RPG ツクール VXace で作る
2D アクションゲームスクリプト素材の作成

平成 27 年 1 月 30 日

阪南大学　経営情報学部 経営情報学科 花川典子ゼミ

5111066　釘嶋 大起

目次

第１章　はじめに

RPG ツクールは、株式会社エンターブレインが開発・販売をしている、プログラミングをしたことの無い人やプログラミング経験の浅い人でも、モチベーションがあればロールプレイングゲーム（以下 RPG）を作成できるソフトのシリーズである。2004 年に発売された RPG ツクール XP（以下 XP）と 2007 年に発売された RPG ツクール VX（以下 VX）と 2011 年に発売された RPG ツクール VXAce（以下 VXace）はスクリプトの編集が可能となっていて、編集にはプログラミングの技術を必要とするものの、ツクールユーザー（通称ツクラー）のゲーム作成の幅が無限に広がり、RPG ツクールを授業用教材とする大学が増加するようになった。

RPG ツクールは RPG 以外のジャンルも作成可能であり、シミュレーションゲームやシューティングゲームやパズルゲーム等がネット上に公開されている。アクションゲーム（以下 ACT）もあるが、その中でも横スクロール 2D アクションゲーム（以下 2DACT）は他のジャンルよりも少ない。RPG ツクールでの 2DACT 作品の作成難易度は基本的に高く、2DACT を作成するだけである場合は 2009 年から同社開発・販売されているアクションゲームツクールという ACT 等を作成する事に長けたソフトがあり、作成難易度や容量等の問題でアクションゲームツクールでの 2DACT 作成を辞退しない限り、RPG ツクールで 2DACT を作成する意味は無いと考えられる。そのため 2014 年 12 月現在での VXace 製 2DACT 作品の数は 2DACT 素材があるのみで、それは 1 つのサイトが公開している 2 種類のみであり、それらはどちらも基本的には同じ動き方をしている。

そこで私は、VXace で使用できる 2DACT スクリプト素材を改造し、若干の操作感変更と元の素材には無かったシステムを導入した。それを素材としてネット上に無料で公開する事で、一部のツクラーのゲーム作成の助けとなり、今後の VXace 製 2DACT 作品の発展に繋がると考えられる。

本論文では第 2 章では関連研究について述べる。第 3 章では使用した 2DACT スクリプト素材の紹介をして、第 4 章では改造した 2DACT スクリプトを紹介と説明をする。第 5 章ではまとめと今回作成した VXace の 2DACT スクリプト素材の利用規約について述べる。

第 2 章　関連研究

第 1 節　Ruby

Ruby とは、まつもとゆきひろ氏（Matz）によって開発されて誕生した「Perl のように手軽でストレス無くプログラミングできる」ことをコンセプトとしたオブジェクト指向のスクリプト言語であり、継承や Mix-in や、例外によるエラー処理や、自動的にメモリの解放を行うガベージコレクタ等の機能が備わっている。マルチプラットフォームなので、Windows や Unix 系の OS 等の様々なプラットフォームで作動し、それでいて多くのスクリプトは全く書き換えないで別のプラットフォームへ移してそのまま実行できる。オープンソースで開発されているフリーソフトウェアなので、ユーザーが自由に扱う事ができ、公式だけではなくユーザーも自由に修正や追加実装ができる[1]。

第 2 節　RGSS3

RGSS (Ruby Game Scripting System)とは、Ruby を PC 版 RPG ツクール用に拡張したスクリプト言語である。2004 年に発売された XP から RGSS が導入され、2007 年発売の VX で RGSS2 となり、2011 年発売の VXace で RGSS3 にバージョンアップされた。RGSS3 は Ruby1.9 がベースとなっていて、Ruby1.8 がベースだった XP の RGSS と VX の RGSS2 との必要スペックの差が激しくツクラーは、VXace も VX より必要スペックが劇的に高くなるのではと懸念されていたが、Ruby1.9 の導入による軽量化がされていて、必要スペックのインフレーションは緩やかになった。更に、RGSS2 以上に多数の機能が細部までメソッド化されており、メソッドを上書きする事が減り、メソッドに追加する形でスクリプト素材を追加した時の競合が少なくなった。RGSS はそれぞれ異なったバージョンとの互換性が無く、RGSS のスクリプトや、RGSS2 のスクリプトをそのまま RGSS3 のスクリプトに導入するとエラーが起き、RGSS は全てのバージョンにおいてフリーソフトウェアである Ruby とは違ってオープンソースでは無い為、ユーザーによる修正や実装はできない。

