|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 仕様書 | | 人形及び敵スプライトのエフェクト | | |
| 種類 | 追加と編集 | | 作成者 | たかなし |
| 日付 | 2014/7/21 | |

**1　内容**

　人形及び敵の画像に対して変化を与えるエフェクトを新規に作成し、表示のテストを行う。

**2　予備知識**

2. 1　スプライト

　戦闘画面の人形及び敵の画像は、スプライトというもので管理されている。このゲームのプログラムで言うスプライトは、拡大縮小や回転など変形に特化した画像を扱うクラスであり、その基本クラスはSprite\_Baseクラスである。

　人形はSprite\_BattleDollクラス、敵はSprite\_BattleEnemyクラスと派生しているが、必要なメンバ変数などを増やした程度のものであり、基本的にはSprite\_Baseの機能が多少拡張した程度のものと考えて構わない。

2. 2　スプライトのメンバ変数と描画の仕組み

　一旦スプライトをアニメーションさせる話は後回しにして、スプライトのメンバ変数と描画についてノベル。

　スプライトの変形などのパラメータを保持するのが、Sprite\_Baseクラスのメンバでもある、SpriteParam構造体（Sprite\_Base.hを参照）である。（以下ソースコード抜粋）

// 本来の描画位置からの変位

float dx, dy;

// 回転角

float angle;

// 隠すか否か(trueだと表示しない。)

// visibleがtrueかつhideがfalseの場合のみ描画を行う。

// 両者の違いは、hideはmorphIDによって設定され、

// visibleはそれ以外のプログラムから設定される。

bool hide;

// 不透明度(0～255, 255で不透明)

int opacity;

// 拡大倍率(x方向、y方向それぞれ)

float xScale, yScale;

// アニメーションする場合のセル。使用しない場合は0。

int cell;

// 特殊効果。白く光る、透明など。

// 使用するものだけ対応させれば良い。

BYTE effect;

// 汎用パラメータ。

// この値だけは、morphIDによって変化せず、

// SetMorphID内で適宜渡される。

// 描画の際はこの値を使ってもよい。

int param;

上記のように、基本位置からのずれや回転、拡大倍率など、画像の変形に関連するパラメータが並んでいる。

しかし、実はこれらのパラメータは定義されているだけで、このパラメータを変えたからと言ってその通りに画像が変形されることは必ずしも保証されない。

Sprite\_BaseのDraw関数は

virtual void Draw() const{};

この通り何も書いておらず、実際の描画は派生クラスに丸投げされている。

すなわち、実際の描画をどうするかは、Sprite\_BattleDollなどの派生クラスのDraw関数内で記述する必要がある。

このような設計にしたのは、描画部分をより柔軟にしたかったためである。例えばSprite\_Base内で描画を決め、派生クラスでは描画はSprite\_Baseクラスの内容に従うとすると、例えば複数画像を組み合わせたりしたい場合に対応が難しい。そうならないよう、描画に関しては二度手間（と言うか、派生クラスの数だけ手間）になる部分があるとしても描画は派生クラスに任せるようにした。

2. 3　スプライトエフェクト

　上記のSpriteParamは画像の変形方法を表す構造体だが、なめらかにアニメーションするエフェクトなどを付加したい場合はこのSpriteParamの値が時間に応じて変化しなければならない。それを行うための仕組みがスプライトエフェクトであり、その鍵となるのがSetMorphIDによるmorphIDという値の指定である。

　Sprite\_BaseにはmorphIDという変数があり、SetMorphIDという関数でmorphIDに値を入れることが出来る。このmorphIDをセットするというのが、スプライトアニメーションを開始させるということを意味する。

プログラム的には、morphに関する変数として以下の変数がある。

morphID　…　変形アニメーションのID

morphCount　…　現在の変形アニメーションが現在何フレーム目か

morphTime　…　現在の変形アニメーションが終わるフレーム数

morphSkipTime　…　アニメーションがスキップ可能（後述）になるフレーム数

SetMorphIDを行うと、morphIDに値が入るだけでなく、morphCountの値がリセットされ、morphTimeとmorphSkipTimeに、morphIDに対応した値が入る。そして、morphIDが入っている間は自動でアニメーションが進み、morphCountがmorphTimeに達した時にアニメーションが終了する（自動でmorphIDが規定の値である0に戻る）。

　要は、前もって必要な内容をプログラムに書いておけば、後はSetMorphIDを行うだけでそのスプライトに対してアニメーションが実行される。

**3　手順**

3. 1　新しい自作のスプライトエフェクトに対してmorphIDを割り当てる

　morphIDの各定数は、Header\_System\Header\_Static\Static\_Sprite.hに記されている。まず、そこに新たに追加したいスプライトエフェクトを定数で書き、数値を割り当てる。仮に派生クラスでしか使用しないエフェクトがあったとしても、定数はStatic\_Sprite.hに記述する。

　割り当てる数値は、派生クラスごとに百の位の値を変える。例えば、複数のクラスに共通するエフェクトの数値は0, 1, 2…と割り当て、Sprite\_BattleDollのみに関するエフェクトの数値は100, 101, 102…と割り当てる。重複しなければ特に深く考える必要はない。

3. 2　エフェクトの継続時間を指定する

　SetMorphIDではIDを指定すれば自動でエフェクトの継続時間が決まるが、その時間を実際に記述しているのがSprite\_Base\_Morph.cpp内のSetMorphIDの中身である。

