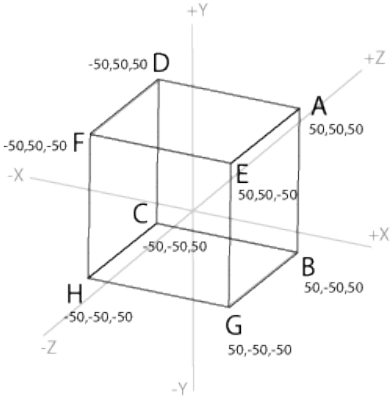
```
50,50,50
50,-50,50
-50,-50,50
-50,50,50
50,50,50
50,50,-50
-50,50,-50
-50,50,50

50,-50,50
50,-50,-50
50,50,-50

-50,50,-50
-50,-50,-50
50,-50,-50

-50,-50,-50
-50,-50,50
```

サンプル：
X軸だけの回転、 Y軸だけの回転、 Z軸だけの回転、その他複合の回転ができるように
rX, rY, rZ のチェックボックスを作ります。-- まず、ボタン「tStart」に、
カスタムプロパティにフィールド「tPoints」にある、
-- 基本形体の XYZ座標をセットします。
 set the cPoints of btn "tStart" to field "tPoints"




```
-- ボタン「スタート」のスクリプト-- ファンクション名が「 rotateIsoPoint」 に変わるだけで、
rotate3DPoint と同じです。
local lPoints,lAngleX,lAngleY,lAngleZ

on mouseUp
 put the cPoints of me into lPoints
 -- カスタムプロパティのcPoints を基本形体のポイントリストにする

 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me

 if the flag of me then
        -- XYZ の回転アングルを 0 に
        put 0 into lAngleX
         put 0 into lAngleY
         put 0 into lAngleZ
         set the label of me to "stop"
         rotateMe
 else
         set the label of me to "start"
 end if
end mouseUp

-- send rotate to me があるので、 Rev 3.0 では private command は使えません
on rotateMe
 lock screen
 -- チェックボックスがハイライトになっていたら、アングルを 1 づつ加える
 if the hilite of btn "rX" then
         put lAngleX+1 into lAngleX
 end if
 if the hilite of btn "rY" then
         put lAngleY+1 into lAngleY
 end if
 if the hilite of btn "rZ" then
```





```
        put lAngleZ+1 into lAngleZ
    end if

    -- 基本形体のポイント数だけリピート
    repeat for each line theLine in lPoints
        if theLine is not empty then
        -- 回転アングルの数値の 3D ポイントを得て theLine2に入れる
            put rotateIsoPoint(item 1 of theLine,item 2 of theLine,item 3 of theLine,\
            lAngleX,lAngleY,lAngleZ) into theLine2

        -- 上の 3D ポイントを平面上のポイントに変換する
            put ae3DConvertToScreen(item 1 of theLine2,item 2 of theLine2,item 3 of theLine2,\
        the loc of this cd,800)&cr after screenPoints
        else
        -- ひと筆描きのポリゴンのブレーク
            put cr&cr after screenPoints
        end if
    end repeat

    -- 得られた screenPointsでポリゴンを描く
    set the points of grc "threeD" to screenPoints
    wait 0 milliseconds with messages
    unlock screen

    if the flag of me then send rotateMe to me in 5 milliseconds
end rotateMe
```

LIVECODE Community Edition

アニメエンジン日本語ノート

rotate3DPoint

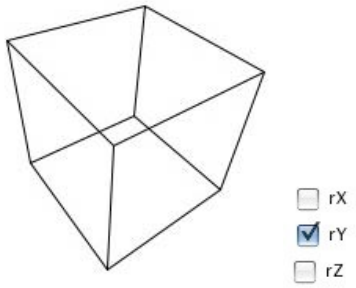
始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/rotate3DPoint.livecode"

3D 座標の立体が、平面上で回転しているようにポリゴンを描く値を返します。図参照。
基本形体の XYZ座標は、右の表のようにae3dConvertToScreenで変換した立体を描くポリゴンのグループです。
基本形体に回転のアングルをパラメータで与えて、それぞれのポイントを得ます。

```
50,50,50
50,-50,50
-50,-50,50
-50,50,50
50,50,50
50,50,-50
-50,50,-50
-50,50,50

50,-50,50
50,-50,-50
50,50,-50

-50,50,-50
-50,-50,-50
50,-50,-50

-50,-50,-50
-50,-50,50
```

