AIMS
関数リファレンス

D.N.A.Softwares 編著

AIMS Ver1.50 準拠

1-1 アクターを操作する関数

◆ アクター操作のまとめ

アクターの操作方法には大きく分けて 2 通りあります。

- アククラスを使う。
- addMover 関数を使い、Mover で制御する

アククラスは Lua スクリプトを用いてきめ細かい制御ができますが、数が増えるとそれだけ CPU パワーを消費し、環境によってはフレームレートの低下を招きます。

Mover はきわめて高速ですが、条件分岐などはできないのであまり凝ったことはできません。

アククラスを使う場合は、生成直後に呼ばれる OnStart, 毎フレーム 1 回呼ばれる OnStep, 消滅時に呼ばれる OnVanish の 3 イベントを実装（＝関数を記述）しなければなりません。例としてアククラス「hoge」を実装する場合の関数定義は次のようになります。

```
1. function hoge_OnStart()
2. end
3.
4. function hoge_OnStep()
5. end
6.
7. function hoge_OnVanish(cause)
8. end
```

hoge_OnVanish だけ引数を一つとっていますが、この引数は「そのアクターがなぜ消滅したか」を表す値です。common.lua に定数定義が準備されています。

表 1-1-1. OnVanish イベントの引数 cause の定数定義

定数名	内容
KILL_ORDERED	vanish で消滅を指示された
KILL_OUTOFSCREEN	画面外に出て自動消去が働いた

◆ createActor

アクターを作成する

■ 構文

```
hActor = createActor(hChip,x,y[,angle,speed],layer[,"class"]);
```

■ 引数

hChip	このアクターに割り当てるチップハンドル。
x,y	アクターの初期座標。
angle	アクターの初期角度。省略時 0。
speed	アクターの初速。省略時 0。
layer	アクターを置くレイヤー。0 ～ 11。
class	アクタークラスを指定。クラスを指定するとアクターは Lua スクリプトで制御されるようになる。

■ 戻り値

hActor	作成されたアクターのハンドル。

■ 解説

- アクターを新規に作成する。
- 初期座標は、シーンクラスから呼んだ場合はシーン原点からの座標を示すが、アクタークラスから呼んだ場合はアクターの位置からの相対座標を指す。
- アクターに初期角度と初速を与えると、生成直後から指定された方向・速度に従い移動を開始する。
- アクタークラスを指定すると、createActor を抜ける前に新規に作成したアクターの OnStart が呼ばれる。

◆ accel

アクターの移動速度を現在速度からの相対値で変更する

■ 構文

```
accel(dspeed);
```

■ 引数

dspeed	相対速度

■ 戻り値

なし

■ 解説

- アクターの速度を相対変化させる。正の値を与えると加速、負の値を与えると減速になる。

◆ addMover

Mover（自動制御要素）を追加する

■ 構文

```
addMover(hActor,starttime,duration,movertype[,arg1[,arg2[,arg3]]]);
```

■ 引数

hActor	追加先のアクターハンドル。
starttime	開始時刻。単位フレーム。負数にすると呼んだ瞬間に 1 ステップ実行。
duration	継続時間。単位フレーム。
movertype	Mover の種類を指定。
arg1...arg3	Mover ごとの引数を指定。Mover の種類により引数の数、内容は異なる

■ 戻り値

なし

■ 解説

- アクターに Mover（自動制御要素）を追加する。
- addMover を呼んでから starttime フレームが経過したところで動作を開始し、そこから duration フレームの間指定した動作を繰り返し実行する。
- 第一引数にはアクターハンドルの指定が必須。自分自身を指定する場合は iSelf() 関数を使うとよい。
- 指定できる movertype およびそれぞれで arg1 ～ 3 に指定すべき値は表 1-1-2 にまとめてある。

表 1-1-2.　movertype の定数名と与える引数一覧

movertype（定数名）	arg(引数)			概要
	1	2	3	
MOVER_SETANGLE	角度			方角指定
MOVER_SETSPEED	速度			速度指定
MOVER_TURN	回転角			旋回
MOVER_ACCEL	加速度			加減速
MOVER_SETFACE	チップハンドル			キャラクタ切り替え
MOVER_SETPOSITION	X	Y		座標指定
MOVER_SETVECTOR	X 速度	Y 速度		ベクトル指定
MOVER_ADDVECTOR	X 加速度	Y 加速度		ベクトル加算
MOVER_SETZOOM	X 拡大率	Y 拡大率		拡大率設定
MOVER_ADDZOOM	X 拡大率増分	Y 拡大率増分		拡大率加減算
MOVER_SETBLEND	ブレンド種別			描画時のブレンド法変更（表 1-1-3 参照）
MOVER_SETALPHA	α値			α値定義
MOVER_ADDALPHA	α値増分			α値加減算
MOVER_SETIMMORTALTIME	無敵時間[フレーム]			無敵(当たり判定無効)時間の設定
MOVER_VANISH				消滅指示
MOVER_SETWAIT	待機時間[フレーム]			処理停止時間の設定
MOVER_SETANIMEPLAY	再生フラグ			アニメ動作フラグ
MOVER_SETHEADING	角度			見た目方角指定
MOVER_TURNHEADING	回転角			見た目旋回
MOVER_SETHEADSYNC	フラグ			見た目と移動方角の同期設定
MOVER_SETCOLOR	R	G	B	色設定
MOVER_SETZ	Z 値			Z 値設定
MOVER_MOVETO_L	行き先 X	行き先 Y		指定座標へ移動(線形)
MOVER_MOVETO_A	行き先 X	行き先 Y		指定座標へ移動(加速)
MOVER_MOVETO_B	行き先 X	行き先 Y		指定座標へ移動(ブレーキ)
MOVER_MOVETO_C	行き先 X	行き先 Y		指定座標へ移動(両方)

movertype（定数名）	arg(引数)			概要
	1	2	3	
MOVER_ZOOMTO_L	拡大率 X	拡大率 Y		指定拡大率に変化 (線形)
MOVER_ZOOMTO_A	拡大率 X	拡大率 Y		指定拡大率に変化 (加速)
MOVER_ZOOMTO_B	拡大率 X	拡大率 Y		指定拡大率に変化 (ブレーキ)
MOVER_ZOOMTO_C	拡大率 X	拡大率 Y		指定拡大率に変化 (両方)
MOVER_INVERT_OBCHECK	フラグ			画面外判定処理を レイヤー設定と逆にする

表 1-1-3.　MOVER_SETBLEND のブレンド種別 (arg1) の定数一覧

定数名	内容
BLEND_NONE	ブレンド無効
BLEND_NORMAL	半透明（デフォルト）
BLEND_ADD	加算ブレンド
BLEND_MUL	乗算ブレンド
BLEND_REVERSE	反転ブレンド

◆ clearMover

Mover(自動制御要素）をすべて消去する

■ 構文

```
clearMover([hActor]);
```

■ 引数

hActor　　Mover を消去するアクターのハンドル。省略時自分自身。

■ 戻り値

なし

■ 解説

- 割り当てられている Mover をすべて消去する。動作中の Mover も中断する。

◆ cls

シーン内の全アクターを消去する

■ 構文

```
cls([layer]);
```

■ 引数

layer	アクターを消去したいレイヤー。省略すると全レイヤーの全アクターを消す。

■ 戻り値

なし

■ 解説

- 指定レイヤー、または全レイヤーのアクターを消す。

◆ deleteHitOverride

アクターの当たり判定上書きをキャンセルし、チップの当たり判定に戻す

■ 構文

```
deleteHitOverride([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

なし

■ 解説

- setHitOverride で指定した当たり判定上書きをキャンセルする。

◆ getAlpha

アクターのアルファ値（不透明度）を返す

■ 構文

```
alpha = getAlpha([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

alpha	そのアクターのアルファ値。0 ～ 255。

■ 解説

- アクターのアルファ値（不透明度）を返す。

◆ getAngle

アクターの角度を返す

■ 構文

```
angle = getAngle([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

angle	アクターの角度（移動方向）

■ 解説

- アクターの移動方向を返す。

◆ getColor

アクターの描画色を得る

■ 構文

```
r,g,b,a = getColor([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

r,g,b	RGB 各成分。
a	α 値。

■ 解説

- 指定したアクターの描画色を得る。すべて値は 0 ～ 255 の範囲。

◆ getHeading

アクターの見た目の角度を返す

■ 構文

```
heading = getHeading([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

heading	アクターの見た目の角度。

■ 解説

- アクターの見た目の角度を返す。setHeading で見た目の向きだけを変えていた場合、移動方向とは一致しない場合がある。

◆ getHitLayer

特定レイヤーのすべてのアクターとの当たり判定

■ 構文

```
table = getHitLayer([hActor,]layer);
```

■ 引数

hActor	当たり判定をとる対象のアクター。省略時自分自身。
layer	判定対象のレイヤー。

■ 戻り値

table	当たっていたアクターハンドルが入ったテーブル

■ 解説

- hActor で指定したアクター（または自分自身）と指定レイヤーのアクター全てとの当たり判定をとり、当たったアクターのハンドルをテーブルに入れ返す。どのアクターとも当たらなかった場合は空のテーブルを返す。

◆ getPattern

アクターに割り付けられているチップハンドルを得る

■ 構文

```
hChip = getPattern([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

hChip	チップハンドル。

■ 解説

- アクターが今表示しているチップハンドルを得る。

◆ getSpeed

アクターの速度を返す

■ 構文

```
speed = getSpeed([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

speed	アクターの速度。

■ 解説

- アクターの速度を返す。

◆ getUptime

アクターの生成からの経過時間を返す

■ 構文

```
time = getUptime([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

time	アクターの生成からの経過時間。

■ 解説

- アクターが生成されてから経過した時間をフレーム単位で返す。

◆ getv

アクターの値スロットから値を読み出す

■ 構文

```
value = getv([hActor],slot);
```

■ 引数

hActor	値を読み出すアクター。省略時自分自身。
slot	値スロットの番号。0 ～ 63。

■ 戻り値

value	読み出した値。

■ 解説

- 指定したアクターの値スロットを読み出す。
- 自分自身の値スロットを読む場合は配列変数 VAR を経由するとシンプルになる。

◆ getVX

アクターの移動ベクトルの X 成分を得る

■ 構文

```
vx = getVX([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

vx	移動ベクトル X 成分。

■ 解説

- setVector や setAngle、setSpeed によって設定された、アクターの移動ベクトルの X 成分を得る。

◆ getVY

アクターの移動ベクトルの Y 成分を得る

■ 構文

```
vy = getVY([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

vy	移動ベクトル Y 成分。

■ 解説

- アクターの移動ベクトルの Y 成分を得る。

◆ getX

アクターの X 座標を得る

■ 構文

```
x = getX([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

x	アクターの X 座標。

■ 解説

- 指定したアクターの X 座標を得る。

◆ getY

アクターの Y 座標を得る

■ 構文

```
y = getY([hActor])
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

y	アクターの Y 座標。

■ 解説

- 指定したアクターの Y 座標を得る。

◆ getZ

アクターの Z 値を得る。

■ 構文

```
z = getZ([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

z	アクターの Z 値。

■ 解説

- Z ソート時の並べ替え基準となる Z 値を得る。生成直後は 0。

◆ getZX

アクターの X 方向拡大率を得る

■ 構文

```
zx = getZX([hActor])
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

zx	アクターの X 方向拡大率。1.0 が原寸。

■ 解説

- アクターの X 方向拡大率を得る。

◆ getZY

アクターの Y 方向拡大率を得る

■ 構文

```
zy = getZY([hActor])
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

zy	アクターの Y 方向拡大率。1.0 が原寸。

■ 解説

- アクターの Y 方向拡大率を得る。

◆ getNearestActor

指定レイヤーのアクターの中から、特定アクターに最も近いものを得る

■ 構文

```
hNearestActor,angle,distance = getNearestActor([Layer[,hSrcActor]);
```

■ 引数

Layer	調べるレイヤー。省略時は hSrcActor のあるレイヤーを調べる。
hSrcActor	基準アクター。省略時自分自身。

■ 戻り値

hNearestActor	hSrcActor の最も近傍にあったアクターのハンドル。
angle	hSrcActor から hNearestActor への方角。単位は度。
dictance	hSrcActor から hNearestActor への距離。

■ 解説

- 指定レイヤーのアクターのうち、hSrcActor に最も近いアクターを探し、そのハンドルと、方角、距離を返す。
- レイヤーに自分以外のアクターが無かった場合は nil を返す。

◆ iLayerActors

指定レイヤーに存在するアクターのハンドルをすべて得る

■ 構文

```
table = iLayerActors(layer);
```

■ 引数

layer	ハンドルを得る対象のレイヤー。

■ 戻り値

table　　ハンドルが入ったテーブル。

■ 解説

- 指定レイヤーにあるアクターのハンドルをテーブルの形で得る。

◆ iMaster

一次生成元のアクターハンドルを返す

■ 構文

```
hActor = iMaster()
```

■ 引数

なし

■ 戻り値

hActor　　一次生成元のアクターハンドル。

■ 解説

- 一次生成元 (シーンクラスで生成されたアクター) の ID を得る。自分自身がステージ制御部で生成されているのなら自分自身の ID が帰る。
- シーンがアクター A を作成、アクター A のクラスがアクター B、アクター B のクラスがアクター C を生成したとき、アクター C から iMaster() を呼ぶとアクター A のハンドルを返す。

◆ iParent

自分を生成したアクターのハンドルを得る

■ 構文

```
hActor = iParent()
```

■ 引数

なし

■ 戻り値

hActor　　生成元アクターのハンドル。

■ 解説

- 自分を生成したアクターのハンドルを得る。自分がシーンクラスで生成されていた場合、この関数は自分自身のハンドルを返す。

◆ isAlive

指定したアクターがまだ存在しているか返す

■ 構文

```
flag = isAlive(hActor)
```

■ 引数

hActor　　対象のアクター。

■ 戻り値

flag　　存在しているなら true、すでに消滅しているなら false。

■ 解説

- 指定したアクターの存在を調べる。

◆ iSelf

「自分自身」のハンドルを得る

■ 構文

```
handle = iSelf()
```

■ 引数

なし

■ 戻り値

handle	アクタークラスで呼んだ場合は呼びだし元アクターのハンドル、シーンクラスで呼んだ場合はシーンのハンドル

■ 解説

- 「自分自身」のアクターまたはシーンハンドルを返す。文脈によって戻り値が変わる関数。
- アクタークラス (アクターの OnStart,OnStep,OnVanish, またはアクターのスレッド) の中で呼ぶと、この関数はアクターのハンドルを返す。
- シーンクラス (シーンの OnStart,OnStep,OnClose、またはシーンのスレッド) の中で呼ぶと、この関数はシーンのハンドルを返す。
- アクタークラスの中からアクターが存在するシーンのハンドルを得たい場合は iThisScene 関数を使う。

◆ isHitActor

特定アクターとの当たり判定

■ 構文

```
flag = isHitActor([hActorA],hActorB);
```

■ 引数

hActorA,hActorB 当たり判定をとるアクター。一方を省略すると自分と他のアクターとの判定になる。

■ 戻り値

flag 当たり判定が重なっていたら true。さもなければ false

■ 解説

- 指定した 2 つのアクター、または自分自身と指定したアクターとの当たり判定をとる。
- 当たり判定矩形はあらかじめチップに設定しておくこと。
- 一対多数の当たり判定を調べる場合は getHitLayer が便利。

◆ release

アクタークラス・シーンクラスを終了する

■ 構文

```
release();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- このアクター、またはシーンのスクリプト駆動を終了する。
- アクタークラスで呼んだ場合はアクターのスクリプトが次フレームから呼ばれなくなる。アクターの消滅ではないので、OnVanish 関数は呼ばない。
- シーンクラスで呼んだ場合はシーンクラスの停止となり、OnClose 関数を呼んでからシーンクラスが止まる。closeScene と違ってシーンは閉じられない。
- どちらの場合でも、startThread 関数で動作させたスレッドは動作し続ける。

◆ setAngle

アクターの移動方向を変更する

■ 構文

```
setAngle(speed)
```

■ 引数

speed	アクターの速度。

■ 戻り値

なし

■ 解説

- アクターの速度を変更する。単位は度。
- 0 以下または 360 以上の値を与えても内部で 0 ～ 360 の範囲に丸められる。
- この関数を呼ぶと設定されている速度（setSpeed 参照）に基づいて移動ベクトルが再計算される。

◆ setAnimeActive

チップに設定されたアニメーションの有効・無効を設定する

■ 構文

```
setAnimeActive(flag);
```

■ 引数

flag	アニメーションを有効にするかのフラグ。true で有効、false で無効。

■ 戻り値

なし

■ 解説

- チップに割り当てた自動アニメーションを有効にするかを設定する。false に設定するとアニメが止まる。初期状態ではアニメーションは有効になっている。

◆ setHeading

アクターの見た目の角度を変更する

■ 構文

```
setHeading(angle);
```

■ 引数

angle	見た目の角度。

■ 戻り値

なし

■ 解説

- アクターの見た目の角度を変更する。単位は度。
- 0 以下または 360 以上の値を与えても内部で 0 ～ 360 の範囲に丸められる。
- setAngle などで設定した移動方向には影響しない。

