ユニークシーケンスを使用したギミックについて

ジムや四天王部屋などの閉じた空間もしくは使用個所が限定された場所において特別な仕掛けを作成したいときに利用することを目的とした仕組みです。

例）ジムギミック／Ｃ２通行人／四天王部屋演出

ユニークシーケンスクラスの用意

prog\src\field\unique\_sequenceにあるFieldUniqueSequence.hをインクルードし、

FieldUniqueSequenceクラスを継承したクラスを作成します。Cppファイル,hファイルを作成してください。

prog\include\fieldにあるFieldUniqueSeqId.hにユニークシーケンスＩＤを追加します。命名規則はUNIQUE\_SEQ\_ID\_「自分で決める好きな文字列」となります。ギミックに関連する名前が望ましいです。

prog\src\field\unique\_sequenceにあるFieldUniqueSequenceHeader.cppを開き、

作成したユニークシーケンスのヘッダーファイルをインクルード、および同ファイル内のCreate関数のswithcase文に自分の追加したいユニークシーケンスＩＤのcase文を追加します。

例）UNIQUE\_SEQ\_ID\_MIRRORを追加したい

// 鏡

case UNIQUE\_SEQ\_ID\_MIRROR:

{

nn::ro::Module\* p\_field\_module = dll::OnlyLoadFieldDll( "rom:/DllUSMirror.cro" );

pFieldmap->SetUniqueSequenceModule( p\_field\_module );

　unique\_sequence=

GFL\_NEW(heap\_memory)FieldUniqueSequenceMirror(pFieldmap);

}

break;

モジュールをＤＬＬ化した場合の例です。

ＤＬＬ化していない場合は自分で作成したユニークシーケンスクラスをＮｅｗするだけになります

resource\field\zonedataにあるzonetable.xlsを開きUNIQUE\_SEQの項目に自分が追加したユニークシーケンスＩＤの自分で決めた文字列部分を記述してください。

ここまでの作業で空のユニークシーケンスができます。

所定のゾーンに行くことでユニークシーケンスクラスの各関数を呼ぶことができます。

ユニークシーケンスの関数

仮想関数化してあるので詳しくはソースコードを参照してください。

実装に必要そうな関数を紹介します

//----------------------------------------------------------------------------

/\*\*

\* @brief SETUPの最後に呼ばれる関数

\*/

//-----------------------------------------------------------------------------

virtual b32 SetupLast(void);

リソースのロードやギミックワークの作成をするシーケンスです。

ギミックワークについては別途説明します。

//----------------------------------------------------------------------------

/\*\*

\* @brief READYの最後に1回だけ呼ばれる関数

\*/

//-----------------------------------------------------------------------------

virtual void ReadyLastOnce(void);

ロードしたリソースを使用してモデルのセットアップや初期配置等を行います

//----------------------------------------------------------------------------

/\*\*

\* @brief UPDATEの最初に呼ばれる関数

\*/

//-----------------------------------------------------------------------------

virtual void UpdateFirst(void);

メイン関数となります常に監視したい処理を書きます。

//----------------------------------------------------------------------------

/\*\*

\* @brief UPDATEの最後に呼ばれる関数

\*/

//-----------------------------------------------------------------------------

virtual void UpdateLast(void)

同上。違いは更新タイミングです

//----------------------------------------------------------------------------

/\*\*

\* @brief FREEの最初に1回だけ呼ばれる関数

\*/

//-----------------------------------------------------------------------------

virtual void FreeFirstOnce(void);

解放関数です。シーケンス内で作成したクラスやメモリの解放を行ってください。

ギミックワーク（アサインワーク）

ギミックの中には状態をセーブしたい案件が含まれるかもしれません。その時に使用しててください。

例）

/// ギミックワークの準備

gamesystem::GameData\*p\_gdata=

m\_pFieldmap->GetGameManager()->GetGameData();

// アサインの状態を確認して初期化

b32 is\_assigned = GimmickWorkAccessor::IsAssigned(

p\_gdata, GIMMICK\_ID\_GYM\_ICE);

// アサインを行う

GimmickWorkAccessor::SecondAssign( p\_gdata, GIMMICK\_ID\_GYM\_ICE);

// バッファーを保存

mp\_work\_buffer = GimmickWorkAccessor::GetWork( p\_gdata );

// アサインしていない状態の場合はデータを初期化

if( is\_assigned == FALSE )

{

ICE\_GYM\_WORK gym\_work;

gfl::std::MemClear( &gym\_work, sizeof(ICE\_GYM\_WORK) );

BufferAccessorType<ICE\_GYM\_WORK>::SetValue(mp\_work\_buffer,0,gym\_work);

}

prog\src\fieldにあるFieldGimmickWorkAccessor.hに自分がアサインしたいＩＤを追加します。

確保したい構造体を作成しておきます。

IsAssigned関数で自分自身のワークＩＤがアサインされていないことを確認し、

アサインＩＤを設定します。

以降、アサインＩＤを介して、アサインワークのバッファにアクセスします。ワークサイズは３２バイト固定で、好きなように使用できます。

ワークの値適用においては、取得したポインタをキャストするのではなくSetValue関数を使用して下さい。

ギミックに使用するリソース

リソース管理については自由に行えますが、今回はギミックセットアップクラスを使用してセットアップすることが多かったので、ここで紹介します。

gimmick\_mdl\_setup.h .cpp (prog\src\field)

使用するリソースの定義等をdatファイル等にあらかじめ配列化して保持しておきセットアップを行います

セットアップ定義ホッパー対応

typedef struct {

MDL\_SETUP\_TYPE setup\_type; ///< セットアップタイプ

s8 binary\_id; ///< バイナリデータID

RES\_ARY\_TYPE type; ///< 配列タイプ

} MDL\_SETUP\_DATA\_EXP;

MDL\_SETUP\_TYPE setup\_type; モデル、アニメ、ライト等リソースの種類を定義します

s8 binary\_id; アーカイブバイナリデータIDです

RES\_ARY\_TYPE type; リソースがホッパーかどうかを定義します

上記の構造体を配列化してあらかじめ保持しておきます。

ホッパーを使用しない場合、MDL\_SETUP\_DATA構造体の配列が使用できます。

・クラスをＮｅｗします。ホッパー対応データの場合、引数にセットアップ配列のポインタを渡しておきます。

・LoadArc、LoadArcWaitメソッドを使用して対象アーカイブファイルをロードします。

・ホッパー非対応の場合、SetupResourceメソッドを使用して、リソースのセットアップを行います

Creaate○○メソッドを使用して、モデルやアニメの作成を行います。リソース管理は行いますが、使用するモデルやアニメのポインタは作成先で管理することになります。