第 3 節　過去の RPG ツクール製の 2DACT

PC 版 RPG ツクールで 2DAC 作品が公開される事は TPRG ツクール 2000（以下 2000）の頃からあり、2DACT 作品が一番多く公開されているのも 2000 である。2000 はスクリプト編集ができないが、どのマップでも使用することができるコモンイベントにイベントを設定して 2DACT を作成できる。2000 製 2DACT は、気紛れな空間というサイトで 2002 年から公開されている『横アクションジャンプサンプル』や、ファミ通.com のツクールコミュニティで公開されている『ACTIV』等や、VIPRPG の祭りで提出される作品の中で時々見つかるアクション作品が挙げられる。

『横アクションジャンプサンプル』は敵キャラとなるイベントを踏んで倒すシステムがあり、マス移動バージョンとドット移動バージョンがある。説明書が入っていて、そこではイベント構

造の説明等がされている。『ACTIV』は爆弾を放って敵を倒したり、爆風を利用して飛んだりしてステージを攻略するゲームであり、RPG ツクールのデフォルト操作との切り替えをする場面がある。

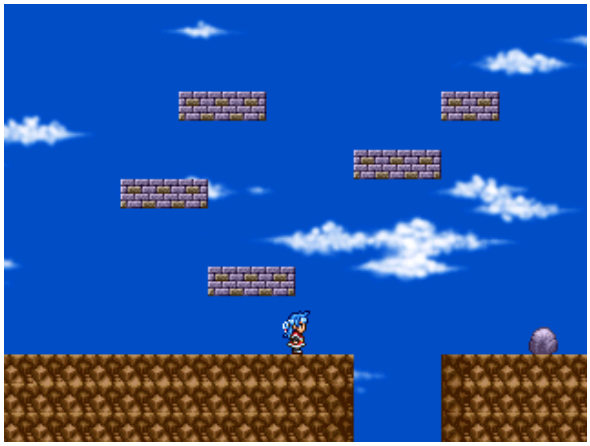


図 1　横アクションジャンプサンプル（ドット移動）

図 2　ACTIV

第 3 章　2DACT スクリプト素材の紹介

第 1 節　使用する素材について

今回使用するスクリプト素材はひきも記で公開されている『RGSS3_JUMPVer0.1a』と『RGSS3_ジャンプアクション Ver0.11a』である。サイト名に「閉鎖しました」とあるが、スクリプト素材は公開し続けていて、素材の修正もされている。[2]

RGSS3_ジャンプアクション Ver0.11a は最終更新日が 2012 年 1 月 23 日のものを使用し、RGSS3_JUMPVer0.1a は最終更新日が 2014 年 11 月 26 日のものを使用する。

RPG ツクールの初期のスクリプトでは 1 マス単位で移動するのに対しこれらのスクリプトを導入したゲームはドット単位の移動が可能であり、ジャンプの動作も滑らかである。画面左上にプレイヤーキャラクター（以下 PC）の体力ゲージがあり、体力が 0 になると体力のある他のメンバーと替わり、全員の体力が 0 になった時点でゲームオーバーである。RGSS3_ジャンプアクション Ver0.11a ではシューティングの要素があり、RGSS3_JUMPVer0.1a ではその要素が削られて水中表現と時計が取り入れられ、ダッシュでの NPC 押し出し等が追加されている。

図 3　ジャンプアクション Ver0.11a

図 4　JUMPVer0.1a

第 2 節　スクリプトの概要

本節では使用素材のスクリプトについて解説する。第 1 項ではクラス図の説明を、第 2 項ではジャンプアクション Ver0.11a と JUMPVer0.1a の両方で導入されている 2DACT の基本となるスクリプトの一部を、第 3 項ではジャンプアクション Ver0.11a のシューティングに関するスクリプトの一部を、第 4 項では JUMPVer0.1a のスクリプトの一部を紹介し、説明する。本節で紹介するスクリプトの著作権はひきも記の管理人の tomoaky 氏にある。

第 1 項　クラス図

今回使用するスクリプト素材のクラス図を、図 5 に示す。無色のクラスやメソッドやモジュ

ールはジャンプアクション Ver0.11a と JUMPVer0.1a の両方で編集・導入がされているものであり、赤色のクラスやメソッドはジャンプアクション Ver0.11a 側で編集・導入がされているものであり、青色のクラスやメソッドやモジュールは JUMPVer0.1a で編集・導入されているものである。