◆ setHeadSync

移動方向と見た目の向きの同期を設定

■ 構文

```
setHeadSync(flag);
```

■ 引数

flag	向きの同期フラグ。true で同期、false で同期をはずす

■ 戻り値

なし

■ 解説

- setAngle などで設定できる移動方向と、setHeading などで設定できる見た目の向きを同期させるかどうかを設定する。
- 初期状態では、createActor で向きを指定して生成した場合は同期する（true）、向きを指定しなかった場合は同期しない（false）設定になっている。

◆ setHitOverride

アクターの当たり判定を一時的に上書きする

■ 構文

```
setHitOverride([hActor,]x,y,w,h);
```

■ 引数

hActor	対象のアクター。省略時自分自身。
x,y	当たり判定矩形の左上座標。
w,h	当たり判定矩形の幅と高さ。

■ 戻り値

なし

■ 解説

- グラフィックチップに指定されている当たり判定矩形を指定したアクターに限り一時的にキャンセルし、別の当たり判定矩形を割り当てる。元に戻すには deleteHitOverride 関数を呼ぶ。

◆ setPattern

アクターのグラフィックチップを切り替える

■ 構文

```
setPattern(hImage);
```

■ 引数

hImage	切り替え先のチップハンドル。

■ 戻り値

なし

■ 解説

- アクターに割り当てているチップハンドルを切り替える。

◆ setPos

アクターの座標を変更する

■ 構文

```
setPos(x,y)
```

■ 引数

x,y	新しい座標。

■ 戻り値

なし

■ 解説

- アクターの座標を変更する。ベクトルなどほかの要素は影響を受けない。

◆ setSpeed

アクターの速度を変更する

■ 構文

```
setSpeed(speed);
```

■ 引数

speed	アクターの速度

■ 戻り値

なし

■ 解説

- アクターの速度を変更する。単位はピクセル / フレーム。負を与えてもかまわない。
- この関数を呼ぶと設定されている角度（setAngle 参照）に基づいて移動ベクトルが再計算される。

◆ setv

値スロットに数値を設定する

■ 構文

```
setv([hActor,]slot,value);
```

■ 引数

hActor	対象のアクター。省略時自分自身。
slot	値スロットの番号。0 ～ 63。
value	書き込む値。

■ 戻り値

なし

■ 解説

- 指定したアクターの値スロットに数値を書き込む。
- 自分自身の値スロットに書き込む場合は配列変数 VAR を経由するとシンプルになる。

◆ setVector

アクターの移動ベクトルを設定する

■ 構文

```
setVector(vx,vy);
```

■ 引数

vx,vy	アクターの移動ベクトル。

■ 戻り値

なし

■ 解説

- アクターに移動ベクトルを設定する。この関数でベクトルを設定すると、方角、速度は指定したベクトルに従って再設定される。

◆ setZ

アクターの Z 値を設定する

■ 構文

```
setZ(z);
```

■ 引数

z	Z 値。実数。

■ 戻り値

なし

■ 解説

- Z ソート時の並べ替え基準となる Z 値を設定する。生成直後は 0。
- Z ソートを用いるには、setLayerUseFastDraw でレイヤーの高速描画を有効にするか、setGlobalZSort でグローバル Z ソートを行う設定をしなければならない。
- 優先順位は Z 値の低い方がより手前（描画順としては後）になる。
- Z 値の範囲は描画に使用している Luna ライブラリの仕様により、-4096 ～ 4095 になっている。この値を超えてもエラーにはならないが、レイヤーの高速描画を有効にすると描画されなくなる。

◆ thaw

アクターの動作停止状態を解除する

■ 構文

```
thaw(hActor)
```

■ 引数

hActor	対象のアクター。

■ 戻り値

なし

■ 解説

- addMover の MOVER_SETWAIT コマンドで動作停止状態になっているアクターの動作を再開させる。

◆ turn

アクターを相対的に回転させる

■ 構文

```
turn(delta);
```

■ 引数

`delta`　　回転させる角度。

■ 戻り値

なし

■ 解説

- アクターの移動方向を今の角度から delta だけ変化させる。

◆ turnHeading

アクターの見た目の向きを相対的に回転させる

■ 構文

```
turnHeading(delta);
```

■ 引数

delta	回転させる角度。

■ 戻り値

なし

■ 解説

- アクターの見た目の向きを今の角度から delta だけ変化させる。

◆ vanish

アクターを消去する

■ 構文

```
vanish([hActor]);
```

■ 引数

hActor	消去するアクター。省略時自分自身（アクタークラスからの呼び出しに限る）

■ 戻り値

なし

■ 解説

- 指定されたアクターを消去する。アクタークラスで呼びだした場合は引数を省略でき、その場合は自分自身を消去する。

◆ VAR

「自分自身」の値スロットを示すシステム変数

■ 構文

```
value = VAR[slot];
VAR[slot] = value;
```

■ 引数

slot	値スロットの番号。0 ～ 63。

■ 戻り値

なし

■ 解説

- アクタースクリプト内で、「自分自身」の値スロットを示すシステム変数。すなわち、VAR[slot] への代入は setv と、VAR[slot] の参照は getv とそれぞれ等価である。

1-2 レイヤーを操作する関数

◆ clearLayerSettings

レイヤー設定を起動直後の状態に戻す

■ 構文

```
clearLayerSettings();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- シーンのすべてのレイヤーの各種設定を初期化する。
- 具体的には「仮想解像度、スクロール、高速描画関係の全設定、自動 Z 値設定、自動アクター消去の範囲、キャラクターマップの設定、レイヤーの動作停止状態」が含まれる。

◆ getLayerScrollX

レイヤーの X スクロール量を得る

■ 構文

```
sx = getLayerScrollX(layer);
```

■ 引数

layer	対象のレイヤー番号。

■ 戻り値

sx	レイヤーの X 方向スクロール量。

■ 解説

- 指定レイヤーの X 方向のスクロール量を得る。

◆ getLayerScrollY

レイヤーの Y スクロール量を得る

■ 構文

```
sy = getLayerScrollY(layer);
```

■ 引数

layer	対象のレイヤー番号。

■ 戻り値

sy	レイヤーの Y 方向スクロール量。

■ 解説

- 指定レイヤーの Y 方向のスクロール量を得る。

◆ setGlobalZSort

レイヤーをまたいだ Z ソート描画を設定する

■ 構文

```
setGlobalZSort(flag[,layer_from,layer_to]);
```

■ 引数

flag	Z ソートを有効にするか。true で有効。false で無効。
layer_from	Z ソート開始レイヤー。
layer_to	Z ソート終了レイヤー。layer_from 以上を指定すること。

■ 戻り値

なし

■ 解説

- 複数レイヤーのアクターをまとめて Z ソートする描画を有効にする。
- layer_from、layer_to を省略した場合は最後に設定した値を引き継ぐ。
- 通常はレイヤー番号の若い方、アクターの生成が早かった方が先に描画（より奥に表示）されるが、この設定を入れると layer_from と layer_to のレイヤー間に限り、setZ などで与えた Z 値に従って描画順が決定される。Z 値が低い方がより奥に、高い方はより手前になる。
- layer_from と layer_to は同じ値を指定してもいい。その場合、指定したレイヤーのみ Z ソートを行う。
- setLayerFastDraw による単独レイヤーの高速描画と比べると、テクスチャの使用などに制限がないため融通は利く一方、ソートの分だけ描画のコストが上昇するため、マシンによっては処理落ちが通常より多く発生する恐れがある。大量のアクターが存在するレイヤーに対してこの Z ソートを使うべきではない。

■ 例

```
--レイヤー1からレイヤー5までをZソート表示する
setGlobalZSort(true,1,5);

--レイヤー2だけZソート
setGlobalZSort(true,2,2);

--ソートしない
setGlobalZSort(false);
```

◆ setLayerAutoVanish

レイヤーのアクター自動消去を設定する

■ 構文

```
setLayerAutoVanish(layer,flag);
```

■ 引数

layer	対象のレイヤー番号。
flag	自動消去を行うかどうか。true でオン。false でオフ。

■ 戻り値

なし

■ 解説

- 指定したレイヤーについて、特定の範囲の外に出たアクターを自動消去するかどうかを設定する。
- 自動消去判定を行う範囲は初期状態では画面全体、すなわち画面の外に出たら自動消滅となる。この矩形はsetLayerAutoVanishRect関数で再設定できる。

◆ setLayerAutoVanishRect

レイヤーのアクター自動消去の境界線を設定する

■ 構文

```
setLayerAutoVanishRect(layer,x,y,x2,y2);
```

■ 引数

layer	対象のレイヤー番号。
x,y	境界線矩形の左上座標。
x2,y2	境界線矩形の右下座標。

■ 戻り値

なし

■ 解説

- 指定したレイヤーについて、自動消去の判定を行う矩形を設定する。この矩形の外側に出たアクターは自動消去の対象となる。
- アクター側のはみ出し判定はチップの矩形を用いるが、回転を考慮しないので、場合によってはアクターの一部が矩形の内側にあるのに自動消去されることがある。

■ 例

```
-- レイヤー1の境界線を(200,50)から高さと幅それぞれ400の正方形にする。
-- 右下座標を設定するので、高さと幅を直接指定していないことに注意。
setLayerAutoVanishRect(1,200,50,599,449);

-- 座標は画面をはみ出したりマイナスになってもかまわない。
-- 640x480の画面から各辺が外に10ドットずつはみ出した矩形を設定
setLayerAutoVanishRect(1,-10,-10,660,500);
```

◆ setLayerBilinear

レイヤーのバイリニアフィルタを設定する

■ 構文

```
setLayerBilinear(layer,flag);
```

■ 引数

layer	対象のレイヤー番号。
flag	テクスチャフィルタの選択。true でバイリニアフィルタ、false でニアレストネイバー。

■ 戻り値

なし

■ 解説

- 指定したレイヤーの描画時に使うフィルタを指定する。
- true にするとバイリニアフィルタを使う。回転、拡大縮小時に補間を行いなめらかな表示が得られるが、元画像をそのまま表示する場合でもドットがにじみ、絵がぼやける場合がある。
- false にすると補間を使わない、いわゆるニアレストネイバー法になる。回転、拡大縮小をしない場合は元画像がそのまま表示されるが、回転などを行うと「荒れた」感じの表示となる。
- どっちがよいというのは一概には言えないので、そのレイヤーに置くアクターの性質にあわせて選択することが必要である。

◆ setLayerFreeze

レイヤーの処理停止を設定する

■ 構文

```
setLayerFreeze(layer,flag);
```

■ 引数

layer	対象のレイヤー番号。
flag	処理停止フラグ。true で停止。false で再開。

■ 戻り値

なし

■ 解説

- 指定したレイヤーに属するすべてのアクターの動作を止める。（描画は続ける）
- flag を true にすると停止、false にすると動作を再開する。

◆ setLayerGraphicForFastDraw

レイヤーの高速描画モードにおいて使用するテクスチャを指定する

■ 構文

```
setLayerGraphicForFastDraw(layer,hChip);
```

■ 引数

layer	対象のレイヤー番号。
hChip	そのレイヤーに割り当てるテクスチャ。指定チップに使われているテクスチャが割り付けられる。

■ 戻り値

なし

■ 解説

- 指定レイヤーに対して、高速描画モードで使うテクスチャを指定する。
- テクスチャ指定となっているが、実際に指定するのはチップハンドル。そのチップに割り当てられたテクスチャを使用する。
- cutGraphic で切り出したチップは切り出し元のチップととテクスチャを共有しているので、元テクスチャから cutGraphic で作成したチップであれば正常に描画できる。
- 高速描画モードについてそのほか詳しくは setLayerUseFastDraw 関数を参照。

◆ setLayerScroll

レイヤーのスクロール量を設定する

■ 構文

```
setLayerScroll(layer,sx,sy);
```

■ 引数

layer	対象のレイヤー番号。
sx,sy	X,Y それぞれのスクロール量。

■ 戻り値

なし

■ 解説

- 指定レイヤーについて、画面左上角の座標が (sx,sy) となるようにスクロールを設定する。負数も指定できる。
- スクロールを行った場合でも、自動消去の矩形は画面上に固定されたようになる。たとえば画面端が自動消去の境界だった場合、スクロールを行った結果アクターが画面外に出た場合は自動消滅の対象となる。

◆ setLayerUseAutoYSort

レイヤーのアクターに対する自動 Z 値調整 (Y 座標ソート) を設定する

■ 構文

```
setLayerUseAutoYSort(layer,flag);
```

■ 引数

layer	対象のレイヤー番号。
flag	Y ソートを使うなら true、使わないなら false。

■ 戻り値

なし

■ 解説

- 指定レイヤーに対して、そのレイヤーに属するアクターの Z 値を自動的にアクターの Y 座標値に一致させる処理を行うようにする。
- この設定を入れた状態で、描画時の Z ソートを有効にする（setLayerUseFastDraw または setGlobalZSort）と、Y 座標が大きい（＝画面下にいる）アクターほど手前に描画されるようになる。

◆ setLayerUseFastDraw

レイヤーの高速描画モードを設定する

■ 構文

```
setLayerUseFastDraw(layer,flag);
```

■ 引数

layer	対象のレイヤー番号。
flag	高速描画モード指定。true でオン、false でオフ。

■ 戻り値

なし

■ 解説

- 指定したレイヤーの高速描画モードを設定する。
- 高速描画とは、チップの描画方法に制限が加わる代わりに描画速度を大幅に引き上げるモードである。具体的には「使用できるテクスチャが 1 レイヤーにつき 1 枚だけ」「ブレンドモードの設定不可、通常の α 合成のみ」の二点の制限がかかる。
- 一方、高速描画中はアクターの Z 値に基づいた前後関係のソートが有効になる。すべてのアクターの Z 値を同値にしておけば描画順は通常描画時と同じになる。

■ 例

```
g_base = loadGraphic("texture.png"); -- 元の画像
g_chara = cutGraphic(g_base,0,0,32,32); -- 切り出したチップ

--レイヤー5を高速描画モードに
setLayerUseFastDraw(5,true);
--読み込んだテクスチャをレイヤー5に割り当て
setLayerGraphicForFastDraw(5,g_base);

--このアクターは高速描画される
createActor(g_chara,320,240,5);
```

1-3 シーンを操作する関数

◆ シーン操作のまとめ

シーンはゲームにおける「場面」を表現する概念です。アクタークラスのようにシーンにおいても「シーンクラス」が存在し、アクター同様に所定のタイミングでイベントが起動します。

シーンクラスにおいてはシーン動作開始時の OnStart、動作中毎フレーム呼ばれる OnStep、終了時に呼ばれる OnClose の 3 イベントが実装されていなければなりません。例として、シーンクラス「foo」を定義する場合の関数定義は次のようになります。

```
1. function foo_OnStart()
2. end
3.
4. function foo_OnStep()
5. end
6.
7. function foo_OnClose()
8. end
```

今動いているシーンを終了して次のシーンに切り替えるときは changeScene 関数を使います。addScene 関数を使うと今動いているシーンはそのままにさらに新しくシーンを追加することができます。この場合、既存のシーンと新しいシーンはほぼ並列に動作すると考えて差し支えありません。

◆ addScene

シーンを新たに追加する

■ 構文

```
hScene = addScene("sceneclass",addTop[,scenegroup]);
```

■ 引数

sceneclass	シーンクラス名。
addTop	シーングループ内のどこに配置するか。true なら他のシーンの手前、false なら他のシーンの背後。
scenegroup	追加先のシーングループを指定。0 ～ 3。省略時 0（ブートシーンと同じグループ）

■ 戻り値

hScene	追加したシーンを指すハンドル。

■ 解説

- 指定したクラスのシーンを新しく追加する。
- 追加したシーンは他のシーンと並列に動作する。（正確にはシーングループ番号の若いグループの、さらに描画順が奥になる方から順番に実行される）
- 既存シーンとの位置関係は scenegroup の値と addTop の値で調整する。
- シーンクラスについては別項参照のこと。

■ 例

```
-- 新しいシーンをブートシーンの上に置く
addScene("newscene",true,0);

-- シーンクラス newsceneの実体
function newscene_OnStart()
  ..
end

function newscene_OnStep()
  ..
end

function newscene_OnClose()
  ..
end
```

◆ changeScene

稼働中のシーンを切り替える

■ 構文

```
changeScene([hScene,]"sceneclass");
```

■ 引数

hScene	対象のシーンハンドル。省略時自分自身。
sceneclass	切り替え先のシーンクラス。

■ 戻り値

なし

■ 解説

- 今のシーンクラスを終了し、指定したシーンクラスに移る。閉じるのではなく、あくまで切り替えであり、シーンハンドルはそのまま引き継がれる。
- 切り替えを指示すると、次のフレームで今のシーンクラスの OnClose →新しいシーンクラスの OnStart →新しいシーンクラスの OnStep　という順番で各イベントが呼び出される。

◆ closeScene

稼働中のシーンを閉じる

■ 構文

```
closeScene([hScene]);
```

■ 引数

hScene	対象のシーンハンドル。省略時自分自身。

■ 戻り値

なし

■ 解説

- 指定したシーンを閉じる。この関数を呼んだ次のフレームで OnClose 関数が呼ばれ、シーンが消去される。すべての動作中のシーンが閉じられるとAIMS は終了するが、AIMS を終了させる目的であれば quit() 関数を呼んだ方が単純である。
- すでに解放されたシーンハンドルを指定するとエラーになる。動作しているかどうか不定なシーンを閉じようとするときは、直前に isSceneAlive 関数でシーンの存在を確認するとよい。

◆ getSceneUptime

シーンの動作開始からの経過時間を得る

■ 構文

