

RPGの作り方

- [RPGの作り方](#)
 - [これは何？](#)
 - [RPGツクールを作る](#)
 - [全体像](#)
 - [マップ](#)
 - [イベントオブジェクト](#)
 - [特殊イベントオブジェクトの配置について](#)
 - [イベントスクリプト](#)
 - [ラベル方式](#)
 - [INITラベル](#)
 - [end命令](#)
 - [load命令](#)
 - [BOOL変数のみ](#)
 - [1行1命令方式](#)
 - [イベントスクリプトのコマンド例](#)
 - [フロー制御](#)
 - [フラグ制御](#)
 - [メニュー](#)
 - [ダイアログ](#)
 - [メッセージ](#)
 - [プレイヤー制御](#)
 - [イベントオブジェクト制御](#)
 - [アイテム制御](#)
 - [サウンド](#)
 - [演出](#)
 - [参考](#)

これは何？

RPGの核となるシステムである、「マップ」と「イベントスクリプト」の作り方・連携について解説します。

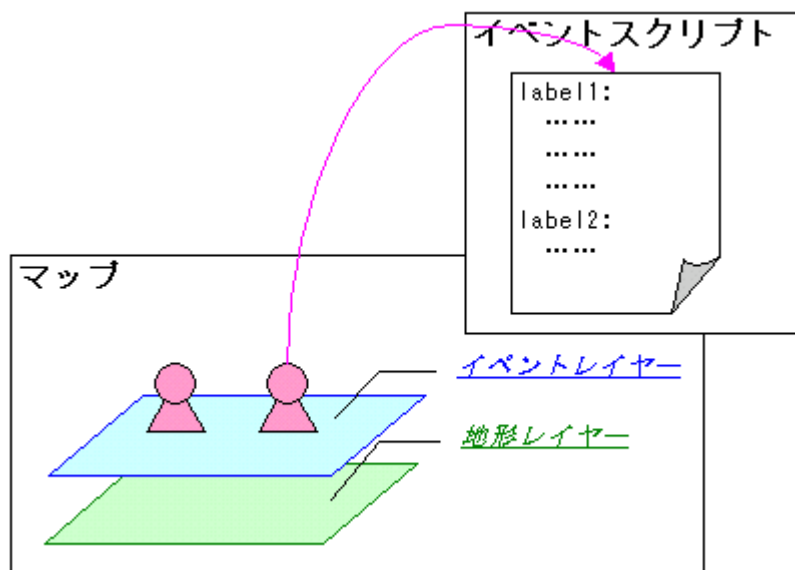
ただ、RPGの本質は、プレイヤーのステータス、アイテム、敵パラメータの調整や、シナリオや世界観などの雰囲気作りにあります。

そういった本質部分の作り込みをやすくする、というのが、「マップ」「イベントスクリプト」システムの目的になります。

RPGツクールを作る

ここで構築するシステムは「RPGツクール」の簡易版のようなものです。この内容が今一つピンとこない場合は、「RPGツクール」を試しに試してみるのもいいかもしれません。

全体像



「マップ」は、地形レイヤーとイベントレイヤーという2層構造になっています。
1つの「マップ」につき、1つの「イベントスクリプト」が対応します。

地形レイヤーには、草原・山・海などプレイヤーが移動するフィールドを配置します。

イベントレイヤーは、NPCやお城など、プレイヤーが接触したり、なんらかのアクションを起こすことで、イベントが発生するオブジェクトです。
つまり、イベントスクリプトを呼び出すオブジェクトです。
(ちなみにRPGツールでは、1つのイベントオブジェクトが1つのイベントスクリプトを持っています。)

「イベントスクリプト」は、イベントが、ずらーっと書かれたテキストです。
ここではシンプルに、

- ラベル方式
- 1行1命令方式
- BOOL変数のみ

をとることにします。

マップ

地形レイヤーは、たいていの場合通過可能・通過不能オブジェクトを配置するだけなので、問題はないと思います。

ただ、イベントオブジェクトについては、イベントスクリプトとの連携を考慮する必要があります。

イベントオブジェクト

イベントオブジェクトは、1つの「イベントID」を保持しています。
イベントの発生条件を満たした場合、「イベントID」と同名のラベルにジャンプし、イベントを実行します。