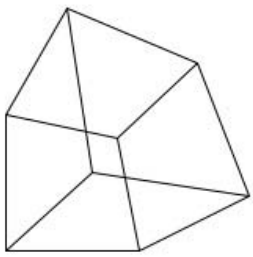
基本構文：

```
rotate3DPoint(基本形体のX座標, 基本形体のY座標, 基本形体のZ座標, \
 Xの回転アングル,Yの回転アングル,Zの回転アングル[, 焦点距離])

rotate3DPoint(x,y,z,XRotation,Yrotation,Zrotation[,pFocalLength])
```

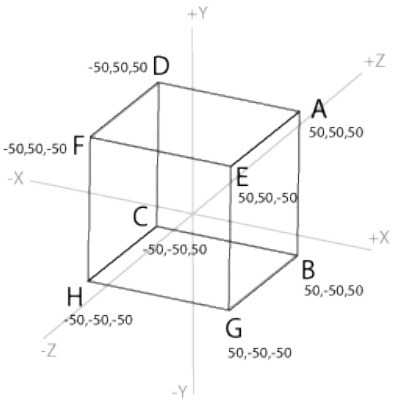


サンプル：
X軸だけの回転、 Y軸だけの回転、 Z軸だけの回転、その他複合の回転ができるように
rX, rY, rZ のチェックボックスを作ります。

```
-- まず、ボタン「tStart」に、カスタムプロパティにフィールド「tPoints」にある、
-- 基本形体の XYZ座標をセットします。
 set the cPoints of btn "tStart" to field "tPoints"
```



```
-- ボタン「スタート」のスクリプト
local lPoints,lAngleX,lAngleY,lAngleZ

on mouseUp
 put the cPoints of me into lPoints
 -- カスタムプロパティのcPoints を基本形体のポイントリストにする

 if the flag of me is empty then set the flag of me to false
 set the flag of me to not the flag of me

 if the flag of me then
 -- XYZ の回転アングルを 0 に
        put 0 into lAngleX
        put 0 into lAngleY
        put 0 into lAngleZ
        set the label of me to "stop"
        rotateMe
 else
        set the label of me to "start"
 end if
end mouseUp

-- send rotate to me があるので、 Rev 3.0 では private command は使えません
on rotateMe
 lock screen
 -- チェックボックスがハイライトになっていたら、アングルを 1 づつ加える
 if the hilite of btn "rX" then
        put lAngleX+1 into lAngleX
 end if
 if the hilite of btn "rY" then
        put lAngleY+1 into lAngleY
```



```
 end if
 if the hilite of btn "rZ" then
        put lAngleZ+1 into lAngleZ
 end if

 -- 基本形体のポイント数だけリピート
 repeat for each line theLine in lPoints
        if theLine is not empty then
         -- 回転アングルの数値の 3D ポイントを得て theLine2に入れる
              put rotate3DPoint(item 1 of theLine,item 2 of theLine,item 3 of theLine,\
              lAngleX,lAngleY,lAngleZ) into theLine2

        -- 上の 3D ポイントを平面上のポイントに変換する
              put ae3DConvertToScreen(item 1 of theLine2,item 2 of theLine2,item 3 of theLine2,\
              the loc of this cd,800)&cr after screenPoints
        else
         -- ひと筆描きのポリゴンのブレーク
         put cr&cr after screenPoints
        end if
 end repeat

 -- 得られた screenPointsでポリゴンを描く
 set the points of grc "threeD" to screenPoints
 wait 0 milliseconds with messages
 unlock screen

 if the flag of me then send rotateMe to me in 5 milliseconds
end rotateMe
```

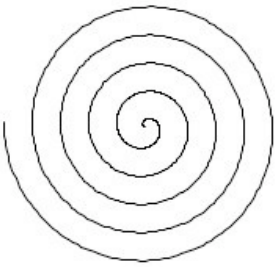
LIVECODE Community Edition

アニメエンジン日本語ノート

spiral

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/spiral.livecode"