```
frame = getSceneUptime([hScene]);
```

■ 引数

hScene	対象のシーンハンドル。省略時自分自身。

■ 戻り値

frame	経過時間。単位フレーム。

■ 解説

- 指定シーンの OnStart 関数が呼ばれた時点を起点として、そこから経過した時間をフレーム単位で得る。changeScene でシーンを切り替えた場合は、シーンが切り替わった時点で経過時間は 0 にリセットされる。

◆ isSceneAlive

シーンが動作中かどうかを得る

■ 構文

```
state = isSceneAlive(hScene);
```

■ 引数

hScene	対象のシーンハンドル。

■ 戻り値

state	存在していれば true、すでに消去されていれば false。

■ 解説

- 指定シーンが存在するかどうかを返す。

■ 例

```
-- 別の場所で閉じられている可能性があるシーン Scene を閉じるとき
-- まずそのシーンの存在を確認してから閉じる
if isSceneAlive(Scene) then
  closeScene(Scene);
end
```

◆ setBGColor

背景色を設定する

■ 構文

```
setBGColor(r,g,b[,a]);
```

■ 引数

r,g,b	背景色の RGB 成分。各 0 ～ 255。
a	α 成分。0 ～ 255。省略時 255。

■ 戻り値

なし

■ 解説

- 背景色（毎フレーム開始時に、前の画面を消去するのに使う色）を指定する。
- α が指定できるが、これを 255 より少なくすると前の画面が完全に消えずに残るようになる。

■ 例

```
-- 背景を青色にする
setBGColor(0,255,0);
-- 背景を黒の不透明度25%にする。
-- アクターを動かすと残像が残ったような絵になる
setBGColor(0,0,0,64);
```

◆ setSceneFreeze

シーンの処理停止を設定する

■ 構文

```
setSceneFreeze([hScene,]flag)
```

■ 引数

hScene	対象のシーンハンドル。省略時自分自身。
flag	シーンを止めるなら true、動作させるなら false。

■ 戻り値

なし

■ 解説

- 指定したシーンの動作を止める。
- すべてのアクターの動作とシーンクラス、およびそのシーンが呼び出したスレッドの動作が停止する。（ローダースレッドは除く）描画は毎フレーム行われる。
- 自分自身で停止を指示した場合は、そのフレームの処理は一通り行った後次のフレームから止まる。他のシーンだった場合、自分より後の実行順（描画順が自分より後）であれば、今のフレームから停止する。
- 自分自身を停止させる場合は他のシーンから解除を指示しなければならないことに注意。一つしかシーンが存在しないとき、自分自身を止めると解除する方法がなくなり、プログラムが続行できなくなる。

■ 例

```
--あるシーン／アクターのスレッド関数
function somethread()
  setSceneFreeze(SLAVE,true);  -- シーン SLAVE を止める
  wait(60);                    -- 1秒待って
  setSceneFreeze(SLAVE,false); -- 再開する
end
```

◆ setSceneGroupVisible

シーングループの表示・非表示を設定する

■ 構文

```
setSceneGroupVisible(scenegroup,flag);
```

■ 引数

scenegroup	シーングループ番号。0 ～ 3。
flag	指定グループを表示するなら true。隠すなら false。

■ 戻り値

なし

■ 解説

- 指定したシーングループを描画するかどうか設定する。
- 単純に描画するかどうかの指定であり、描画していない場合でもイベント類の呼び出しや、アクターの移動などの処理は行われる。アクターが Lua スクリプトでの描画を選択している場合、OnDraw 関数は呼ばれない。また、renderSceneGroupToGraphic 関数でテクスチャへの書き出しをする際はこの関数の設定に関わらず、描画が行われる。

◆ setScreensize

画面の仮想解像度を変更する

■ 構文

```
setScreenSize(w,h);
```

■ 引数

w,h	仮想画面の幅と高さ。正の値。

■ 戻り値

なし

■ 解説

- 画面の仮想解像度を設定する。物理解像度は起動時の設定から変更できない。
- スクロールをしていない状態で、画面左上が (0,0) 右下が (w-1,h-1) を指すように座標変換が行われる。
- 仮想解像度の方が物理解像度より広ければ、グラフィックは縮小して描画され、逆に仮想解像度が物理解像度より狭ければ、グラフィックは拡大される。

◆ setSceneExclusive

シーンを排他モードで動作させる

■ 構文

```
setSceneExclusive([hScene,]flag);
```

■ 引数

hScene	シーン指定。省略時自分自身。
flag	シーンを排他モードにするか。true で排他指定。

■ 戻り値

なし

■ 解説

- hScene で指定したシーンを排他モードに設定する。または解除する。
- 排他モードとは指定したシーンだけを動作させ、そのシーンが閉じられる、もしくは排他モードが解除されるまで他のシーンの進行 (描画処理および OnDraw 関数を除く全動作) を抑制させるモードである。一度に一つのシーンしか排他モードに指定することはできない。
- setSceneIgnoreExclusive 関数で排他モード例外を設定したシーンは排他モードに入ったシーンが存在する場合でも引き続き動作する。
- 排他モードのシーンから新たにシーンを作ることはできるが、排他モード例外の設定をしなければ新しいシーンは起動しないので注意すること。
- 排他シーン例外を指定しているシーンで setSceneExclusive 関数を宣言して排他モードを再度宣言した場合、それまで排他モード扱いとなっていたシーンは停止する。(排他モード例外が指定されていれば、引き続き動作する)

◆ setSceneIgnoreExclusive

シーンを排他モードの例外に設定する

■ 構文

```
setSceneIgnoreExclusive([hScene,]flag);
```

■ 引数

hScene	シーン指定。省略時自分自身。
flag	このシーンを排他モードの例外にするか。true で排他例外指定。

■ 戻り値

なし

■ 解説

- hScene で指定したシーンを排他モードの例外に設定、または解除する。

◆ iSceneExclusive

排他モードにされているシーンのハンドルを得る構文

■ 構文

```
hScene = iSceneExclusive();
```

■ 引数

なし

■ 戻り値

hScene	排他モードになっているシーン。排他モードでなければ nil。

■ 解説

- 排他モードで動作中のシーンのハンドルを得る。排他シーンで動作しているシーンがなければ、nil が返る。

◆ iThisScene

自分自身のシーンハンドルを得る

■ 構文

```
hScene = iThisScene();
```

■ 引数

なし

■ 戻り値

hScene	自分自身を示すシーンハンドル。

■ 解説

- シーンクラスで呼んだ場合は自分自身のシーンハンドル、アクタークラスで呼んだ場合はアクターが所属するシーンのハンドルが返る。
- iSelf() をシーンクラスで呼んだ場合も同じ結果になるが、iSelf はアクタークラスで呼ぶとアクターハンドルを返すのに対し、iThisScene では常にシーンハンドルを得られる。

1-4 グラフィックチップ・テクスチャを操作する関数

◆ グラフィックチップのまとめ

AIMS ではアクターに与える画像をグラフィックまたはチップと呼んでいます。グラフィックを準備しないことには画面には何も出せません。

画像を読み込むのは loadGraphic 関数を使います。この状態では画像全体が 1 枚のチップとして扱われます。

1 枚の画像に 1 つのチップしかに入れないのでは、ゲームによっては何百枚もの画像ファイルを用意せねばならず現実的ではありません。AIMS では、1 枚の画像から複数のチップを「切り出し」て利用できるよう、cutGraphic という関数が用意されています。チップの一部を切り出しますが、内部では同じメモリを共有するため、1 枚の画像から大量にチップを切り出した場合でも、メモリ消費はかなり抑えられます。

実際にゲームでチップを用意する流れは、だいたい下記の通りになります。

- 複数のチップをまとめた画像を loadGraphic で読み込む。
- その画像から cutGraphic で各チップを切り出す。
- 切り出したチップごとに当たり判定などを設定する。

実際に画像を読み込み、そこからチップを切り出すコードの例を以下に示します。

```
G = {}; -- チップハンドルをまとめて保持しておくテーブル
-- シーンの開始時……
function scene_OnStart()
  -- いくつかのキャラクタをまとめた画像ファイルを読み込む
  G.base = loadGraphic("graphic.png");
  -- 元絵からキャラを切り抜く
  G.chara_hoge = cutGraphic(G.base,0,0,50,50);
  -- 当たり判定をつけてできあがり
  addGraphicHit(G.chara_hoge,0,0,50,50);

  --元絵からさらにキャラを切る
  G.chara_fuga = cutGraphic(G.base,50,100,100);
  addGraphicHit(G.chara_fuga,0,0,100,100);
  --基準座標を変える
  setGraphicCenter(G.chara_fuga,50,100);
end
```

◆ addGraphicHit

チップに当たり判定矩形を追加する

■ 構文

```
addGraphicHit(hChip,x,y,w,h);
```

■ 引数

hChip	対象のチップハンドル。
x,y	当たり判定矩形の左上座標。
w,h	矩形の幅と高さ。

■ 戻り値

なし

■ 解説

- 指定したチップに当たり判定矩形を追加する。当たり判定矩形は 1 チップ当たり最大 16 個まで登録可能。

◆ cloneGraphic

チップを複製する

■ 構文

```
hChip = cloneGraphic(hChipSrc);
```

■ 引数

hChipSrc	複製元のチップハンドル。

■ 戻り値

hChip	複製したチップのハンドル。

■ 解説

- チップを複製する。呼んだ時点でのチップの設定がまるごとコピーされる。
- テクスチャは元のテクスチャを共用するので、fillGraphic などテクスチャそのものに影響を与える関数を使う場合は注意すること。

◆ createBlankGraphic

空のテクスチャを作成する

■ 構文

```
hChip = createBlankGraphic(w,h);
```

■ 引数

w,h	新しいテクスチャの幅と高さ。

■ 戻り値

hChip	作成したテクスチャのチップハンドル。

■ 解説

- 空のテクスチャを作成し、そのハンドルを得る。
- drawTextToGraphic などの TrueType 描画関数で描画先のバッファとして使うのが主用途。
- 幅と高さは 1 ～ 2048 で指定。DirectX のテクスチャに存在する 2 の累乗制限については内部で自動的に処理するので、特に制限を気にする必要はない。

◆ createRenderTarget

レンダリングターゲットにできる空のテクスチャを作成する

■ 構文

```
hRenderTargetChip = createRenderTarget(w,h);
```

■ 引数

w,h	作成するレンダーターゲットの幅と高さ。

■ 戻り値

hRenderTargetChip	作成したレンダーターゲットのチップハンドル。

■ 解説

- 空のテクスチャを作成するのは createBlankGraphic と同じだが、こちらにはレンダーターゲットとして renderSceneGroupToGraphic 関数でシーンを丸ごとレンダリングすることができる。

◆ cutGraphic

チップの一部を切り出し新しいチップを作成する

■ 構文

```
hChip = cutGraphic(hChipSrc,x,y,w,h);
```

■ 引数

hChipSrc	切り出す元となるチップのハンドル。
x,y	切り出し矩形の左上座標。
w,h	切り出し矩形の幅と高さ。

■ 戻り値

hChip	切り出したチップのハンドル。

■ 解説

- チップの一部を切り出し、新しいチップとして登録しそのハンドルを返す。
- テクスチャ自体は元のチップと共用しているので、1 枚のテクスチャからいくら切り出しを行ってもメモリ消費は最小限に抑えられる。
- 切り出し座標は常に「チップの元テクスチャの左上隅を原点とする」ことに注意。

◆ deleteAllGraphic

テーブル内のチップをすべて削除する

■ 構文

```
deleteAllGraphic(table);
```

■ 引数

table	削除対象のチップハンドルが収まったテーブル。

■ 戻り値

なし

■ 解説

- common.lua で定義されたサービス関数。
- 指定したテーブルに入っている値をすべてチップハンドルと見なし、すべてのチップを削除する。テーブル内テーブルもすべて再帰的に調べて削除する。
- チップハンドルはこの関数で一括削除ができるので、チップハンドルはできるだけテーブルの中に保存するようにコーディングすることを推奨している。

◆ deleteGraphic

チップを削除する

■ 構文

```
deleteGraphic(hChip);
```

■ 引数

hChip	削除するチップのハンドル。

■ 戻り値

なし

■ 解説

- チップを削除し、メモリを解放する。
- 複数のチップがテクスチャを共用している場合は、そのテクスチャを使っているすべてのチップが削除されるまでテクスチャはメモリに残る。チップの削除順は問われない。
- AIMS には自動的にチップを削除する機構はない（終了時のクリーンアップは除く）ので、不要になったチップはスクリプトの責任で削除すること。

◆ fillGraphic

テクスチャを指定色で塗りつぶす

■ 構文

```
fillGraphic(hChip,r,g,b,a);
```

■ 引数

hChip	塗りつぶすテクスチャ（を参照しているチップ）のハンドル。
r,g,b,a	塗りつぶし色。

■ 戻り値

なし

■ 解説

- 指定したチップの元テクスチャを塗りつぶす。
- チップの切り出し範囲に関わらず、元テクスチャの全領域が塗りつぶされる。
- α を指定した場合は元の画像が残るのではなく、半透明の色で塗りつぶしとなる。

◆ getGraphicCenterX

チップの拡縮基準点 X 座標を得る

■ 構文

```
x = getGraphicCenterX(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

x	拡大縮小基準点の X 座標。

■ 解説

- 指定したチップを拡大縮小するときの、基準となる X 座標を得る。

◆ getGraphicCetnerY

チップの拡縮基準点 Y 座標を得る

■ 構文

```
y = getGraphicCenterY(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

y	拡大縮小基準点の Y 座標。

■ 解説

- 指定したチップを拡大縮小するときの、基準となる X 座標を得る。

◆ getGraphicClipRect

チップの切り出し矩形を得る

■ 構文

```
x,y,w,h = getGraphicClipRect(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

x,y	切り出し矩形の左上座標。
w,h	矩形の幅と高さ。

■ 解説

- 指定したチップの切り出し矩形を得る。矩形の座標の基準はテクスチャの左上隅。

◆ getGraphicRCenterX

チップの回転基準 X 座標を得る

■ 構文

```
x = getGraphicRCenterX(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

x	回転基準点の X 座標。

■ 解説

- チップの回転時の中心点の X 座標を得る。

◆ getGraphicRCenterY

チップの回転基準 Y 座標を得る

■ 構文

```
y = getGraphicRCenterY(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

y	回転基準点の X 座標。

■ 解説

- チップの回転時の中心点の Y 座標を得る。

◆ loadGraphic

画像ファイルを読み込みチップを定義する

■ 構文

```
hChip = loadGraphic("filename"[,r,g,b]);
```

■ 引数

filename	読み込む画像ファイルのパス。
r,g,b	カラーキーの色指定。

■ 戻り値

hChip	読み込んだ画像のチップハンドル。

■ 解説

- 外部の画像ファイルを読み込み、画像全体を 1 枚のチップとして登録しそのハンドルを返す。
- 画像ファイルの形式としては BMP,JPG,TGA,DDS,PNG が使用可能。アルファチャンネルを持った画像ならばアルファチャンネルごと読み込むことができる。
- 画像サイズの制限は VGA の制限による。ほとんどの場合縦、横それぞれ最大 2048 ピクセルまで。それ以外の制限については、AIMS 内部で自動的に適応させて管理するので気にする必要はない。
- r,g,b を指定すると、その色が透明色となり、指定した色の部分だけ描画されなくなる。省略した場合は画像をそのまま使用する。

◆ resetGraphicClip

チップの切り出し矩形をリセットし元画像全体とする

■ 構文

```
resetGraphicClip(hChip);
```

■ 引数

hChip	対象のチップハンドル。

■ 戻り値

なし

■ 解説

- チップの切り出し範囲を元テクスチャの全領域にする。それ以外の拡大縮小・回転の基準座標などはリセットされない。

◆ setGraphicAnime

チップにアニメーション情報を設定する

■ 構文

```
setGraphicAnime(hChip,hNextChip,delay);
```

■ 引数

hChip	設定対象のチップハンドル。
hNextChip	切り替え先のチップハンドル。切り替えしないなら負数。
delay	切り替えまでの時間。

■ 戻り値

なし

■ 解説

- 指定チップにアニメーション情報を設定する。
- hChip を表示すると、指定時間後に hNextChip で指定されたチップに自動的に切り替わる。切り替わったあとはそのチップのアニメーション情報に基づいてさらに別のチップへ……と数珠つなぎ式にチップを切り替えて一連のアニメーションを実現する。
- 定型的なアニメーション定義を簡単に行うために別の関数が用意されている。setGraphicAnimeSequence および setGraphicAnimeLoop を参照。

◆ setGraphicAnimeLoop

テーブル内のチップに繰り返しアニメを設定する

■ 構文

```
hFirstChip = setGraphicAnimeLoop(tableChips,delay,looppoint);
```

■ 引数

tableChips	インデックス [1] から順に定義された一連のチップハンドルのテーブル。
delay	各チップの切り替え間隔。
looppoint	アニメが末尾に到達したときの戻り先。テーブルのインデックスで指定。

