|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 仕様書 | | 戦闘時エフェクトの組み込み | | |
| 種類 | 追加と編集 | | 作成者 | たかなし |
| 日付 | 2014/7/19 | |

**1　内容**

　戦闘時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　エフェクトクラス

　戦闘時に画面上に表示されるタイプのエフェクトには、主に二つの種類がある。

・情報関連のエフェクト（MyTask\_InfoEffectクラス）

　ダメージ表示など、情報を伴うエフェクト。

　便宜上、文字もエフェクトとして扱っている。

　基本的に演出関連のエフェクトより前に描画される。

・演出関連のエフェクト（MyTask\_ParticleEffectクラス）

　攻撃アニメーションなど、本来の意味でのエフェクト。

この他に、画面全体にかかるエフェクトなどがあるが、ここでは扱わない。

この二つのエフェクトは、いずれもMyTask\_ObjectEffectクラス→MyTask\_BaseEffectクラス→MyTaskクラスから派生している。MyTaskクラスは、メモリを最初に確保しておいてその中でオブジェクトを作成したり消去したり出来るクラス、MyTask\_BaseEffectクラスは画面にかかるエフェクトなどを含めたエフェクト全体の親となるクラス、ObjectEffectクラスは、位置情報を持つ物体（火花、爆発など）のエフェクトの親となるクラス。

ちなみにこの二つのクラスは互いに代用が出来るが、便宜上分けている。

2. 2 エフェクトの特徴

　エフェクトの特徴は、他のクラス（例えば人形とか、ウィンドウとか）と相互作用せず、一定フレームの間表示され、表示が終了すると消えるということにある。

（例外として、エフェクトが別のエフェクトを生み出すことは可能である）。

他のクラスと相互作用する何かを作りたいのであれば、エフェクトクラスでなく他のクラスを使用すべきである。

**3　手順**

3. 1　使用する画像を増やす。

　仕様書「画像の追加」を参照のこと。

3. 2　エフェクトの種類を増やす

*各エフェクトは、表示内容ごとにtypeIDという値で管理されている（このメンバ変数はMyTask\_BaseEffectクラスで宣言されている）。*

*そのtypeIDがとりうる値（有効なtypeIDの値）を増やす。*

　Header\_Battle\\BHeader\_Static\\Static\_InfoEffect.hを開くと、

// 敵に対するダメージ表示

#define INFO\_DAMAGENUM\_ENEMY 1

#define INFO\_DAMAGENUM\_ENEMY\_LIFETIME 36

のような定数の定義がある。

この、INFO\_○○がMyTask\_InfoEffectクラスの定数である。

また、INFO\_○○\_LIFETIMEは、パーティクルが画面上に存在する時間を表している。

このLIFETIMEも後ほど記述する必要がある。

3. 3　描画以外の部分

　エフェクトの重要な部分は描画であるが、描画以外の部分を先に作成する。

Source\_Battle\\BSource\_Instance\\MyTask\_Particle(or Info)Effect.cppの中の

MyTask\_Particle(or Info)Effect::SetLifeTime()

に、既にあるtypeIDのエフェクトと同様に、lifeTime値を代入する式を書く。

3. 4　描画

Source\_Battle\\BSource\_Instance\\MyTask\_Particle(or Info)Effect\_Draw.cppの中の

MyTask\_Particle(or Info)Effect::Draw()

に、既にあるtypeIDのエフェクトと同様に、描画したい内容を書く。

3. 5　描画に用いることの出来る関数

以下に、描画を行う上で便利と思われる関数を記す。

GetLifeTime() … このエフェクトの存在時間を返す。

GetRestTime() … このエフェクトがあと何フレーム存在するかを返す。

GetExistRate() … このエフェクトが、存在時間に対して現在どれだけ存在しているかを返す。すなわち、エフェクトの存在時間に対して、0.0～1.0まで変化する。

**4. 確認**

4. 1　とりあえず確認する

Scene\_Battle内のScene\_Battle::Update関数内を見ると、

//##########################################

// タスクを発生させるテスト

という部分があり、その下に

/\*

if(sceneTime % 120 == 60){

MyTask\* pTask = gMyTask\_InfoEffect->Call();

if(pTask!=NULL){

new (pTask) MyTask\_InfoEffect(

GetRand(200), GetRand(200), 0, 0, 0);

}

}

\*/

というコメントアウトされた部分がある（見つからなければ追記する）。

これは、120フレーム毎に、指定されたエフェクトを発生させるというもので、

new (pTask) MyTask\_InfoEffect(

GetRand(200), GetRand(200), 0, 0, 0);

部分を任意に変えた上でコメントアウトを外すことにより、好きなエフェクトを表示させることが出来る。戦闘の間中ずっと、120フレーム毎に指定したエフェクトが発生する。時間や数は適宜いじってよい。MyTask\_InfoEffectの括弧の中に入っている5つの変数がエフェクトの設定で、x座標、y座標、typeID（上で決めたもの）、パラメータその1、パラメータその2となっている。パラメータその1、その2はtypeIDごとにどういう意味の変数か異なるもので、例えばtypeID = INFO\_DAMAGENUM\_DOLL（人形がダメージを受けた時の数値表示）の場合は、パラメータ1はダメージの値、パラメータ2は使用しないということになっている（14/07/21時点）。

ParticleEffectも同様の記述が出来る。ただし、下記のようにgMyTask\_InfoEffectではなくgMyTask\_ParticleEffectを使用する。

if(sceneTime % 120 == 60){

MyTask\* pTask = gMyTask\_ParticleEffect->Call();

if(pTask!=NULL){

new (pTask) MyTask\_ParticleEffect(

GetRand(200), GetRand(200), 0, 0, 0);

}

}

4. 2　より具体的な使用場所

具体的に、使用されている場所の例を挙げる。

①Scene\_Battle\_Action.cpp内のAction\_Damage関数内。

これは人形または敵がダメージを受けた時に実行される部分である。

内部で大きく

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

…… (A)

}else{

…… (B)

}

という分岐があるが、これは、ダメージを受けたのが人形であればA、敵であればBということを示している。

4. 3　技術的なこと

gMyTask\_xxxというのは、MyGroupというクラスのグローバル変数であり、これはメモリの管理を行っている。Call関数を実行すると、ゲームの起動時に確保したメモリから「ここにnewしていいよ」というメモリを取ってくる（もうメモリが余っていないなど、取って来れなかった場合はNULLを返す。その場合はエフェクトは発生しない）。

その場所を借りてnewしているのが実際のnewの内容である。通常のnewのようにnewの瞬間にメモリを確保するわけではなく、前もって確保している。

それぞれのgMyTask\_xxxには、どのクラスのタスクを割り当てるかが最初から決まっている（MyTaskList.cpp内のCreate\_MyTaskList関数。派生クラスがある場合は、最もクラスの大きさが大きい派生クラスを指定する。）。ここは基本的にはいじらないこと。