グラフィクを渦巻き状のポイントを作る、コマンドです。グラフィックであれば、
直線でも円でも、プロパティ(style) を Free Hand Curve にして、points を渦巻き状に設定します。



基本構文：

set the spiral of grc “グラフィック名” to \
 グラフィックのロケーションX, グラフィックのロケーションY, 間隔のサイズと向き, 回数, 精度

set the spiral of grc “graphic name” to (cx,cy,step,windings(,accuracy))

サンプルスクリプト：

 set the spiral of grc 1 to 200,150,0.03,3,draft
 set the spiral of grc 1 to 200,150,0.04,4,low
 set the spiral of grc 1 to 200,150,-0.05,5,high
 set the spiral of grc 1 to 200,150,-0.06,6,best

プロパティ：

step: 間隔のサイズと向き -1 から 1までの間。 数値が大きくなると間隔が開いて、
向きはプラスで外側から右回り、マイナスで外側から左回り
windings： 渦を巻く回数
accuracy: グラフィックの精度（滑らかさ）は、数値で 1 から 36 まで、
　　または語の best、high、low、draft が使えます。

LIVECODE Community Edition

アニメエンジン日本語ノート

ae3dConvertToScreen

始めにAnimation Engineを開いて、start using stack "animationEngine"
サンプルスタックは、下記をメッセージボックスにコピペしてリターンキーを
go stack \
　url "http://kenjikojima.com/livecode/ae/ae3dconverttoscreen.livecode"

３D 座標 の XYZ 点を、レボルーションのカード上又はグループ上の、2次元座標のデフォルトビューに変換します。
最後のパラメータ「focalLength」は、3D オブジェクトからの視点の距離で、数値によって視角が変わってきます。
「focalLength」のデフォルトは、3D オブジェクトが標準的な形に見える値の300 ですが、
数が小さくなると視角が広くなるので、3D オブジェクトの歪みが大きくなります。

基本構文：
ae3dConvertToScreen(3D座標のX点, 3D座標のY点, 3D座標のZ点, \
 カード又はグループ上のX点, カード又はグループ上のY点 [, 焦点距離])

ae3dConvertToScreen(x,y,z,originX,originY[,focalLength])

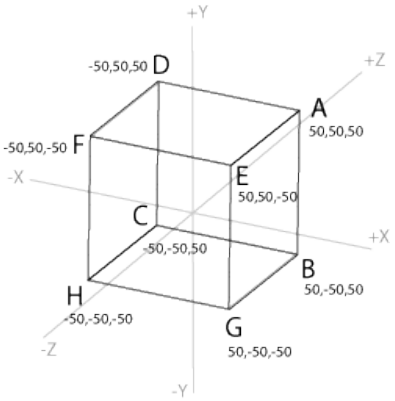
サンプル：
3D 座標の 0 から、それぞれ50ピクセル（タテXヨコX高さ、100x100x100)の立方体をモデルに考えてみます。
図１：立方体の角となるポイントは8箇所ですから、それを表すためのポイントを結ぶ線は 12 必要です。
ポリゴンで一筆書きをして行くと、「ABCDAEFD」「BGE」「FHG」「HC」のポイントを結ぶ、
４つのポリゴンのブロックで（図３ブルーのライン）、立方体を書き表す事ができます。
図２の表は、それぞれのポイントの３D座標とポリゴンにする場合の順序。つまりこの表を
「ae3dConvertToScreen」で平面座標に変換して、ポリゴンを描けば平面上に、
デフォルトの視点からの立体を描くことができます。

カードのサイズ 400 x 400 の中央(3D オブジェクトのオリジンの座標 200, 200) に、
それぞれの3D座標を「ae3dConvertToScreen(x,y,z,200,200)」で、平面座標に変換したのが図4 で、
平面上のポイントを図に書き込んでいます。
「ae3dConvertToScreen」で得られる座標は、Z軸プラスからマイナスに向かって見ている図で、
立方体の面「ABCD」が、もう1方の面「EGHF」より小さく描かれます。