■ 戻り値

hFirstChip	一連のアニメの最初のチップ。tableChips[1] の値に等しい。

■ 解説

- common.lua で定義されたサービス関数。
- 指定テーブルに定義してあるチップについて、インデックス 1 から末尾まで進み、その後 looppoint で示されたインデックスに戻って繰り返すようにアニメを設定する。
- たとえば、5 個のチップからなるテーブルを与え、looppoint に 3 を指定した場合、チップのアニメは「1,2,3,4,5,3,4,5,3,4,5……」という風にループする。

◆ setGraphicAnimeSequence

テーブル内のチップに連続したアニメを設定する

■ 構文

```
hFirstChip = setGraphicAnimeSequence(tableChips,delay);
```

■ 引数

tableChips	インデックス [1] から順に定義された一連のチップハンドルのテーブル。
delay	各チップの切り替え間隔。

■ 戻り値

hFirstChip	一連のアニメの最初のチップ。tableChips[1] の値に等しい。

■ 解説

- common.lua で定義されたサービス関数。
- 指定テーブルに定義してあるチップについて、インデックス 1 から末尾まで連続的にアニメするように、各チップのアニメ情報を設定する。

◆ setGraphicCenter

チップの回転・拡縮基準点を設定する

■ 構文

```
setGraphicCenter(hChip,x,y[,type])
```

■ 引数

hChip	対象のチップハンドル。
x,y	中心座標。
type	何の中心を設定するか指定。（別表）

■ 戻り値

なし

■ 解説

- 指定チップの回転、または拡縮の中心座標を指定する。type を省略した場合は回転・拡縮の基準を同じに設定する。ここで指定する座標はチップの左上を原点とする。

◆ setGraphicClip

チップの切り出し矩形を再設定する

■ 構文

```
setGraphicClip(hChip,x,y,w,h);
```

■ 引数

hChip	対象のチップハンドル。
x,y	切り出し矩形の左上座標。
w,h	切り出し矩形の幅と高さ。

■ 戻り値

なし

■ 解説

- チップの切り出し矩形を再設定する。
- 各種基準座標、当たり判定などには影響を与えない。

◆ renderSceneGroupToGraphic

シーングループをテクスチャにレンダリングする

■ 構文

```
renderSceneGroupToGraphic(hRenderTargetChip,groupNo[,r,g,b,a]);
```

■ 引数

hRenderTargetChip	レンダリング先のレンダーターゲットチップのハンドル。
groupNo	レンダリングするシーングループの番号。0 ～ 3。
r,g,b,a	レンダーターゲットの消去色。描画前に行われる。省略時は内容を残したままさらにレンダリングする。

■ 戻り値

なし

■ 解説

- 指定したシーングループの内容を指定したレンダーターゲットテクスチャに書き出す。シーンのスクリーンショットを撮影し、それをテクスチャに保存する、と考えるとわかりやすいかもしれない。
- 描画範囲は hRednerTargetChip の切り出し範囲となる。cutGraphic をうまく使うことで 1 枚のレンダーターゲットに複数のシーンをレンダリングすることもできる。
- レンダリングした内容は画面モード切替などのデバイスロスト時に失われる。backupRenderTarget 関数を使うその瞬間のレンダーターゲットの内容をバックアップできる。バックアップされた画像はデバイスロスト後のテクスチャ復帰に使われる。

■ 例

```
-- 大本のレンダーターゲット
G = createRenderTarget(1024,1024);
G_part = {
  -- そのうち左上640x480を切り取る
  [1] = cutGraphic(G,0,0,640,480);
  -- [1]のチップの下の320x240の領域を切り取る
  [2] = cutGraphic(G,0,480,320,240);
};
-- シーングループ0番をG_part[1]のチップに書き出す。背景は透明
renderSceneGroupToGraphic(G_part[1],0,0,0,0);
-- シーングループ0番をG_part[2]のチップに書き出す。背景は赤
renderSceneGroupToGraphic(G_part[2],255,0,0,255);
```

◆ backupRenderTarget

デバイスロスト時のため、現在のレンダーターゲットをバックアップする

■ 構文

```
backupRenderTarget(hRenderTargetChip);
```

■ 引数

hRenderTargetChip　レンダーターゲットを持つチップのハンドル。

■ 戻り値

なし

■ 解説

- デバイスロスト時に使うテクスチャ復帰用の画像として、現在のレンダーターゲットの内容をバックアップする。
- レンダーターゲットごとにバックアップのバッファ一つが用意されている。
- DirectX では画面モード切替時などにすべての画像リソースなどが失われることがある（デバイスロスト）。AIMS ではファイルから読み込んだ画像はすべて自動的に復活させるが、レンダーターゲットの内容はこの命令で一時保存しておかないと復活されない。
- 毎フレームレンダーターゲットを書き換えるような場合は、バックアップをとる必要はない。バックアップをとらずとも、毎フレームの書き換えがあるならテクスチャは正常に戻るし、この関数はかなり時間を消費するので、毎フレームバックアップをとるのは現実的でない。
- 一方、一度レンダリングした画像を長期間使い回すような場合には途中でのデバイスロストに備え、この関数でバックアップをとるべきである。

1-5 ムービーテクスチャを扱う関数

◆ loadMovie

動画ファイルからチップを作成する

■ 構文

```
hMovieChip = loadMovie("filename");
```

■ 引数

filename	読み込む動画ファイル名。

■ 戻り値

hMovieChip	作成したムービーチップのファイル名。

■ 解説

- 動画ファイルからチップを作成し、そのハンドルを得る。playMovie 関数にこのハンドルを与えると動画の再生を開始する。ムービーチップも通常のチップ同様、アクターに割り当てて表示させる。cutGraphic 関数で一部を切り出したり、当たり判定を割り当てるなどの操作も通常のチップと同様に可能である。
- 読み込ませることができる動画ファイルは DirectShow で読めるもの。コーデックがなければ読み込みできない。MPEG 形式を選ぶのがもっとも無難と思われる。
- ハンドルの解放は deleteGraphic を呼べばよい。

◆ playMovie

ムービーを再生する

■ 構文

```
playMovie(hMovieChip);
```

■ 引数

hMovieChip　　対象のムービーチップハンドル。

■ 戻り値

なし

■ 解説

- 指定したハンドルのムービーを再生する。pauseMovie で一時停止されていたムービーなら、一時停止した箇所から再開する。

◆ pauseMovie

ムービーを一時停止する

■ 構文

```
pauseMovie(hMovieChip);
```

■ 引数

hMovieChip　　対象のムービーチップハンドル。

■ 戻り値

なし

■ 解説

- 再生中のムービーを一時停止する。再開には playMovie を呼ぶ。

◆ stopMovie

ムービーを停止する

■ 構文

```
stopMovie(hMovieChip)
```

■ 引数

hMovieChip	対象のムービーチップハンドル。

■ 戻り値

なし

■ 解説

- ムービーを停止する。次に playMovie を呼んだ場合先頭からの再生になる。

◆ getMoviePosition

指定したムービーの再生位置を得る

■ 構文

```
pos = getMoviePosition(hMovieChip);
```

■ 引数

hMovieChip	対象のムービーチップハンドル。

■ 戻り値

pos	ムービーの再生位置。単位ミリ秒。

■ 解説

- 指定したムービーの再生位置をミリ秒単位で返す。

◆ isMoviePlaying

指定したムービーが再生中かを得る

■ 構文

```
state = isMoviePlaying(hMovieChip);
```

■ 引数

hMovieChip	対象のムービーチップハンドル。

■ 戻り値

state	再生中なら true、停止中なら false。

■ 解説

- 指定したムービーが再生中かどうかを返す。

1-6 文字列描画を扱う関数

◆ 文字列描画のまとめ

AIMS で文字列を描画する方法は主に 3 つあります。

- TrueType フォントを用い、テクスチャに文字列を描画して、それをアクターで表示する
- 所定の並べ方に従い等幅の英字フォントを描き、createAsciiFont で読み込み、createTextActor でアクターとして表示する。(ASCII フォント)
- TrueType フォントからテクスチャフォントを事前に作成し、createTextureFont で読み込み、createTextActor でアクターとして表示する。

まず最初の TrueType を用いる方法ですが、createTextFont 関数で TrueType フォントを設定し、createBlankGraphic 関数で空のテクスチャを作成、そこに対して drawTextToGraphic 関数で文字列を書くという流れになります。

この方法だと表示できる文字はほとんど制限がありませんが、drawTextToGraphic 関数の動作速度があまり速くなく（これはテクスチャに描画する処理の都合上仕方がありません)、リアルタイムに変化するような文字列には向いていません。また、ユーザーの環境にないフォントは使えませんから、表現という点ではかなりの制限を受けます。

スコアのように逐次変化する文字列を扱うなら残り 2 つの方法が速度も速く便利です。

ASCII フォントはグラフィックツールで作成できますが、その名の通り 1 バイト文字しか扱えません。

テクスチャフォントは専用ツール (AIMS 付属の FontUtil) でなければ作成できない上に、事前生成されてない文字は表示できませんが、2 バイト文字も扱えることに加えて、相手の環境に依存せず好きなフォントが使えるというメリットもあります。また、事前生成については FontUtil がコマンドラインにも対応していますので、これを使うことでほぼ全自動化できます。

◆ createAsciiFont

画像ファイルから ASCII 文字フォントを作成する

■ 構文

```
hTextFont = createAsciiFont("filename",w,h);
```

■ 引数

filename	テクスチャファイル名。
w,h	文字チップのサイズ。

■ 戻り値

hTextFont	作成されたフォントハンドル。

■ 解説

- テクスチャを読み込み、createTextActor で使う新しい ASCII フォントを作成する。
- 文字はすべて同じサイズで、w,h で指定したサイズのチップが等間隔に並んでいる必要がある。
- 文字チップの並べ方も決まっており、右表のように並んでいなければならない。空欄の部分は制御文字などであるが、ここに画像を入れた場合でも文字コードを直接指定すれば表示できる。

表 1-6-1. ASCII フォントの配列

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
	.	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜

◆ createTextFont

TrueType フォントからフォントハンドルを作成する

■ 構文

```
hTTFont = createTextFont("name",size[,bold[,italic]]);
```

■ 引数

name	TrueTrpe フォント名。
size	フォントサイズ。単位ピクセル。
bold	ボールド体を指定するか。true または false。
italic	イタリック体を指定するか。true または false。

■ 戻り値

hTTFont	作成された TrueType フォントのハンドル。

■ 解説

- 指定した名前、サイズの TrueType フォントを登録しハンドルを得る。
- この関数で作成したフォントハンドルは、drawTextToGraphic、および drawAATextToGraphic での描画フォント指定、また startIMEInput での入力領域の表示フォントとして使用される。
- AIMS を動かしているマシンに指定した TrueType フォントがなかった場合は適当なフォントで代替されるが、そもそもすべての環境に間違いなくインストールされているフォント以外を使うべきでない。そのようなフォントを使いたい場合はテクスチャフォントを検討すべきである。

■ 例

```
--MSゴシック、サイズ12ピクセルを登録
hTTFont = createTextFont("ＭＳ ゴシック",12);
```

◆ createTextureFont

テクスチャフォントの読み込みを行う

■ 構文

```
hTextureFont = createTextureFont("lfdfilename","texture"[,isPropotion
al]);
```

■ 引数

lfdfilename	LFD ファイル名。
texture	文字列先頭に "." を付けた場合はテクスチャファイルの拡張子指定。または LFD に対応する LAG ファイルを指定。
isPropotional	プロポーショナルフォントか。true を指定するとプロポーショナルフォントとして扱われる。省略時は false。

■ 戻り値

hTextureFont	作成されたテクスチャフォントのハンドル。

■ 解説

- テクスチャに事前に文字を書き込んでおき、それを並べて表示するテクスチャフォントの読み込みを行う。事前生成が必要であり、生成していない文字は表示できないが、TrueType にくらべて取り扱いが簡単かつ処理速度に優れる。
- lfdfilename はテクスチャフォントの配置データである LFD ファイルを指定。
- texture はフォント作成時の形式にあわせて選ぶ。LFD 生成時、一緒に生成されたテクスチャファイルの拡張子を指定すればよい。
- LAG ファイルを読ませる場合はパス指定ができるが、テクスチャ拡張子を指定する場合は仕様上、EXE ファイルや boot.lua と同じ階層にテクスチャファイルを置く必要がある。
- isPropotional は元フォントにあわせて選ぶ。等幅フォントで true を指定すると不自然に文字間が詰まる、逆にプロポーショナルフォントで false にすると文字間が不自然に開くので、間違えないこと。

- LFD ファイルとテクスチャ、および LAG ファイルの生成は開発キット同梱のツール FontUtil と LagUtil で行える。

■ 例

```
-- 等幅のフォントfont.lfdとそのテクスチャfont_00.pngが自動生成されているとして、
-- これを読み込む例
f = createTextureFont("font.lfd",".png",false);
```

◆ createTextActor

ASCII 文字、またはテクスチャフォントを表示するアクターを生成する

■ 構文

```
hTextActor = createTextActor(hTextFont,"string",x,y,layer[,r,g,b,a[,align[,blend]]][,"classname"]);
```

■ 引数

hTextFont	描画に使うフォントハンドル。createAsciiFont、または createTextureFont で作成したものであること。
string	表示したい文字列。
x,y	初期位置。
layer	生成先レイヤー番号。
r,g,b,a	表示色。すべて 0 ～ 255。
align	文字列の整列方法。定数定義あり。（表 1-6-2）
blend	ブレンドの状態指定。addMover の定数（表 7-1-3）が使える。
classname	アクタークラスの指定。

■ 戻り値

hTextActor	作成されたアクターのハンドル。

■ 解説

- ASCII フォント、またはテクスチャフォントを用いて指定した文字列を表示するアクターを生成する。TrueType フォントは使用できない。
- このアクターにも普通のアクター同様に関数や Mover で制御することができる。ただし、見た目の角度を変化させることはできない。
- align に指定できる定数は common.lua に定義がある。

表 1-6-2.　テキストの文字揃え定数

定数名	内容
ALIGN_LEFT	左寄せ。表示座標にテキストの左上を合わせる。
ALIGN_CENTER	センタリング。表示座標にテキストの中央を合わせる。
ALIGN_RIGHT	右寄せ。表示座標にテキストの右上を合わせる。

◆ deleteAsciiFont

ASCII 文字フォントを削除する

■ 構文

```
deleteAsciiFont(hAsciiFont);
```

■ 引数

hAsciiFont　　対象の ASCII 文字フォントハンドル。

■ 戻り値

なし

■ 解説

- 指定した ASCII 文字フォントを削除する。

◆ deleteTextFont

TrueType フォントハンドルを削除する

■ 構文

```
deleteTextFont(hTTFont)
```

■ 引数

hTTFont	対象の TrueType フォントのハンドル。

■ 戻り値

なし

■ 解説

- 指定した TrueType フォントハンドルを削除する。

◆ deleteTextureFont

テクスチャフォントを削除する

■ 構文

```
deketeTextureFont(hTextureFont);
```

■ 引数

hTextureFont	対象のテクスチャフォントのハンドル。

■ 戻り値

なし

■ 解説

- 指定したテクスチャフォントを削除する。
- フォント周りは 3 通りのフォント生成方法があるが、それぞれ削除の関数は別々になっているので、十分注意が必要。

◆ drawAATextToGraphic

テクスチャに TrueType フォントを用いて文字列を描画する (AA あり)

■ 構文

```
newx = drawAATextToGraphic(hChip,hTTFont,x,y,"string",r,g,b,a,isPropotional);
```

■ 引数

hChip	書き込み先のテクスチャを指しているチップのハンドル。
hTTFont	使用する TrueType フォントのハンドル。
x,y	描画位置左上の X,Y 座標。
string	描画する文字列。
r,g,b,a	描画色。
isPropotional	プロポーショナルフォントか。true か false。

■ 戻り値

newx	描画した文字列の右端の座標。

■ 解説

- hChip のチップが参照しているテクスチャに対して、TrueType フォントを用いてアンチエイリアス (AA) つきの文字を描画する。
- 座標はチップの切り出し矩形の左上基準ではなく、テクスチャ全体の左上端が基準となる。
- プロポーショナルフォントかどうかを指定するフラグは、元のフォントに併せて適切に設定されなければならない。
- 若干重い処理であり、特に長いテキストを描画するとフレーム落ちの原因になりうる。

◆ drawTextToGraphic

テクスチャに TrueType フォントを用いて文字列を描画する (AA なし)

■ 構文

```
newx = drawTextToGraphic(hChip,hTTFont,x,y,"string",r,g,b,a,isPropotional);
```

■ 引数

hChip	書き込み先のテクスチャを指しているチップのハンドル。
hTTFont	使用する TrueType フォントのハンドル。
x,y	描画位置左上の X,Y 座標。
string	描画する文字列。
r,g,b,a	描画色。
isPropotional	プロポーショナルフォントか。true か false。

■ 戻り値

newx	描画した文字列の右端の座標。

■ 解説

- hChip のチップが参照しているテクスチャに対して、TrueType フォントを用いての文字を描画する。こちらはアンチエイリアスを使わない。
- drawAATextToGraphic と引数や戻り値は変わらない。