特殊イベントオブジェクトの配置について

イベントスクリプトで処理をさせるまでもない場合、特殊なイベントオブジェクトとして処理を行い、イベントを呼び出さないようにします。

例えば、鍵のかかった扉があるとします。これがゲーム中、1度しか登場しない場合、イベントで対応した方がいいです。ですが、これが20、30とあると、そのたびに同じイベントスクリプトを20、30と書くこととなります。これは面倒なので、扉イベントオブジェクトとして、自動でイベントを処理するようにします。

イベントスクリプト

イベントスクリプトは、

- ラベル方式
- INITラベル
- end命令によるイベント終了
- load命令による別マップへの移動
- BOOL変数のみ
- 1行1命令方式

で実装するとシンプルになります。

ラベル方式

ラベル方式とは、

```
label1:  
.  
.  
goto label1
```

というように、goto命令でスクリプト内のラベルに、どこからでもジャンプできるようにする方式です。

関数呼び出し方式と比べると、引数が渡せなかったり、再利用が困難であったりしますが、イベントスクリプトでは、引数渡しも特に必要なく、使い捨てのスクリプトを書くことが多いので、この方が、有利な点が多かったりします。
(実装も楽だし、、、)

実装の注意点としては、goto命令が呼ばれるたびに、先頭からラベルを検索するのはあまり効率が良くないので、最初にスクリプトをロードした時点で、ラベル名をキー、シーク位置を値としたハッシュテーブルを作っておくと、goto命令の処理を速くすることができます。
(まあ、長いスクリプトを書かないようにすれば問題ないのですが)

INITラベル

最初にマップ・イベントスクリプトをロードした時点で、何らかの初期化処理を行いたい場合があります。

(例えば、町に入った直後に会話イベントが発生する、など)
そのために、イベントスクリプトの先頭に、

```
INIT:  
  // 会話イベント  
end
```

という初期化ラベルを記述しておくことができるようにしておく、何かと便利です。

end命令

マップ・イベントスクリプトをロードすると、イベント初期化 (INITラベル) を行った後、マップに制御が戻ります。

この「マップに制御を戻す」というのがend命令です。

逆に言うと、このend命令がないと、イベントスクリプトが制御を握ったままになります。

load命令

町の外に出た場合、階段に登った場合、などは、イベントの中で、load命令を呼び出し、別のマップ・イベントスクリプトをロードし、マップを切り替えるようにします。

BOOL変数のみ

RPGはフラグを消化するゲームであり、数値型変数はほとんど必要ありません。

例えば、

- プレイヤーのHPが30%以下なら、回復イベント発生
- プレイヤーのお金が3000G以上なら、お買い物イベント発生

という場合でも、

「プレイヤーのHP」「プレイヤーのお金が3000G」というのは、システム的な変数であり、汎用的な変数として用意する必要はないからです。

さらにフラグの実装のテクニックとしては、フラグをIDで管理するよりも、名前で管理したほうが意味が明確になります。

そして、ハッシュテーブルで、キーを変数名、値をBOOL値で持ち、キーが存在しない場合はFalse
というように変数を管理するとシンプルに実装できることになります。

1行1命令方式

例えば、宝石を手に入れたとします。

```
seton flag_jewel
```

で、とある商人に話しかけたとします。

```
if flag_jewel  
  talk "おおっ、その輝く宝石は、"
```

```

talk "...ぜひとも、私に譲ってくれないか?"
// 「はい」「いいえ」ダイアログを表示
yesno
// retはシステムフラグ
if ret
  talk "ありがとう、"
  talk "お礼の10000Gだ。"
  addmoney 10000
  setoff flag_jewel
  end
endif
talk "売る気になったら、いつでも話しかけてくれ"
endif

```

1行1命令方式だと、
flag_jewelがFalseの場合は、endifまでスキップする、
というようにシンプルな実装を行うことができます。

まあ、本当はアイテムはフラグでなくてアイテムリストに入れてしまうほうがいいのですがー

```

getitem "光り輝く宝石"