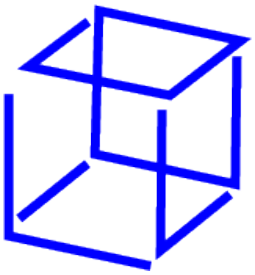
3D座標を変換して、3Dを平面上に描くスクリプト：

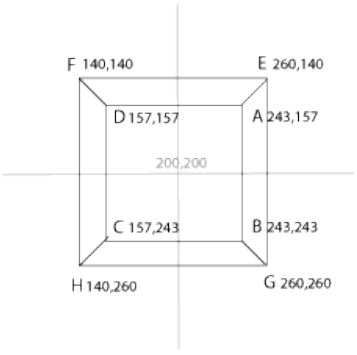
```
-- fld “tPoints” に、図２の3D座標があるとして、
-- グラフィックの polygon 「threeD」は、予め描いておく
-- カードのセンターが 3D オブジェクトのオリジンの座標に設定
 repeat for each line theLine in field "tPoints"
        if theLine = "" then
          put return after output
          next repeat
        end if
        put ae3dconverttoscreen(theLine,the loc of this cd,300) & return after output
 end repeat
 set the points of graphic "threeD" to char 1 to -2 of output
```

このページは、Björnke von Gierke の作ったスタック「Mouse rotator for 3d」を参考にしています。



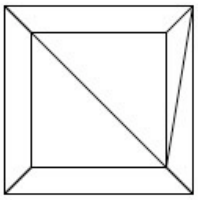

```
A 50,50,50
B 50,-50,50
C -50,-50,50
D -50,50,50
A 50,50,50
E 50,50,-50
F -50,50,-50
D -50,50,50

B 50,-50,50
G 50,-50,-50
E 50,50,-50

F -50,50,-50
H -50,-50,-50
G 50,-50,-50

H -50,-50,-50
C -50,-50,50
```





LIVECODE Community Edition

アニメエンジン日本語ノート



ae3dConvertToScreen 続き

立方体のワイアフレームを、4つのひと筆書きのポリゴンで作ると、線だけの透明な立方体では問題ないですが、
グラフィックをオペークにして、面に色を付けたりすると、右図１のように閉じられていないポイントに、
線が入ってしまいます。これを避けるには、立方体の6面を単位としたグループで、
ポリゴンを構成するのが良いようです。

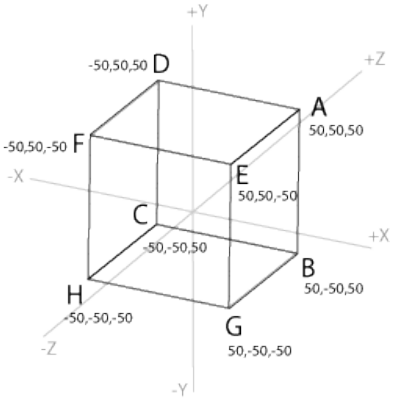
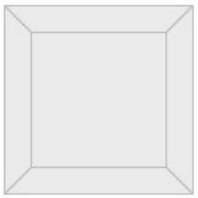


図３のポイントリストで、立体の面に色を付ける

```
on mouseUp
 repeat for each line theLine in field "tPoints"
        if theLine = "" then
         put return after output
         next repeat
        end if
        put ae3dconverttoscreen(theLine,the loc of this cd,300) & return after output
 end repeat
 set the points of graphic "threeD" to char 1 to -2 of output
 set the foregroundcolor of grc "threeD" to 179,179,179
 set the backgroundcolor of grc "threeD" to 230,230,230
 set the opaque of grc "threeD" to true
end mouseUp
```



しかし、とすると？　　Z軸のプラスマイナスは逆向きかもしれない？ と思ったけれど、
オペークの面が向こう側の線を、見えなくはしていないようで、
面に色を付けても、やはりすべての線が表示されます。

```
a 50,50,50
b 50,-50,50
g 50,-50,-50
e 50,50,-50
a 50,50,50

a 50,50,50
e 50,50,-50
f -50,50,-50
d -50,50,50
a 50,50,50

a 50,50,50
b 50,-50,50
c -50,-50,50
d -50,50,50
a 50,50,50

h -50,-50,-50
f -50,50,-50
e 50,50,-50
g 50,-50,-50
h -50,-50,-50

h -50,-50,-50
f -50,50,-50
d -50,50,50
c -50,-50,50
h -50,-50,-50

h -50,-50,-50
c -50,-50,50
b 50,-50,50
g 50,-50,-50
h -50,-50,-50
```