◆ getActorString

文字列アクターの表示文字列を取得する

■ 構文

```
string = getActorString([hActor]);
```

■ 引数

hActor	対象のアクター。省略時自分自身。

■ 戻り値

string	そのアクターが保持している文字列。

■ 解説

- 文字フォント表示機能を持ったアクターが今表示している文字列を返す。
- ただし、文字列の保持機能はすべてのアクターが保有しているので、普通のアクターにこの関数を使っても正しく動作する。(setActorString も同様。)その場合文字列データはアクターの動作に影響しないので、スクリプトで適当なテキストデータを保存するなどの用途に使うことができる。

◆ getTextWidth

指定フォントで描画した場合の文字列幅を計算する

■ 構文

```
w = getTextWidth(hFont,"string",isPropotional);
```

■ 引数

hFont	テクスチャフォントまたは TrueType フォントのハンドル。
string	横幅を計算する文字列。
isPropotional	プロポーショナルフォントか。true か false。

■ 戻り値

w　　string の横幅。

■ 解説

- フォント (テクスチャフォントか TrueType フォント) を用いて文字列を描画すると仮定して、その横幅がいくらになるかを返す関数である。
- hfont にはテクスチャフォントか TrueType フォントのハンドルが指定できる。ASCII フォントは指定できないが、「チップの幅×文字数」で簡単に横幅が出るのでこの関数では処理しない。

◆ getTextureFontHeight

指定テクスチャフォントの文字高さを取得する

■ 構文

```
h = getTextureFontHeight(hFont);
```

■ 引数

hFont　　テクスチャフォントのハンドル。

■ 戻り値

h　　hFont で示すフォントの高さ。

■ 解説

- 指定したテクスチャフォントの文字高を返す。TrueType フォント、ASCII フォントは指定できない。

◆ setActorString

文字列アクターの表示文字列を設定する

■ 構文

```
setActorString([hActor,]"string");
```

■ 引数

hActor	対象のアクター。省略時自分自身。
string	新しい表示文字列。

■ 戻り値

なし

■ 解説

- 文字フォント表示機能を持ったアクターの、表示文字列を変更する。
- 通常のアクターでも文字列の設定は可能であるが、アクターの動作には影響しない。設定した文字列は getActorString で取得でき、スクリプトから自由に利用することができる。

◆ setTextActorAlign

テキストアクターの文字揃えを指定する

■ 構文

```
setTextActorAlign([hTextActor,]align);
```

■ 引数

hTextActor	対象の文字列アクター。省略時自分自身。
align	文字整列の方法。定数定義あり。

■ 戻り値

なし

■ 解説

- 文字列表示を行うアクターについて、その文字揃えの方法を変更する。
- hTextActor に通常のアクターを指定するとエラーとなる。

1-7 効果音・BGM を扱う関数

◆ 効果音・BGM のまとめ

AIMS には PCM 形式の WAV ファイルを読み込み効果音として再生する機能と、Ogg Vorbis 形式の ogg ファイルを BGM としてストリーミング再生する機能があります。

BGM は playMusic/playMusicLoop で再生、stopMusic で停止します。同時に 1 本のストリームしか再生できません。停止以外にも、fadeMusic でフェードアウトさせることができます。

効果音は loadSound で一度メモリに読み込んだあと、playSound で再生します。

WAV ファイルは一般的な PCM 形式でさえあればサンプリング周波数やステレオ／モノラルはほとんど制限がありませんが、ogg ストリームの方は 44KHz のステレオ形式以外は正しく再生できません。お手数ですが、事前に 44KHz ステレオに変換しておいてください。ogg ファイルのエンコードについては Web に多くのフリーソフトウェアがありますので、google あたりで検索してみるとよいでしょう。

◆ loadSound

サウンドファイルを読み込み効果音のハンドルを得る

■ 構文

```
hSound = loadSound("filename");
```

■ 引数

filename	読み込む WAV ファイル名。

■ 戻り値

hSound	読み込まれた WAV ファイルのサウンドハンドル。

■ 解説

- WAV ファイルを読み込み効果音ハンドルを返す。

◆ playSound

効果音を再生する

■ 構文

```
playSound(hSound[,volume[,pan]]);
```

■ 引数

hSound	再生するサウンドハンドル。
volume	再生音量。
pan	再生時のパン（左右の定位）。

■ 戻り値

なし

■ 解説

- loadSound で読み込んだ効果音を再生する。
- volume には再生音量を 0 ～ 100 で指定。100 のとき setSoundVolume で指定した音量、0 で無音。
- pan は左右定位。-100（左）～ 0（中央）～ 100（右）の範囲で指定。
- それぞれ、省略すると音量は 100、パンは 0 を指定したものとされる。

■ 例

```
local s = loadSound("se/sound.wav");

playSound(s);  -- 元音量、定位中央
playSound(s,100,-100) -- 元音量、左いっぱい
playSound(s,0) -- 音量0 = 無音
```

◆ deleteSound

効果音を削除する

■ 構文

```
deleteSound(hSound);
```

■ 引数

hSound	対象のサウンドハンドル。

■ 戻り値

なし

■ 解説

- 指定したハンドルの効果音をメモリから解放する。

◆ isSoundPlaying

指定した効果音が再生中か返す

■ 構文

```
state = isSoundPlaying(hSound);
```

■ 引数

hSound	対象のサウンドハンドル。

■ 戻り値

state	再生中なら true。そうでなければ false。

■ 解説

- 指定したサウンドハンドルの再生状態を返す。

◆ setGlobalMasterVolume

AIMS 全体の音量 (マスター音量) を設定する

■ 構文

```
setGlobalMasterVolume(volume);
```

■ 引数

volume	音量。0 ～ 100。

■ 戻り値

なし

■ 解説

- AIMS の全体の音量を設定する。BGM、効果音の両方に影響する。

◆ setMusicMasterVolume

音楽の音量を設定する

■ 構文

```
setMusicMasterVolume(volume);
```

■ 引数

volume	音量。0 ～ 100。

■ 戻り値

なし

■ 解説

- 音楽（ogg ストリーム）の音量を設定する。

◆ setSoundMasterVolume

効果音全体の音量を設定する

■ 構文

```
setSoundMasterVolume(volume);
```

■ 引数

volume	音量。0 ～ 100。

■ 戻り値

なし

■ 解説

- 効果音のマスター音量を設定する。0 にすれば完全ミュートとなる。

◆ setSoundVolume

個別の効果音の音量を設定する

■ 構文

```
setSoundVolume(hSound,volume);
```

■ 引数

hSound	対象のサウンドハンドル。
volume	音量。0 ～ 100。

■ 戻り値

なし

■ 解説

- 指定した効果音の音量を設定する。
- setSoundVolume で効果音ごとの音量バランスを調整しておき、playSound の音量指定でゲーム中の音量変化を行うとよい。

■ 例

```
local s = loadSound("se/sound.wav");
setSoundVolume(s,50);
playSound(s,50);  -- 50%のさらに50% → 元のレベルの25%の音量で再生される
```

◆ stopSound

効果音を止める

■ 構文

```
stopSound(hSound);
```

■ 引数

hSound	対象のサウンドハンドル。

■ 戻り値

なし

■ 解説

- 指定した効果音の再生を強制的に止める。

◆ playMusic

音楽を再生する（ループなし）

■ 構文

```
playMusic("filename");
```

■ 引数

filename	再生する ogg ファイル名。

■ 戻り値

なし

■ 解説

- 指定した ogg ファイルを 1 回再生する。なお、ogg ファイルのフォーマットは必ず「44kHz、ステレオ」としておくこと。これ以外のフォーマットでは正しく再生されない。

◆ playMusicLoop

音楽を再生する（ループあり）

■ 構文

```
playMusicLoop(["intro_filename",]"loop_filename");
```

■ 引数

`intro_filename`　イントロ部分の ogg ファイル名。

`loop_filename`　ループ部分の ogg ファイル名。

■ 戻り値

なし

■ 解説

- playMusic 同様に ogg ファイルを再生するが、ストリームの末尾に来た場合先頭から再び再生を行うところが異なる。
- 二つファイルを指定した場合は、最初のファイルをイントロ、後のファイルをループ部分と見なし、「イントロ」→「ループ」→「ループ」……とイントロ部分を一度再生した後ループに入る。

◆ playMod

MOD ファイルを再生する (プラグイン必須)

■ 構文

```
playMod("filename");
```

■ 引数

filename　　再生する MOD ファイル名。

■ 戻り値

なし

■ 解説

- MOD ファイルを読み込み、再生する。
- MOD,XM,IT,S3M をはじめとした MODPlug Player が再生可能な音楽ファイルを再生できる。
- あらかじめアプリケーションコンフィグで USE_MODPLUG = 1 を指定しておかなければならない。
- この関数で再生した場合は曲のループ設定を無視する。
- 曲を止める場合は stopMusic、fadeMusic などの関数が使える。

◆ playModLoop

MOD ファイルをループ再生する (プラグイン必須)

■ 構文

```
playModLoop("filename");
```

■ 引数

filename　　再生する MOD ファイル名。

■ 戻り値

なし

■ 解説

- MOD ファイルを読み込み、ループ再生する。
- あらかじめアプリケーションコンフィグで USE_MODPLUG = 1 を指定しておかなければならない。
- ループ設定は各モジュールファイルでの設定が使われる。

◆ fadeMusic

音楽をフェードアウトさせる

■ 構文

```
fadeMusic(time);
```

■ 引数

time	フェードアウトにかける時間。単位フレーム。

■ 戻り値

なし

■ 解説

- 再生中の ogg ストリームを time で指定したフレーム数だけかけてフェードアウトさせる。

◆ stopMusic

音楽を止める

■ 構文

```
stopMusic();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- 再生中の ogg ストリームを止める。

1-8 入力を制御する関数

◆ 入力制御のまとめ

AIMS では論理ボタンという概念を導入しています。パッドボタンやキーボードの入力を直接取り扱うのではなく、一度それらの入力を「論理ボタン」という仮想のボタンにマッピングし、すべての入力をその「論理ボタン」を通して扱いましょう、という考え方です。

論理ボタンを表す定数は common.lua に定義されています。getJoyPressCount 関数などで論理ボタンを指定するときはこれらの定数が使えます。詳しくは getJoyPressCount 関数の説明をごらんください。

物理ボタン (キーボード､パッド) と論理ボタンの割り付けの初期状態はアプリケーションコンフィグで定義しますが、パッドボタンと論理ボタンの割り付けに限っては setJoyBind 関数で任意のタイミングで変更することができます。

マウスに関してはこれらとは別に管理されており、左、右、中央それぞれを直接指定するようになっています。同様にボタンに対応する定数があり、こちらは getMousePressCount 関数でボタンを指定するのに使えます。

◆ getJoyPressCount

論理ボタンが「押され始めてから経過した時間」を得る

■ 構文

```
count = getJoyPressCount([player,]buttonID);
```

■ 引数

player　　プレイヤー番号（パッド番号）。省略時は 0（＝プレイヤー 1）になる。

buttonID　　論理ボタン ID。

■ 戻り値

count　　そのボタンが押され始めてから経過した時間。押されていないならば 0。

■ 解説

- 指定した論理ボタンが押され始めてからの時間をフレーム数で返す。
- 論理ボタン指定には定数が使える。定義は表 1-8-1 の通り。

表 1-8-1．common.lua にある論理ボタン定数

定数名	ボタン割り当て
BUTTON_RIGHT	方向入力 右
BUTTON_DOWN	方向入力 下
BUTTON_LEFT	方向入力 左
BUTTON_UP	方向入力 上
BUTTON_TRIG1	論理ボタン 1
BUTTON_TRIG2	論理ボタン 2
……以下BUTTON_TRIG12まで	論理ボタン 3 ～ 12

◆ clearJoyPressCount

ジョイスティックの入力カウンタをクリアする

■ 構文

```
cleaJoyPressCount();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- getJoyPressCount で取得できるジョイスティックの入力カウンタの値を強制的に 0 にリセットする。

◆ getJoyBind

ジョイスティックの論理ボタン→物理ボタン対応状態を取得する

■ 構文

```
pButtonNo = getJoyBind([player,]lButtonID);
```

■ 引数

player	プレイヤー番号（パッド番号）。省略時は 0（＝プレイヤー 1）になる。
lButtonID	論理ボタン ID。定数が定義されている。（別表）

■ 戻り値

pButtonNo	物理ボタン番号。

■ 解説

- 指定したプレイヤーについて、指定した論理ボタンに割り当てられている物理ボタンの番号を返す。
- プレイヤー番号は 0 がプレイヤー 1 を指し、アプリケーションコンフィグで設定したプレイヤー数まで指定できる。
- 論理ボタンの定数について別表参照。論理ボタン ID には方向入力は指定できない。（ジョイスティックでの方向入力は X,Y 軸で固定）
- 戻り値は 0 から順番にパッドのボタンに対応。負数ならば割り当てなしを示す。

◆ getJoyPhysicalButton

今ジョイスティックで押されている物理ボタンを取得する

■ 構文

```
pButtonNo = getJoyPhysicalButton([player]);
```

■ 引数

player	プレイヤー番号（パッド番号）。省略時は 0（＝プレイヤー 1）になる。

■ 戻り値

pButtonNo	物理ボタンが押されていたらそのボタン番号。何も押されていなければ負数。

■ 解説

- ジョイスティックの状態を読み取り、今押されている物理ボタンを一つ取得する。
- 複数のボタンが押されていた場合はもっとも若い番号が返る。

◆ getMousePressCount

マウスボタンの「押され始めてから経過した時間」を得る

■ 構文

```
count = getMousePressCount(buttonID);
```

■ 引数

buttonID　　マウスボタンを示す番号。

■ 戻り値

count　　そのボタンが押され始めてから経過した時間。押されていなければ 0。

■ 解説

- 指定したマウスボタンが押され始めてからの時間をフレーム数で返す。
- マウスボタンの指定には定数が使える。定義は表 1-8-2 の通り。

表 1-8-2.　common.lua にあるマウスボタン定数

定数名	ボタン割り当て
MOUSE_LEFT	左ボタン
MOUSE_RIGHT	右ボタン
MOUSE_CENTER	中央ボタン (ホイールクリック)

◆ getMouseW

マウスホイールの状態を取得する

■ 構文

```
w = getMouseW()
```

■ 引数

なし

■ 戻り値

w	マウスホイールの移動量

■ 解説

- マウスホイールの状態を取得する。
- 戻り値は直前フレームからの変化量。＋で奥方向、－で手前方向に回転したことを示す。
- v1.20 までは累積値（起動直後を 0 として -32768 ～ +32767 の範囲で回転量の累積値を返していた）を返していたが、今後は相対値を返すように変更。

◆ getMouseX

マウスポインタの X 座標を得る

■ 構文

```
x = getMouseX()
```

■ 引数

なし

■ 戻り値

x	マウスポインタ X 座標。

■ 解説

- マウスポインタの X 座標を得る。
- 座標の基準は常にゲーム画面左上。
- ウィンドウモードにおいて、ゲーム画面の外にポインタがある場合には、この関数が返す座標はマイナスや画面外を示すことがあるので、注意が必要。

■ 例

```
-- マウスポインタの場所に追従するアクター
function mousepointer_OnStart()
end

function mousepointer_OnStep()
  setPos(getMouseX(),getMouseY());
end

function mousepointer_OnVanish(cause)
end
```

◆ getMouseY

マウスポインタの Y 座標を得る

■ 構文

```
x = getMouseY()
```

■ 引数

なし

■ 戻り値

y　　マウスポインタ Y 座標。

■ 解説

- マウスポインタの Y 座標を得る。
- 座標の基準は常にゲーム画面左上。
- ウィンドウモードにおいて、ゲーム画面の外にポインタがある場合には、個の関数が返す座標はマイナスや画面外を示すことがあるので、注意が必要。

■ 例

getMouseXを参照。

◆ inkey

キーボードで押されたキーを得る

■ 構文

```
key = inkey();
```

■ 引数

なし

■ 戻り値

key　　キーコード。DirectInput のキーコードに準拠。

■ 解説

- キーボードを読み取り、キーボードで押されたキーを 1 つ返す。何も押されていなければ 0 が返る。キーコードは DirectInput におけるコードに対応している。common.lua に DIK_*** の定数が移植されている。
- キーコードが返ってくるのは「押した瞬間」の 1 フレームだけであり、押し続けている間コードが得られるわけではない。また、同時押しをするといくつかのキーを取りこぼすことがある。

◆ isJoyPressed

論理ボタンが「押されているか否か」を得る

■ 構文

```
state = isJoyPressed([player,]buttonID);
```

■ 引数

player	プレイヤー番号（パッド番号）。省略時は 0（= プレイヤー 1）になる。
buttonID	論理ボタン ID。

■ 戻り値

state	ボタンが押されていたら true、押されていなければ false。

■ 解説

- getJoyPressCount と用法は同じだが、こちらは論理ボタンが押されているかどうかだけを返す。

◆ isMousePressed

マウスボタンが「押されているか否か」を得る

■ 構文

```
state = isMousePressed(buttonID)
```

■ 引数

buttonID	マウスボタンを示す番号。定数定義あり。

■ 戻り値

state	ボタンが押されていたら true、押されていなければ false。

■ 解説

- getMousePressCount と用法は同じだが、こちらはボタンが押されているかどうかだけを返す。

◆ setJoyBind

ジョイスティックの論理ボタンと物理ボタンの対応を設定する

■ 構文

```
setJoyBind([player,]lButtonID,pButtonNo);
```

■ 引数

player	プレイヤー番号（パッド番号）。省略時は 0（＝プレイヤー 1）になる。
lButtonID	論理ボタン ID。定数が定義されている。（別表）
pButtonNo	物理ボタン番号。