　以下、内容を抜粋する。

switch(morphID){

case SPMORPH\_NONE:

break;

case SPMORPH\_DISAPPEAR:

morphSkipTime = p;

morphTime = p;

param.param = p;

break;

case SPMORPH\_ACTIVATE:

morphSkipTime = SPMORPH\_ACTIVATE\_TIME;

morphTime = SPMORPH\_ACTIVATE\_TIME;

break;

case SPMORPH\_DISACTIVATE:

morphSkipTime = SPMORPH\_ACTIVATE\_TIME;

morphTime = SPMORPH\_ACTIVATE\_TIME;

break;

このように、地道にswitch ~ case ~を書いてmorphTimeとmorphSkipTimeを代入している。なお、値が代入されていない場合はエラーと見なされ、即座にmorphIDがリセットされる。

なお、morphSkipTimeはエフェクトがスキップ可能になる時間であるが、これは、例えば、戦闘でこのエフェクトを行う際、この時間が経過した後は、エフェクトが終わるのを待たずに戦闘を先に進めるという意味の変数である。

morphSkipTimeを短くすることで、例えば全体ダメージの際に微妙にタイミングをずらして連続してダメージを与える、といった演出が可能になる。

逆に、飛ばしたくない場合はmorphSkipTimeにはmorphTimeと同じ値を入れる。

3. 3　スプライトの描画を指定する

　スプライトエフェクトの描画の前に、先にスプライトの描画について記す。

　前述の通り、スプライトの描画の実際はその派生クラスによって異なっている。

　例としてSprite\_BattleEnemyのDraw関数を引用する（14/05/21）。

void Sprite\_BattleEnemy::Draw() const{

if(refID == 0) return;

if(hImg == 0) return;

if(pEnemy == NULL) return;

// 透明度

SetDrawBlendMode(DX\_BLENDMODE\_ALPHA, param.opacity);

if(visible){

// 表示状態の判定

if(!param.hide){

// 本体の描画

DrawRotaGraph3F(

GetX(), GetY(), cx, cy,

baseExRate\*param.xScale, baseExRate\*param.yScale,

hImg, 1, 0);

}

// HPの描画

DrawHPGauge((int)GetX(), (int)GetY());

}

SetDrawBlendMode(DX\_BLENDMODE\_NOBLEND, 255);

}

これを見ると、HPゲージの描画もスプライトに含まれていることが分かる（DrawHPGaugeの中身）。しかも、HPゲージに対しては変形などは適用されず、スプライトの表示・非表示の変数（visible）によって描画を行うか否かだけが決まっている。

このように、このゲームの「スプライト」は単に画像ではなく、便宜上ひとかたまりとして扱いたい画像要素をまとめて全部「スプライト」と呼んでSprite\_Baseクラスから派生させている。

描画には多くはSpriteParam構造体の中身を使っているが、baseExRateというのはSprite\_BattleEnemyの変数である。元の画像サイズに対する描画倍率の比を示している。

この関数の内容はこれで固定ではなく、変更して構わない。特にフィルタなどをかける場合、この関数の内容を編集することで対応させる必要がある。

3. 4　スプライトエフェクト

　前項はSpriteParamの値に応じてどう描画するかという内容だったが、この項はSpriteParamの値をどう変化させるかということに関する。

　その内容は、Sprite\_Base\_Morph.cpp内のUpdateMorph関数に書かれている。ここでswitch ~ case ~ により地道に分岐し、それぞれでparamの値をいじっている。以下に抜粋する（14/07/21時点での全内容）

void Sprite\_Base::UpdateMorph(){

switch(morphID){

case SPMORPH\_NONE:

break;

case SPMORPH\_DISAPPEAR:

if(morphCount < morphTime){

param.opacity = (int)PARAM(GetMorphRestRate()\*255.0);

}else{

param.opacity = 255;

visible = false;

}

break;

case SPMORPH\_ACTIVATE:

param.dy = -SPMORPH\_ACTIVATE\_VY\*GetMorphRate();

break;

case SPMORPH\_DISACTIVATE:

param.dy = -SPMORPH\_ACTIVATE\_VY\*GetMorphRestRate();

break;

case SPMORPH\_BLINK:

if(morphCount < morphTime){

param.hide = true;

}else{

param.hide = false;

}

break;

}

return;

}

この関数内でparamの値を変え、その値を用いてDraw関数で描画を行うことで、結果的にスプライトエフェクトが描画されることになる。スプライトエフェクトの編集というと、主にこちらをいじることになる。

**4. 確認**

4. 1　自作のスプライトエフェクトを確認する

　上記の一通りの手順でスプライトエフェクトを作成した場合、Scene\_Battle\_Action.cpp内のScene\_Battle::Action\_Damage関数内、

// スプライトに対する処理を行う

if(pAction->GetOpponent()->IsDoll()){

pSprite = (Sprite\_Base\*)GetDollSprite((Game\_BattleDoll\*)pAction->GetOpponent());

if(pSprite != NULL){

// タスクを発生させる

pSprite->SetMorphID(SPMORPH\_DAMAGE\_DOLL, false, 8);

}

}else{

pSprite= (Sprite\_Base\*)GetEnemySprite((Game\_BattleEnemy\*)pAction->GetOpponent());

if(pSprite != NULL){

// タスクを発生させる

pSprite->SetMorphID(SPMORPH\_BLINK, false, 8);

}

}

の赤字部分を変えるのが最も分かりやすい。3つの引数は、

適用したいmorphID、強制するかどうか（trueの場合、何かのエフェクトの最中でも上書きする。なお、複数エフェクトの重ねがけは出来ない）、任意のパラメータを意味する。

人形または敵がダメージを受ける際、そのスプライトエフェクトを見ることが出来る。