```

```

has "光り輝く宝石"
.
.
.
endhas

```

とか、

イベントスクリプトのコマンド例

参考までに、こんなコマンドを用意しておくのが良いのではないかと、
という例を挙げておきます。

フロー制御

```

load [マップ名] [イベントスクリプトファイル名] [移動座標X] [移動座標Y]
  [マップ名]のマップに移動、[イベントスクリプトファイル名]のイベントスクリプトをロー
  ドし、プレイヤー座標を(x,y)=([移動座標X],[移動座標Y])に移動する。
goto [ラベル]
  実行ポイントを[ラベル]に移動する
end
  スクリプトの実行を終了し、ゲームループに制御を返す
if [フラグ名]
  [フラグ名]がTrueであった場合、次の行を実行。Falseの場合、endifまで処理をスキップする
ifn [フラグ名]
  [フラグ名]がFalseであった場合、次の行を実行。Trueの場合、endifまで処理をスキップする
endif
  if/ifnの終了
has [アイテム名]
  [アイテム名]を所持している場合、次の行を実行。持っていない場合、endhasまで処理をス
  キップする

```

hasn [アイテム名]
[アイテム名]を所持していない場合、次の行を実行。持っている場合、endhasまで処理をスキップする

endhas
has/hasnの終了

wait [ms]
ゲームループに制御を[ms]ミリ秒返した後、スクリプトを再開する

フラグ制御

seton [フラグ名]
[フラグ名]をTrueにする

setoff [フラグ名]
[フラグ名]をFalseにする

メニュー

minit
メニュー項目の初期化

madd "項目"
メニュー項目に"項目"を追加する

mdisp [キャンセル表示]
メニューを表示する。[キャンセル表示]=Trueの場合、メニューに「キャンセル」を含める。選択結果は、システムフラグcase0～caseN（case0「キャンセル」、以下maddの順番）に入れられる。

mshop
ショップシステムとしてメニューを表示する。選択結果は、システムフラグcase0～caseN（case0「キャンセル」、以下maddの順番）に入れられる。

ダイアログ

yesno
「はい・いいえ」ダイアログを表示する。選択結果は、システムフラグretに入れられる（「はい」を選択した場合、ret=True）

メッセージ

talk "メッセージ"
メッセージウィンドウにメッセージを表示する。キー入力待ちをする。

msg "メッセージ"
メッセージウィンドウにメッセージを表示する。キー入力待ちはしない。

プレイヤー制御

pos [X] [Y]
プレイヤーの座標を(x,y)=([X],[Y])に移動する。

addhp [回復量]
HPを[回復量]回復する

hpmax
HPを最大値まで回復する

addmp [回復量]
MPを[回復量]回復する

mpmax
MPを最大値まで回復する

damage [ダメージ量]
HPを[ダメージ量]減らし、ダメージ処理を行う

イベントオブジェクト制御

- eon [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを可視にする
- eoff [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを不可視にする
- eauto [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを自動移動モードにする
- efix [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを現在の座標に固定する
- epause [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトのイベントを発生しないようにする
- elleft [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス左に移動する
- eup [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス上に移動する
- eright [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス右に移動する
- edown [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス下に移動する
- elefta [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス左に移動する(ゲームループに制御を返す)
- eupa [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス上に移動する(ゲームループに制御を返す)
- erighta [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス右に移動する(ゲームループに制御を返す)
- edowna [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを1マス下に移動する(ゲームループに制御を返す)
- edel [イベントオブジェクト名]
[イベントオブジェクト名]を持つイベントオブジェクトを削除する

アイテム制御

- getitem [アイテム名][フラグ名(省略可)]
[アイテム名]をプレイヤーの持物に追加する。[フラグ名]を指定した場合、[フラグ名]をTrueにする
- delitem [アイテム名]
[アイテム名]をプレイヤーの持物から削除する
- addmoney [お金]
所持金を[お金]増加させる
- submoney [お金]
所持金を[お金]減少させる

サウンド

- bgmon [BGM名][ms]
[BGM名]を[ms]ミリ秒かけて(フェードイン)再生する。
- bgmoff [BGM名][ms]
[BGM名]を[ms]ミリ秒かけて(フェードアウト)停止する。
- seon [効果音名][回数]
[効果音名]を[回数]回再生する

演出

fadein [color] [ms]
[ms]ミリ秒間かけて、画面を[color]色から元に戻す
fadeout [color] [ms]
[ms]ミリ秒間かけて、画面を[color]色にする

参考

[オレバスターのシナリオの仕組み](#)