■ 戻り値

なし

■ 解説

- 指定したプレイヤーについて、指定した論理ボタンに物理ボタンを割り当てる。
- 割り当てのできるボタンは論理ボタン一つにつき物理ボタン一つに限られる。
- また、方向入力のボタン割り当ては変更できない。（パッド X,Y 軸に固定）
- ボタン割り当ては自動的に保存されないので、saveConfig 関数で明示的に保存を指示しなければならない。

◆ setMouseClippingRect

マウスポインタの移動可能範囲を設定

■ 構文

```
setMouseClippingRect();
setMouseClippingRect(x1,y1,x2,y2);
```

■ 引数

x1,y1	移動可能範囲矩形の左上座標。
x2,y2	移動可能範囲矩形の右下座標。

■ 戻り値

なし

■ 解説

- マウスポインタの移動可能範囲を指定する。引数を全省略した場合はウィンドウ全体にする。
- アプリケーションコンフィグで CLIP_MOUSE_POINTER (→ p.178) を指定しておかなければ、この関数は使用できない（エラーにはならないが、設定が無視される）

1-9 文字入力シーンを扱う関数

◆ 文字入力シーンのまとめ

AIMS では、簡易的ではありますが IME を利用した日本語入力をサポートしています。あくまで簡易的なものであり、長文の入力には適していません。普通のテキストボックスと比べると主に下記のような制限があります。

- カーソルを任意に動かすことができない。文章の途中に文字を挿入したり、文章の途中を削除することができない。
- 複数行の編集ができない。
- カット、コピー、ペーストができない。

上記の通りかなり用途はかなり限定されますが、名前の入力など短文を打ち込むくらいなら十分使えると思います。Windows のダイアログなどを使わないので、フルスクリーンモードでも問題なく使用できます。

また、文字入力中でも他のシーン、アクターの動作は停止しないという特徴がありますが、入力中に論理ボタンに割り当てられたキーを押してしまうと、シーンやアクターがそのキー入力に反応して予想外の動作を行ってしまう可能性もはらんでいます。

文字入力を行っているかどうかは isIMERunnning 関数で調べられますので、文字入力中は仮想ボタン入力に反応させないなどの工夫をスクリプト側で行う必要があります。

◆ startIMEInput

文字入力シーンの動作を開始する

■ 構文

```
startIMEInput(x,y,hFont,maxlen[,string]);
```

■ 引数

x,y	入力中の文字列の位置。（＝カーソルの初期位置）左上座標。
hFont	入力中の文字列表示に使うフォントハンドル。
maxlen	入力可能な最大文字数。単位バイト。
string	最初に表示しておく文字列。省略時は空文字列。

■ 戻り値

なし

■ 解説

- カーソルを座標 (x,y) に表示し、文字列入力シーンの動作を開始する。
- フォントハンドルは createFont で作成する。
- 文字入力シーンは他のすべてのシーンより上に表示される。また、他のシーンも同時に動作するので、文字入力中に処理を止めたい場合などは isIMERunning 関数などで明示的に止める処理をすること。
- 文字列入力シーンはユーザーが Enter キーで入力を確定させるか、stopIMEInput 関数を呼ぶことで停止する。
- すでに文字入力シーンが動作している状態でこの関数を呼ぶとエラーとなる。

■ 例

```
--スレッド内で……
hFont = createTextFont("ＭＳ ゴシック",12); -- IME入力表示用フォント
startIMEInput(100,100,hFont,50); -- (100,100)にカーソルを出す
--入力が終わるまで待つ
while isIMERunning() do
  wait(1); --waitで処理をシステムに戻さないとフリーズする
end
string = getIMEString(); -- 戻り値がstringに入る
_dm(string) -- とりあえずデバッグログに出してみる
```

◆ getIMEString

文字入力シーンによって入力された文字列を取得する

■ 構文

```
string = getIMEString()
```

■ 引数

なし

■ 戻り値

string	ユーザーが入力した文字列 (シフト JIS)

■ 解説

- 文字入力シーンによって入力された文字列を取得する。
- 文字入力が確定されていない間は空文字列が返ってくる。

◆ isIMERunning

文字入力シーンが動作中かどうか返す

■ 構文

```
flag = isIMERunning()
```

■ 引数

なし

■ 戻り値

`flag`　　動作中なら true、動作していなければ false

■ 解説

- 文字入力シーンが動作中（文字入力を受け付けている）かどうかを返す。

◆ stopIMEInput

文字入力シーンを終了させる

■ 構文

```
stopIMEInput()
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- 文字入力を中止させる。
- この関数を呼んだ後の getIMEString 関数の戻り値は空文字列となる。

1-10 スレッドを扱う関数

◆ スレッドのまとめ

AIMS でいうスレッドとは 2 種類あります。

- アクターやシーンに紐付けられ、クラスと並行して動作するもの (実体は Lua のコルーチン)
- メインプログラムと同期せず独立に動作する、もっぱらデータ読み込みのためのスレッド（ローダースレッド）

前者のスレッドは、メインプログラムと別のプロセスで動く一般的な「スレッド」とは違い、あくまでも AIMS システムの中で順番に呼び出される関数の一つです。ただ、スレッドは「関数の途中で実行を中断し、次のフレームではそこから再開する」という処理が可能な点が大きな特徴です。下記にスレッドの例を示します。

```
 1. function thread_OnStart()
 2.   while true do    - -以下を無限ループ
 3.     setSpeed(2.0); - -最初の向きから速度2ピクセル/フレームで前進
 4.     wait(60);      -- 1秒間そのまま
 5.     setSpeed(0);   -- 止まれ
 6.     for i=1,45 do
 7.       turn(2);     -- 2度回転
 8.       wait(1);     -- 1フレーム進める
 9.     end            -- 45回繰り返し＝45フレームかけて90度ターンする
10.   end
11. end
```

関数の途中でスレッドを一時停止させるには wait 関数が用意されています。これは指定フレームの間スレッドを停止させるというものです。スレッドにより、通常のアクタークラスではやや面倒な「時系列に沿った制御」を簡単に実行できるようになります。

一方、ローダースレッドは AIMS のメインルーチンとは別プロセスで動作します。名前の通りファイルロードを行わせるスレッドとして使うことを目的としており、メインルーチンがロード中のアニメーションを動かす裏でファイルを読み込む、といった動作を実現できます。ただ、ローダースレッドでそれ以外の動作をすることは推奨されておらず、特にアクター、シーン、レイヤーを操作するのは絶対禁止です。アクターなどを操作する場合はあくまでローダースレッドの外で行うようにしてください。

◆ startThread

スレッド（コルーチン）を起動する

■ 構文

```
startThread("functionname");
```

■ 引数

functionname　スレッドとして起動する関数名

■ 戻り値

なし

■ 解説

- シーン、およびアクターに対してスレッド（コルーチン）を起動する。
- １つのシーンまたは、１つのアクターに対して１つのスレッドを起動することができる。
- すでに他のスレッドを起動している場合はエラーとなる。

◆ stopThread

動作中のスレッドを停止させる

■ 構文

```
stopThread();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- 現在シーンまたはアクターで動作しているスレッドを強制的に停止させる。

◆ isThreadRunning

スレッドが動作しているかどうかを得る

■ 構文

```
state = isThreadRunning();
```

■ 引数

なし

■ 戻り値

state	現在のシーン／アクターでスレッドが動作していたら true。していなければ false。

■ 解説

- スレッドの動作状態を調べ、動作していたら（まだ関数を抜けていなければ）true、終了していたら false を返す。

◆ startLoaderThread

ローダースレッドを起動する

■ 構文

```
startLoaderThread("functionname");
```

■ 引数

functionname　ローダースレッドとして起動する関数名。

■ 戻り値

なし

■ 解説

- 指定した関数をローダースレッドとして動かす。
- すでにローダースレッドが動作中の場合はエラーを返す。
- ローダースレッドはメインスレッド（AIMS のメインルーチン）とは独立、非同期に動作し、外部から止める手段は設定されていない。
- ローダースレッドは名前の通りリソース類の読み込みを目的として設定されており、変数の設定・ファイルの読み込み以外の処理を行った場合の動作は保証されていない。

◆ isLoaderRunning

ローダースレッドの動作状態を得る

■ 構文

```
state = isLoaderRunning();
```

■ 引数

なし

■ 戻り値

`state`　　ローダースレッドが動いていたら true、終了していたら false。

■ 解説

- ローダースレッドが動作しているかどうかを返す。
- また、この関数を呼んだタイミングで、ローダースレッドがエラー終了していた場合はエラーを出して停止する。ローダースレッド内で発生したエラーはデバッグログに出力される。

◆ wait

指定フレームの間スレッドを止める

■ 構文

```
wait(frame);
```

■ 引数

`frame`　　停止するフレーム数。

■ 戻り値

なし

■ 解説

- frame の間スレッドの実行を止める。指定時間が経過したら wait の次から実行を再開する。
- ローダースレッドには使えない。
- 実体は common.lua で定義されている。

1-11 ファイルを扱う関数

◆ extractFile

パッケージからファイルを抽出して書き出す

■ 構文

```
extractFile("srcfile","destfile");
```

■ 引数

srcfile	抽出するパッケージ内のファイル名。
destfile	書き込み先のファイル名。

■ 戻り値

なし

■ 解説

- パッケージ内のファイル srcfile を、destfile で指定したパスに展開して保存する。
- destfile の基準はセーブファイルのパス。

◆ getFileTimeStamp

ファイルのタイムスタンプを得る

■ 構文

```
created, updated, accessed = getFileTimeStamp("filename");
```

■ 引数

filename	調べるファイルのパス。

■ 戻り値

created	作成時刻。1970 年 1 月 1 日からの経過秒数。以下すべて同じ。
updated	更新時刻。
accessed	最終アクセス時刻。

■ 解説

- ファイルのタイムスタンプを得る。戻り値は Lua のライブラリ関数である os.time や os.date で整形できる。

◆ getFilePathDialog

ファイル選択ダイアログからファイルのフルパスを得る

■ 構文

```
filename = getFilePathDialog()
```

■ 引数

なし

■ 戻り値

filename	ファイルセレクタで選ばれたファイルのフルパス。キャンセルされた場合は nil。

■ 解説

- ファイルダイアログを開き、指定されたファイルのパスを得る。
- デバッグビルド専用。リリース環境では無視される。開発ツールやデバッグ時の利用を想定している。

◆ getSaveFilePath

セーブファイルの保存先を得る

■ 構文

```
path = getSaveFilePath();
```

■ 引数

なし

■ 戻り値

path	アプリケーションコンフィグで指定したセーブファイルの保存先。

■ 解説

- アプリケーションコンフィグで指定したセーブファイルの保存先を得る。環境変数は展開され、末尾は必ず円記号「¥」で終わる。

◆ loadFile

指定したファイルの内容を読み込む

■ 構文

```
string = loadFile("filename");
```

■ 引数

filemname	読み込むファイル名。

■ 戻り値

string	ファイルの中身。ファイルがなかった場合 nil。

■ 解説

- 指定したファイルを読み込み、string に代入する。パッケージファイルに圧縮してあるファイルでも展開して読み込む。
- ファイルの検索順は「セーブファイル保存先」→「パッケージファイル内」の順。どちらにも見つからなければ nil を返す。

◆ loadFileDialog

ファイル選択ダイアログを開き、指定されたファイルを読み込む

■ 構文

```
string = loadFileDialog();
```

■ 引数

なし

■ 戻り値

string	ファイルセレクタで指定したファイルの中身。ファイルセレクタがキャンセルされた場合は nil。

■ 解説

- ファイル選択ダイアログを開き、指定されたファイルを読み込んで string に代入する。
- デバッグビルド専用。リリース環境では無視される。

◆ saveFile

指定したファイルに文字列を保存する

■ 構文

```
result = saveFile("filename","string"[,compressed]);
```

■ 引数

filename	保存先ファイル名。
string	ファイルに書き込む内容の文字列。
compressed	ファイルを圧縮するかどうか。true または false。省略時は true。

■ 戻り値

`result` 保存に成功したら true、失敗したら false。

■ 解説

- filename で指定したファイルに文字列 string を保存する。ファイルはセーブファイルの場所に保存される。
- compressed は保存時圧縮を行うかどうかのフラグ。loadFile で開く時は圧縮を意識せずとも自動的に展開してくれる。

◆ saveFileDialog

ファイル保存ダイアログを開き、指定されたファイルに文字列を保存する

■ 構文

```
result = saveFileDialog("string"[,compessed]);
```

■ 引数

`string` 保存する文字列。

`compressed` ファイルを圧縮するかどうか。true または false。

■ 戻り値

`result` 保存に成功したら true、失敗したら false。

■ 解説

- ファイル選択ダイアログを開き、指定されたファイルに string を保存する。
- デバッグビルド専用。リリース環境では無視される。

◆ createFolder

フォルダを作成する

■ 構文

```
createFolder("path");
```

■ 引数

path	作成するフォルダ。

■ 戻り値

なし

■ 解説

- 指定した名前のフォルダを作成する。
- パスの基準はセーブファイルの保存先。

1-12 特殊描画関数

◆ useOnDrawEvent

アクターの描画を Lua スクリプトで行うかどうか設定する

■ 構文

```
useOnDrawEvent(flag);
```

■ 引数

flag	OnDraw イベントの起動を有効にするなら true。無効にするなら false。

■ 戻り値

なし

■ 解説

- アクターの描画部分を Lua スクリプトで行うように設定する。
- true を指定すると、[アクタークラス名]_OnDraw という関数がアクター描画のタイミングで呼ばれるようになる。ユーザーは後述の drawGraphic 関数などで自由にグラフィックを描画できる。

◆ drawGraphic

チップを描画する

■ 構文

```
drawGraphic(hChip,x,y[,zx,zy[,angle[,r,g,b,a[,blend]]]]);
```

■ 引数

hChip	描画するチップのハンドル。
x,y	描画座標。アクターの位置を基準とした相対座標。
zx,zy	チップの拡大率。省略時は等倍。
angle	チップの回転角度。省略時 0。
r,g,b,a	チップの描画色。省略時は不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- チップを描画するユーザー描画用関数。
- OnDraw イベント以外でこの関数を呼んだ場合の動作は保証されていないので注意。これ以外の draw ～～系関数でも同様である。

◆ drawGraphicList

連続した帯状のポリゴンを描画する

■ 構文

```
drawGraphicList(hChip,points[,r,g,b,a][,blend]);
```

■ 引数

hChip	帯に割り当てるチップ。
points	帯の頂点データ。別途説明
r,g,b,a	描画色。省略で不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- points のデータに従い、連続した帯状のポリゴンを描画する。
- points のフォーマットは 1 つの頂点を 1 つのテーブルとし、それを頂点の数だけインデックス [1] から順に収めた 2 次元のテーブルとなる。テーブルへの値の入れ方は下記例を参照。
- チップは帯のサイズに合わせて伸縮させられる。最初の頂点にチップの下端が、最後の頂点に上端がくるように割り付けられる。

■ 例

```
local points = {
   { 0,0,16,45 }; -- { x,y,幅,向き(その頂点での帯の向き) }の順
   { 100,100,16,45,255,0,0,255 }; -- { x,y,幅,向き,r,g,b,a }、色を変更する
   { 200,200,32,45 }; -- 2点以上の頂点が必要
};

--上記頂点データに基づいて描画
drawGraphicList(hChip,points);
```

◆ drawGraphicRect

チップを任意の矩形に拡大縮小して描画する

■ 構文

```
drawGraphicRect(hChip,x1,y1,x2,y2[,angle[,r,g,b,a[,blend]]]);
```

■ 引数

hChip	描画するチップのハンドル。
x1,y1	描画先の矩形の左上座標。
x2,y2	描画先の矩形の右下座標。
angle	矩形の回転角度。省略時 0。
r,g,b,a	描画色。省略時不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- チップを任意の矩形に拡大または縮小して描画する。縦横比などは一切無視して、矩形にぴったり収まるように描画される。
- 回転時の中心座標はチップで設定されている基準座標を元に決まる。

◆ drawTextureFont

テクスチャフォント、または ASCII フォントを描画する

■ 構文

```
drawTextureFont(hFont,x,y,"string",r,g,b,a[,spacing[,align]]);
```

■ 引数

hFont	描画に使うフォント。テクスチャフォントか ASCII 文字フォントのハンドル。
x,y	文字の左上座標。
string	描画する文字列。
r,g,b,a	描画色。
spacing	文字間隔。負の数で詰め、正の数で空き。省略時 0。
align	文字列の整列方法。定数定義あり。（表 1-6-2）

■ 戻り値

なし

■ 解説

- 指定したフォントを用いて文字列を描く。TrueType フォントは使えない。

◆ drawGraphicTorus

ドーナツ状の多角形を描画する

■ 構文

```
drawGraphicTorus(hChip,x,y,rout,rin,div[,angx,angy,angz
   [,r,g,b,a[,blend]]]);
```

■ 引数

hchip	描画するチップのハンドル。
x,y	描画位置。
rout	外円の半径。
rin	内円の半径。
div	多角形の分割数。整数で 3 以上。
angx,angy,angz	それぞれ x,y,z 各軸の回転角度。省略時 0。
r,g,b,a	描画色。省略時不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- ドーナツ状の多角形を描画する。ドーナツの幅は (rout-rin) となり、rin = 0 のときは内側を塗りつぶした多角形と同等の形状になる。
- div を大きくすると円に近い形が得られる。

◆ drawGraphicRing

リング状の多角形を描画する

■ 構文

```
drawGraphicRing(hChip,x,y,r,height,div[,angx,angy,angz
   [,r,g,b,a[,blend]]]);
```

■ 引数

hchip	描画するチップのハンドル。
x,y	描画位置。
r	円の半径。
height	リングの高さ。
div	多角形の分割数。整数で 3 以上。
angx,angy,angz	それぞれ x,y,z 各軸の回転角度。省略時 0。
r,g,b,a	描画色。省略時不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- リング状の多角形を描画する。Z 方向に伸びた厚さの限りなく薄い筒型であり、angx,angy,angz が全て 0 度の時は見えない。
- div を大きくすると円筒に近い形が得られる。

◆ drawGraphic3DPlane

３軸回転のできる平面を描画する

■ 構文

```
drawGraphic3DPlane(hChip,x,y,[zx,zy,[,angx,angy,angz
   [,r,g,b,a[,blend]]]]);
```

■ 引数

hchip	描画するチップのハンドル。
x,y	描画位置。
zx,zy	チップの拡大率。省略時等倍。
angx,angy,angz	それぞれ x,y,z 各軸の回転角度。省略時 0。
r,g,b,a	描画色。省略時不透明の白。
blend	ブレンドモード指定。省略時はアクターの設定にあわせる。

■ 戻り値

なし

■ 解説

- チップを 3 軸回転させて描画する。angx,angy,angz 全てが 0 のときは、drawGraphic と同じ動作になる。

1-13 キャラクターマップ制御関数

◆ キャラクターマップのまとめ

キャラクターマップとは、AIMS においていわゆる「マップチップ」によるフィールドの表現などを実現するための機構です。マップのパーツを敷き詰めたテクスチャと、それをどのように並べるかを示したマップデータを与えることで、広大なマップでも控えめなメモリ消費量で表現できます。2D ゲームにおいては頻出の表現ですので、AIMS ではビルトインの機能として実装しています。

1 枚のレイヤーにつき 1 つのキャラクターマップが割り当て可能です。12 枚のレイヤーすべてに別々、すなわち別々のマップチップ、マップデータを割り当てることができます。もちろん、マップチップのサイズもレイヤーごとに個別に設定することが可能です。

マップデータの作成にはフリーなマップエディタである platinum(HyperDevice software 作、http://www.hyperdevice.net/) が利用できます。AIMS では platinum でエクスポートできる FMF 形式のマップデータを直接読み込むことができます。また、D.N.A.Softwares でも独自にマップエディタの開発を行っており、こちらでも FMF 形式をエクスポートできます。

FMF ファイルを AIMS でどう扱うかについては loadFMF 関数をごらんください。

基本的にはマップチップを loadGraphic で用意し、マップデータを loadFMF 関数で準備、そのあと mapAllocate 関数でマップチップとマップデータを与えて初期化という形になります。マップのスクロールはレイヤーのスクロールに同期しますので、setLayerScroll を使ってください。

◆ loadFMF

FMF 形式のマップを読み込む

■ 構文

```
table = loadFMF("fmffilename");
```

■ 引数

fmffilename　　読み込むファイル名。

■ 戻り値

table　　ファイルの内容。テーブルになっている。

■ 解説

- FMF 形式のマップファイルを読み込む関数。
- 戻り値のテーブルにはマップ幅、高さなどのデータが収められている。形式は表 1-13-1 の通り。
- マップエディタで使用したマップチップ画像も併せて読み込み、mapAllocate 関数にこのデータを与えて初期化すればマップエディタで描いたとおりのマップが得られる。

表 1-13-1. loadFMF が返すテーブルの中身

キー名	内容
mapWidth	マップの幅
mapHeight	マップの高さ
layerCount	ファイル内のレイヤー数
data[n]	マップデータの配列 (n=0 ～ layerCount-1)

■ 例

```
--FMFファイルを読み込んで、レイヤー0番にそのFMFファイルのレイヤー0を読み込む
local t = loadFMF("hoge.fmf");
mapLoadFromTable(0,t.data[0],t.mapWidth,t.mapHeight);
```

◆ mapAllocate

キャラクターマップの使用を開始する

■ 構文

```
mapAllocate(layer,mapw,maph,hMapChip,chipw,chiph[,transparent[,tab
le]]);
```

■ 引数

layer	設定を行うレイヤーの番号。
mapw,maph	マップの幅、高さ。
hMapChip	マップチップを並べたテクスチャを参照しているチップハンドル。
chipw,chiph	マップチップ１つあたりの幅と高さ。
transparent	透過 (描画しない) チップの指定。省略または負数で指定なし。
table	マップデータを納めたテーブル。省略時はマップをゼロで埋める。

■ 戻り値

なし

■ 解説

- 指定レイヤーの下にキャラクターマップを準備する。
- hMapChip で指定したチップが参照しているテクスチャからマップチップを切り出してマップを構成する。
- table を指定するとマップの初期状態を table の通りに設定する。
- table の形式は左上端のチップから順番に値を収めた一次元配列でなければならない。
- キャラクターマップのデータは Lua のメモリ空間と別に確保される。初期化後に table を変更してもキャラクターマップには影響しない。また、キャラクターマップの読み取りや書き換えが必要な場合はそれぞれ mapRead、mapWrite の各関数を使う。

■ 例

```
--FMFファイルを読み込んで、レイヤー0番をそのFMFファイルのレイヤー0で初期化
local t = loadFMF("hoge.fmf");
-- マップチップ。16*16のチップが敷き詰めてあるとする
local g = loadGraphic("mapchip.png");

mapAllocate(0,t.mapWidth,t,mapHeight,g,16,16,-1,t.data[0]);
```

◆ mapBlockTransfer

キャラクターマップの一部にマップデータを転送

■ 構文

```
mapBlockTransfer(srctable,srcw,srch,destLayer,destx,desty
[,transparent]);
```

■ 引数

srctable	コピーするマップデータを収めたテーブル。一次元配列。
srcw,srch	コピー元マップデータの幅と高さ。
destlayer	コピー先レイヤー。
destx,desty	コピー先左上座標。
transparent	透過（コピー時に元のチップのままにする）チップの番号。省略または負数で指定なし。

■ 戻り値

なし

■ 解説

- 指定レイヤーのキャラクターマップに、srctable で指定するマップを転送する。
- srctable の内容は通常のキャラクターマップ作成時などのテーブルと同様、一次元の配列となっていること。幅と高さを正しく指定しないと転送結果が意図した通りにならない。
- chipTransparent で指定したチップはコピー時に転送されない（コピー先に元からあったチップが残る）。

◆ mapClear

キャラクターマップを消去する

■ 構文

```
mapClear([layer]);
```

■ 引数

layer	対象のレイヤー。省略すると全レイヤーを対象にする。

■ 戻り値

なし

■ 解説

- 指定レイヤーのキャラクターマップを消去する。マップデータのメモリも解放される。

◆ mapGetWrappedPos

キャラクターマップ上の座標を繰り返しモードに基づき丸め処理する

■ 構文

```
nx,ncy = mapGetWrappedPos(layer,x,y);
```

■ 引数

layer	計算の元になるキャラクターマップを抱えているレイヤーの番号。
x,y	マップ上の座標。整数で、範囲は無制限。

■ 戻り値

nx,ny	丸め処理後の座標。

■ 解説

- 指定したキャラクターマップ上の座標について、ループ、範囲外などを考慮した座標を計算して返す。
- mapSetWrapMode の設定によって戻り値は変わる。戻り値の関係は表 1-13-2 に示した。戻り値が負数になっている場合は範囲外を指していると解釈していい。

表 1-13-2. 範囲外処理モードと mapGetWrappedPos の戻り値の関係

範囲外処理の定数	0 未満	0 以上 幅 or 高さ未満	幅 or 高さ以上
OUTSIDE_NONE	負数	入力値そのまま	負数
OUTSIDE_FILLWITHEND	0	入力値そのまま	幅 or 高さ -1
OUTSIDE_WRAP	入力値を幅 or 高さで割った余り	入力値そのまま	入力値を幅 or 高さで割った余り

◆ mapLoadFromTable

テーブルからキャラクターマップを作成する

■ 構文

```
mapLoadFromTable(layer,table,mapw,maph);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
table	マップデータを納めたテーブル。
mapw,maph	読み込むマップの横幅と高さ。

■ 戻り値

なし

■ 解説

- テーブルから指定レイヤーのキャラクターマップを作成する。
- mapAllocate の場合と特に変わりはない。

◆ mapRead

キャラクターマップの値を読み込む

■ 構文

```
value = mapRead(layer,x,y);
```

■ 引数

layer	読み取るキャラクターマップを抱えているレイヤー番号。
x,y	読み取る座標。

■ 戻り値

value	指定座標に書き込まれている値。

■ 解説

- キャラクターマップの指定座標の値を読み込む。

◆ mapSetChipSpacing

キャラクターマップの描画間隔 (チップ 1 つの配置間隔) を指定する

■ 構文

```
mapSetChipSpacing(layer,chipsw,chipsh);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
chipsw,chipsh	それぞれチップの横方向、高さ方向の間隔。

■ 戻り値

なし

■ 解説

- チップ自体のサイズは mapSetChipTexture で指定したサイズになるが、ここで間隔を指定すると、次のように見た目が変化する。
- 間隔＞チップサイズのとき……チップとチップの間に (チップサイズ - 間隔) ピクセルの隙間ができる
- 間隔＜チップサイズのとき……チップの右または下が (間隔 - チップサイズ) ピクセルだけ、隣接チップと重なる
- 間隔とチップサイズを同じ値にすれば隙間もできず、重なりもしない (標準の状態)。

◆ mapSetChipTexture

キャラクターマップに割り当てるテクスチャを指定する

■ 構文

```
mapSetChipTexture(layer,hMapChip,chipw,chiph[,transparent]);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
hMapChip	マップチップを並べたテクスチャを参照しているチップハンドル。
chipw,chiph	チップの横幅と高さ。
transparent	透過 (描画しない) チップの指定。省略または負数で指定なし。

■ 戻り値

なし

■ 解説

- 指定したレイヤーのキャラクターマップに割り当てるテクスチャを指定する。

◆ mapSetColor

キャラクターマップの描画色を変更する

■ 構文

```
mapSetColor(layer,r,g,b[,alpha]);
mapSetColor(layer,alpha);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
r,g,b	描画色の R,G,B 値。0 ～ 255。すべて 255 にすると元のチップの色そのまま。
alpha	マップの透明度。0 ～ 255。省略時は 255。

■ 戻り値

なし

■ 解説

- キャラクターマップの描画色を変更する。

◆ mapSetVisible

キャラクターマップの表示、非表示を切り替える

■ 構文

```
mapSetVisible(layer,flag);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
flag	表示状態。true で表示、false で非表示。

■ 戻り値

なし

■ 解説

- 指定レイヤーのキャラクターマップを表示または非表示にする。非表示にした場合でもメモリには残っているので mapRead、mapWrite などは使える。
- mapAllocate を呼んでキャラクターマップ初期化した時は自動的に表示状態にされる。

◆ mapSetWrapMode

キャラクターマップの範囲外の描画方法を選択する

■ 構文

```
mapSetWrapMode(layer,xwrap,ywrap);
```

■ 引数

layer	対象のキャラクターマップを抱えているレイヤーの番号。
xwrap,ywrap	X 方向、Y 方向それぞれについて範囲外時の処理を選ぶ。定数定義あり。（表 1-13-3）

■ 戻り値

なし

■ 解説

- キャラクターマップの範囲外（マップ端のさらに先）について、どのように処理するかを選ぶ。
- OUTSIDE_NONE 以外を指定した場合、mapGetWrappedPos での戻り値もループに対応して変化する。同関数の説明も参照のこと。

表 1-13-3. マップの範囲外処理定数

定数名	内容
OUTSIDE_NONE	何も描画しない
OUTSIDE_FILLWITHEND	マップの端のチップで埋める
OUTSIDE_WRAP	反対端にループ

◆ mapWrite

キャラクターマップに値を書き込む

■ 構文

```
oldvalue = mapWrite(layer,x,y,value);
```

■ 引数

layer	書き込み先のキャラクターマップを抱えているレイヤーの番号。
x,y	書き込む座標。
value	書き込む値。0 以上の整数。

■ 戻り値

oldvalue	指定座標の書き換え前の値。

■ 解説

- キャラクターマップの指定座標に値を書き込む。

1-14 算術ほかサポート関数

◆ aim

２つのアクター間のなす角を求める

■ 構文

```
angle = aim([actor1,]actor2);
```

■ 引数

actor1	向きを調べる元となるアクター。省略時自分自身。
actor2	相手のアクター。

■ 戻り値

angle	actor1 からみた actor2 の向き。単位は度。

■ 解説

- actor1 から actor2 への角度を返す。

◆ dist

２つのアクターの距離を求める

■ 構文

```
distance = dist([actor1,]actor2);
```

■ 引数

actor1	距離を調べる元となるアクター。省略時自分自身。
actor2	相手のアクター。

■ 戻り値

distance	２つのアクター間の (基準点間の) 距離。単位はピクセル。

■ 解説

- ２つのアクターの距離を返す。

◆ randMT,randfMT

Mersenne Twister による乱数

■ 構文

```
real = randMT();
integer = randMT(max);
integer = randMT(min,max);
real = randfMT(max);
real = randfMT(min,max);
```

■ 引数

max	乱数の最大値。
min	乱数の最小値。

■ 戻り値

real	得られた乱数の値（実数）
integer	得られた乱数の値（整数）

■ 解説

- Mersenne Twister による乱数を生成する。引数の与えかたにより戻り値が変わる。
- 引数なしの場合は 0 以上 1 未満の実数を返す。
- 引数 1 つの場合は最大値 max を指定したものとし、0 以上 max 以下の範囲の整数を返す。
- 引数 2 つの場合は最小値 min と最大値 max を指定したものとし、min 以上 max 以下の範囲の整数を返す。
- randMT 関数の代わりに randfMT 関数を呼ぶと、引数を指定した場合でも実数を返すようになる。この場合は 1 引数なら 0 以上 max 未満、2 引数なら min 以上 max 未満の実数が得られる。
- 乱数系列を操作したい場合は srandMT 関数で行う。

◆ srandMT

乱数系列の初期化

■ 構文

```
srandMT([seed]);
```

■ 引数

seed	系列番号。省略時は現在時刻から決定。

■ 戻り値

なし

■ 解説

- Mersenne Twister による乱数の系列を初期化する。同じ seed の値を与えれば毎回同じ値が同じ順序で得られる。

◆ crc32

与えられた文字列の CRC32 を計算する

■ 構文

```
crc = crc32("string");
```

■ 引数

string	元の文字列。

■ 戻り値

crc	文字列 string の 32 ビット CRC 値。数値。

■ 解説

- 文字列 string について、32 ビットの CRC を求めて返す。

◆ md5fromfile

指定したファイルの MD5 ハッシュ値を返す

■ 構文

```
hash = md5fromfile("filename");
```

■ 引数

filename	対象のファイル名。

■ 戻り値

hash	ハッシュ値。16 進数が並んだ文字列の形で返る。

■ 解説

- 指定したファイルについて MD5 ダイジェストを計算しハッシュ値を返す。

◆ md5fromstring

指定した文字列の MD5 ハッシュ値を返す

■ 構文

```
hash = md5fromstring("string");
```

■ 引数

string	対象の文字列。

■ 戻り値

hash	ハッシュ値。16 進数が並んだ文字列の形で返る。

■ 解説

- 指定した文字列について MD5 ダイジェストを計算しハッシュ値を返す。

◆ serialize

テーブルを文字列に変換する

■ 構文

```
string = serialize(table);
```

■ 引数

table	シリアライズするテーブル。

■ 戻り値

string	シリアライズ後の table の内容。

■ 解説

- テーブルをシリアル化し、文字列を返す。
- 戻り値の文字列を deserialize 関数に与えると table と同じ内容のテーブルが得られる。ただし、テーブルの中に関数があった場合、関数だけは無視されるので注意すること。
- この関数は common.lua で定義されている。

◆ deserialize

シリアライズされた文字列をテーブルに変換する

■ 構文

```
table = deserialize(string);
```

■ 引数

string	serialize 関数でシリアライズした文字列。

■ 戻り値

table	string から変換されたテーブル。

■ 解説

- string を解析してテーブルを返す。string には serialize でテーブルから変換した文字列を与える。
- serialize の実体はテーブルを Lua の構文に従ったテーブル定義に変換する関数であり、deserialize は Lua コードを実行するだけである。
- この関数は common.lua で定義されている。

1-15 システム関連関数

◆ enableCloseButton

ウィンドウの閉じるボタンを有効にする

■ 構文

```
enableCloseButton();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- ウィンドウの閉じるボタンを有効にする。

◆ disableCloseButton

ウィンドウの閉じるボタンを無効にする

■ 構文

```
disableCloseButton();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- ウィンドウの閉じるボタンを無効にする。
- この状態ではウィンドウのシステムメニューの「閉じる」も無効になる。
- ゲーム動作中常に閉じるボタンを無効にするような動作はすべきでない。本当にユーザー操作での終了を抑制したい時のみ無効にし、抑制する必要がなくなったら速やかに閉じるボタンは有効にすべきである。

◆ isWindowed

AIMS がウィンドウモードで動作しているかどうかを得る

■ 構文

```
state = isWindowed();
```

■ 引数

なし

■ 戻り値

`state`　　ウィンドウモードなら true。フルスクリーンモードなら false。

■ 解説

- AIMS がウィンドウモードで動作しているかどうかを返す。

◆ setWindowCaption

ウィンドウのキャプションを変更する

■ 構文

```
setWindowCaption("string");
```

■ 引数

string	キャプション文字列。

■ 戻り値

なし

■ 解説

- AIMS のウィンドウキャプションを変更する。

◆ setWindowMode

ウィンドウモード・フルスクリーンモードを切り替える

■ 構文

```
setWindowMode(mode);
```

■ 引数

mode	設定するモード。ウィンドウモードなら true、フルスクリーンモードなら false。

■ 戻り値

なし

■ 解説

- ウィンドウモードとフルスクリーンモードを切り替える。
- ウィンドウモードの場合はどんな解像度のモードでもほぼ問題なく動作する

が、フルスクリーンモードの場合 VGA やモニタの対応状態によっては切り替えに失敗する場合がある。

◆ quit

AIMS を終了する

■ 構文

```
quit();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- 現在動いているすべてのスクリプトの動作を終了し、AIMS を終了する。
- 終了前にはクリーンアップ関数として "OnVanish" 関数が呼ばれる。

◆ sleep

プログラムを指定時間休止する

■ 構文

```
sleep(msec);
```

■ 引数

msec	休止時間。単位ミリ秒

■ 戻り値

なし

■ 解説

- AIMS 全体を指定時間休止する。
- 実態は Windows API の Sleep 関数へのインタフェイス。

◆ saveConfig

設定を保存する

■ 構文

```
saveConfig();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- AIMS のシステム側で保持されている設定項目（ウィンドウモード ON/OFF、パッドボタンと論理ボタンの割り付け）を保存する。
- この関数を呼ばない限り、上記設定は保存されないので、特にパッドボタン

の割り当てを変更した後は必ずこの関数を呼ぶこと。さもないと次の起動時に割り当てがリセットされてしまう。

◆ prequire

別のスクリプトファイルを読み込む

■ 構文

```
preqire("filename");
```

■ 引数

filename	読み込む Lua スクリプトのファイル名

■ 戻り値

なし

■ 解説

- 外部から Lua ファイルを読み込む。
- Lua には標準の外部モジュール読み込み関数として require 関数があるが、AIMS のパッケージファイルからの読み込みはできないため、パッケージ内のスクリプトファイルでも読み込めること、および動作を単純かつわかりやすいものにするため prequire 関数を用意した。
- prequire は C 言語における #include 文のように振る舞う。すなわち、指定されたスクリプトファイルを読み込み、単純にその場で解釈を行い、実行する。

1-16 デバッグ関数

◆ _DEBUG

デバッグモードかどうかを表す定数

■ 構文

```
flag = _DEBUG
```

■ 引数

なし

■ 戻り値

flag	デバッグビルドでの実行なら true、リリースビルドでの実行なら false。

■ 解説

- 今動作中の環境がデバッグ用かリリース用かを識別する。

◆ _dlocate

オンスクリーンデバッグコンソールの文字表示位置を指定

■ 構文

```
_dlocate(x,y);
_dl(x,y); -- 短縮形
```

■ 引数

x,y	表示位置。

■ 戻り値

なし

■ 解説

- デバッグビルド専用。F11 キーで表示される各種情報表示の右半分にあるデバッグコンソールに文字を表示する際の位置を設定する。x,y は文字単位で、X は 0 ～ 39、Y は 0 ～ 59 の範囲で指定可。次の _dprint 命令で有効。

◆ _dprint

オンスクリーンデバッグコンソールへ出力

■ 構文

```
_dprint("string");
_dp("string"); -- 省略形
```

■ 引数

string	コンソールに出力する文字列。半角英数と記号類に限る

■ 戻り値

なし

■ 解説

- デバッグビルド専用。F11 で開くオンスクリーンデバッグコンソールに文字列を出力。
- 出力文字列は \n で改行。コンソールは 40 文字× 60 行のサイズがあり、改行でコンソールの最下段に到達すると全体が上に 1 行スクロールする。

◆ debugOut

デバッグログへの出力

■ 構文

```
debugOut("string");
_dm("string"); -- 短縮形
```

■ 引数

string	コンソールに出力する文字列。

■ 戻り値

なし

■ 解説

- デバッグビルド専用。デバッグログに指定文字列を書き出す。起動オプションによってはタイトルバーにも出力される。
- デバッグログには「どのソースの何行目の debugOut か」という情報も一緒に書き込まれる。

◆ setLayerDrawHitBox

アクターの当たり判定矩形を描画する設定を行う

■ 構文

```
setLayerDrawHitBox(layer,flag[,r,g,b[,a]]);
```

■ 引数

layer	設定を行うレイヤー。
flag	描画を行うかどうか。true で描画する、false でしない。
r,g,b	当たり判定矩形の描画色。
a	同、α値。

■ 戻り値

なし

■ 解説

- デバッグビルド専用。指定レイヤーのアクターについて、アクターの上に当たり判定矩形を描画する。

1-02 非推奨関数

以下の関数は、AIMS の旧バージョンにて搭載されたものの、その後の機能追加、仕様改訂などにより現バージョンでは使用することが推奨されていない関数です。現バージョンでも互換性のため残してはいますが、新規のプログラムでは使用しないでください。

◆ attach

アクターにサブクラスを設定する

■ 構文

```
attach("subclass");
```

■ 引数

subclass　　割り当てるサブクラス名。

■ 戻り値

なし

■ 解説

- アクターにサブクラスを割り当てる。
- すでにサブクラスが動作している場合、この関数を実行するとエラーになる。
- サブクラスの呼び出しタイミングは、subclass_OnStep, subclass_OnVanish の 2 イベントについてはメインのクラスと同様のタイミング、subclass_OnStart は attach 関数から戻る前である。両方のスクリプトはほぼ並列に動作する。（正確にはメインクラスのスクリプトを呼んだ後サブクラスのスクリプトを実行）
- 現在ではスレッドの利用を推奨している。

◆ detach

アクターのサブクラスを停止させる

■ 構文

```
detach();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- 今このアクターに割り当てているサブスクリプトを止める。止める前にはサブクラスの OnVanish 関数が呼ばれる。

◆ setgv

グローバル値スロットに数値を保存する

■ 構文

```
setgv(page,slot,value);
```

■ 引数

page	ページ番号。0 ～ 3。
slot	スロット番号。0 ～ 511。
value	保存する値。実数。

■ 戻り値

なし

■ 解説

- AIMS で用意されている「グローバル値スロット」に数値（実数）を保存する。
- 値スロットは 1 ページにつき 512 個、それが 0 ～ 3 の 4 ページあって合計 2048 スロット存在する。
- 全 4 ページのうち 1 ～ 2 ページは「AIMS 終了時に値が自動保存され、次の起動時に自動復帰する」という性質がある。
- グローバル値スロットには数値しか保存できないので、serialize/deserialize（226 ページ）が実装された最近では setgv の利用はあまり推奨されない。

◆ getgv

グローバル値スロットから数値を読み出す

■ 構文

```
value = getgv(page,slot);
```

■ 引数

page	ページ番号。0 ～ 3。
slot	スロット番号。0 ～ 511。

■ 戻り値

value	指定スロットに保存されていた値。

■ 解説

- グローバル値スロットから数値を読み出す。戻り値は実数。

◆ saveGlobalVars

グローバル値スロットを保存する

■ 構文

```
saveGlobalVars();
```

■ 引数

なし

■ 戻り値

なし

■ 解説

- グローバル値スロットのページ 1 と 2 の内容を即座に保存する。ファイル名は "[EXE ファイルの名前].sav" で固定されており、変更はできない。

1-1 アプリケーションコンフィグの設定項目

◆ APPLICATION_NAME

アプリケーション名を設定する

■ 構文

```
APPLICATION_NAME = "アプリケーション名"
```

■ 解説

- アプリケーションの名前を決める。タイトルバーに最初に表示される文字列になる。

◆ MAXIMUM_PLAYERS

最大プレイヤー数を定義する

■ 構文

```
MAXIMUM_PLAYERS = (1〜8)
```

■ 解説

- プレイヤーの数を設定する。最大 8 人まで対応している。

◆ MAXIMUM_TRIGGERS

プレイヤー当たりの使用ボタン数を定義する

■ 構文

```
MAXIMUM_TRIGGERS = (1〜12)
```

■ 解説

- プレイヤーが使用するボタンの数を定義する。

◆ SAVE_PATH

セーブファイルのパスを定義する

■ 構文

```
SAVE_PATH = "保存先パス"
```

■ 解説

- セーブファイルの置き場所を指定する。空文字列だと AIMS 本体と同じ場所になる。
- 環境変数が利用できる他、「%MYDOCUMENTS%」がマイドキュメントのパスに置換される。この値に限り、パスの区切り文字は「\\」(フォントによっては半角円記号二つ) とすること。また、最後は必ず「\\」で終わること。(空文字列にする場合を除く。)
- なお、指定されたパスが見つからない場合は自動的にフォルダを作成する。
- Windows Vista 以降はプログラムと同じ場所にデータを保存できない場合もあるので、適切なセーブ先を指定しておくことが望ましい。一般的にはユーザーごとに決められたアプリケーションデータの置き場所があるので、そこに保存すべきである。(「%APPDATA%」で参照できる)

■ 例

```
SAVE_PATH = "%APPDATA%\\D.N.A. Softwares\\AIMS\\"
--↑は一般的に"C:\Users\ユーザー名\AppData\Roaming\D.N.A. Softwares\AIMS\"と置き換わる

SAVE_PATH = "%MYDOCUMENTS%\\D.N.A. Softwares\\AIMS\\"
--↑は一般的に"C:\Users\ユーザー名\Documents\D.N.A. Softwares\AIMS\"と置き換わる
```

◆ USE_MOUSE

マウスポインタを表示するかどうか設定する

■ 構文

```
USE_MOUSE = (0 or 1)
```

■ 解説

- 0 にすると Windows のマウスポインタを表示し、1 にするとマウスポインタを隠す。
- マウスポインタを AIMS で使用するかどうかの設定であるが、別にこれを 0 にしたからといって getMouse 系の命令が使えなくなるわけではない。

◆ CLIP_MOUSE_POINTER

マウスポインタの移動範囲を制限するか設定する

■ 構文

```
CLIP_MOUSE_POINTER = 0 or 1
```

■ 解説

- マウスポインターの移動範囲をウィンドウの枠内に限定するかどうかを設定する。移動範囲の制限は AIMS がアクティブの時だけ有効になる。

◆ RESOLUTION_X,RESOLUTION_Y

アプリケーションの物理解像度を設定する

■ 構文

RESOLUTION_X = 物理解像度幅

RESOLUTION_Y = 物理解像度高さ

■ 解説

- 物理解像度、つまり画面モードを設定する。
- ウィンドウモードのときは VGA の制限の範囲で任意の幅と高さが指定できるが、フルスクリーンモードの場合は特定の組み合わせ以外はエラーとなる場合がほとんどである。
- フルスクリーンにできる画面モードとしては、幅×高さ = 640 × 480、800 × 600、1024 × 768 あたりが一般的である。昔であれば 320 × 240 なども指定できたが、最近の VGA では対応していないものもあるため、注意が必要である。

■ 例

```
-- もっとも基本的な640×480モードを指定
RESOLUTION_X = 640
RESOLUTION_Y = 480
```

◆ VRESOLUTION_X,VRESOLUTION_Y

論理解像度の初期値を設定する

■ 構文

VRESOLUTION_X = 論理解像度幅

VRESOLUTION_Y = 論理解像度高さ

■ 解説

- 論理解像度を設定する。画面右下の座標が物理解像度に関わらず(VRESOLUTION_X-1, VRESOLUTION_Y-1)となるように座標系が変換される。アクターなどすべての描画要素は物理解像度と論理解像度の関係に応じて拡大、縮小される。
- 普通はRESOLUTION_X、RESOLUTION_Yの各値と同じものを設定しておけばいい。

■ 例

```
-- 320×240のゲームを作りたいが、
-- VGAの互換性のために画面モードは640×480にするというときの設定
RESOLUTION_X = 640
RESOLUTION_Y = 480

VRESOLUTION_X = 320
VRESOLUTION_Y = 240
```

◆ SLEEP_WHEN_INACTIVE

フォーカスが外れた場合に処理を停止するか設定する

■ 構文

```
SLEEP_WHEN_INACTIVE = (0 or 1)
```

■ 解説

- ウィンドウからフォーカスが外れた場合に、処理を止めるか (1)、それとも処理を続行する (0) か選ぶ。
- どちらを選んだ場合でも、ogg ストリーミングは停止しない。

◆ USE_ESC_IN_APPLICATION

ESC キーの取り扱いを定義する

■ 構文

```
USE_ESC_IN_APPLICATION = (0 or 1)
```

■ 解説

- AIMS では標準で ESC キーがプログラム終了に割り当てられているが、この値を 1 にすると ESC キーでのアプリケーション終了が無効になる。

◆ BIND_PAD_*P

論理ボタン→パッドボタンの割り当て定義

■ 構文

```
BIND_PAD_1P_TRIG1 = 0〜15;
BIND_PAD_1P_TRIG2 = 0〜15;
BIND_PAD_1P_TRIG3 = 0〜15;
...(以下ボタン数だけ繰り返す)

BIND_PAD_2P_TRIG1 = 0〜15;
BIND_PAD_2P_TRIG2 = 0〜15;
BIND_PAD_2P_TRIG3 = 0〜15;
...(以下プレイヤー数だけ繰り返す)
```

■ 解説

- 各プレイヤーのゲームパッド割り当てを定義する。0 〜 15 としているのは仕様上の限界であり、実際にはパッドにあるボタンの数で上限が決まる。
- 値に -1 を指定すると、その論理ボタンはパッドボタンに割り当てを行わない。
- 方向入力は常に X 軸 Y 軸またはハットスイッチに割り付けられるので、定義できない。

◆ BIND_KEY_*P

論理ボタン→キーボードの割り当て定義

■ 構文

```
BIND_KEY_1P_DIR_U = DIK_*** -- 上方向入力
BIND_KEY_1P_DIR_D = DIK_*** -- 下方向入力
BIND_KEY_1P_DIR_L = DIK_*** -- 左方向入力
BIND_KEY_1P_DIR_R = DIK_*** -- 右方向入力
BIND_KEY_1P_TRIG1 = DIK_*** -- トリガ1
[BIND_KEY_1P_TRIG1_ALT = DIK_***]
BIND_KEY_1P_TRIG2 = DIK_*** -- トリガ2
...(以下ボタン数だけ繰り返す)

BIND_KEY_2P_DIR_U = DIK_***
BIND_KEY_2P_DIR_D = DIK_***
BIND_KEY_2P_DIR_L = DIK_***
BIND_KEY_2P_DIR_R = DIK_***
BIND_KEY_2P_TRIG1 = DIK_***
BIND_KEY_2P_TRIG2 = DIK_***
...(以下プレイヤー数だけ繰り返す)
```

■ 解説

- 各プレイヤーのキーボード割り当てを定義する。
- ほぼBIND_PADと同じだが、渡すのはDIK_で始まるDirectInputのキーコード定数である。
- また、方向入力のボタンも割り当てなければならない。
- それぞれの項目名に「_ALT」を付けた値を定義すると、論理ボタンに2つめのキーを定義できる。どちらのキーを押しても同じ論理ボタンが押されていると判定される。

◆ USE_MODPLUG

プラグインによる MOD 再生を構成する

■ 構文

```
USE_MODPLUG = (0 or 1);
```

■ 解説

- MODPLUG Player SDK の読み込みを指示し、MOD ファイルの再生を行えるよう AIMS を設定する。
- USE_MODPLUG を 1 に設定すると、AIMS 本体のあるディレクトリから MPPSDK.DLL を読み込み、playMod と playModLoop の 2 つの関数が有効になる。
- USE_MODPLUG を 1 に設定しているのに MPPSDK.DLL が見つからない場合はエラーとなり AIMS が起動しない。他方、USE_MODPLUG を 0 にしているのに playMod や playModLoop を呼ぶとエラーとなる。
- MPPSDK.DLL は AIMS v1.40 から開発者パッケージ内に同梱されている。必要に応じて再配布してよい。

AIMS
関数リファレンス

編著：D.N.A.Softwares
http://aims.dna-softwares.com/
aims@dna-softwares.com