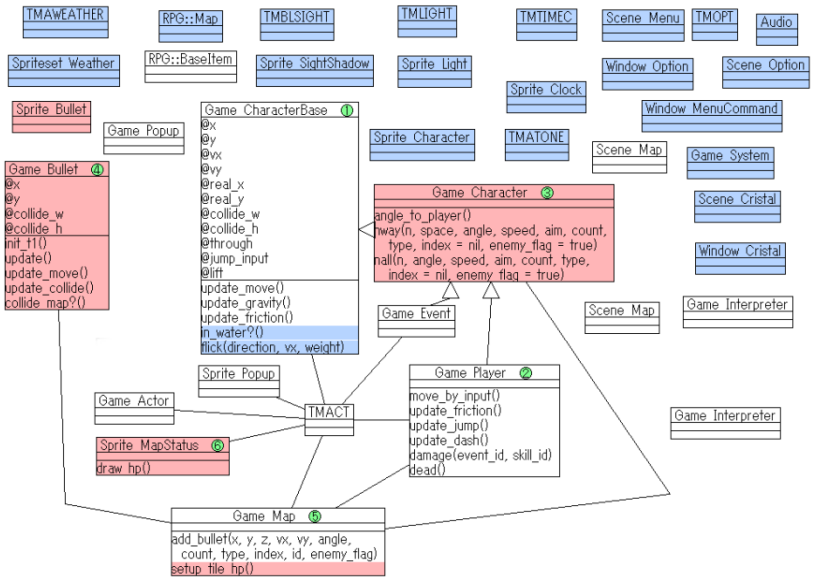


図 5　使用素材のクラス図

第 2 項　共通のスクリプト

本項では、重力や摩擦等を含めた移動関係の処理をするメソッドを含むクラスを詳しく説明する。

(1)Game_CharacterBase

①概要

Game_CharacterBase クラス（図 5 の 1 に位置を参照）はキャラクターを扱う基本のクラスであり、全てのキャラクターに共通する、座標やグラフィック等の基本的な情報の保持をする。このクラスのサブクラスとして Game_Character クラスと、そのサブクラスとして Game_Event クラスと Game_Player クラスがある。今回使用するスクリプトでは、フレーム毎にプレイヤーキャラクターの移動時の更新として、重力や摩擦の処理や、キャラクターやマップとの衝突判定を行いながらプレイヤーキャラクターの移動を行うメソッドがこのクラスの中で最も重要である（図 6 参照）。

②重力と摩擦の処理

まずは重力の処理で、プレイヤーキャラクターはしごにつかまっていない時は他メソッドの処理ではしごにつかまっているかを判定する@ladder が偽になり、その間は毎フレーム縦の速度の変数@vy の値に重力の変数@gravity の値が加算され、@vy が一定の数値（ジャンプアクション Ver0.11a の無編集状態なら 32768）を超えるまで常に繰り返される。次に摩擦の処理で、着地しているかどうかを判定する@landing が nil でなく、リフト属性のあるキャラクターに乗って

いる時に、リフト搭乗時の加算速度変数の@vx_plus が下のキャラクターと同じになる。ただし空中にプレイヤーキャラクターが居て他のメソッドの処理で@landing が nil になっている時、@xv_plus が 0 になり移動は通常どおり行われる。

③移動時の更新

重力と摩擦の処理が終われば 7 行目からキャラクターの座標移動処理に入り、横の速度変数の@vx 又は@vx_plus が 0 でない時に、それらの値がマップの横の実座標変数の@real_x に代入され、キャラが横に移動する。18 行目から 20 行目の縦方向移動は、空中に居て@vy が 0 でない時に縦の実座標@real_y が@vy の数値分増加して落下、又は減少して上昇する。9 行目から 14 行目と 21 行目から 26 行目は縦横の速度がある時に、キャラクターやマップとの接触判定が行われ、接触するとそれぞれの方向の接触判定メソッドが呼び出される。衝突判定の呼び出しの if が終われば@real_y と@real_x の値を 2 進数化して右から 15 桁削除した時の 10 進数が、1 マス単位の論理座標である縦座標@y と横座標@x に代入される。

```
1  #--------------------------------------------------------------------------
2  # ● 移動時の更新
3  #--------------------------------------------------------------------------
4  def update_move
5    update_gravity
6    update_friction
7    if @vx != 0 || @vx_plus != 0
8      @real_x += @vx + @vx_plus
9      if @vx > 0
10       collide_map_right?       # 右方向のマップ接触判定
11       collide_character_right? unless @through  # 右方向のキャラ接触判定
12     else
13       collide_map_left?        # 左方向のマップ接触判定
14       collide_character_left? unless @through   # 左方向のキャラ接触判定
15     end
16     @x = @real_x >> 15
17   end
18   if @vy != 0
19     @landing = nil
20     @real_y += @vy
21     if @vy > 0
22       collide_map_down?        # 下方向のマップ接触判定
23       collide_character_down? unless @through   # 下方向のキャラ接触判定
24     else
25       collide_map_up?          # 上方向のマップ接触判定
26       collide_character_up? unless @through     # 上方向のキャラ接触判定
27     end
28     @y = @real_y >> 15
29   end
30 end
31 #--------------------------------------------------------------------------
32 # ○ 重力の処理
33 #--------------------------------------------------------------------------
34 def update_gravity
35   return if @ladder
36   @vy = [@vy + @gravity, 32768].min
37 end
38 #--------------------------------------------------------------------------
39 # ○ 摩擦の処理
40 #--------------------------------------------------------------------------
41 def update_friction
42   if @landing
43     if @landing.is_a?(Game_Character) && @landing.lift
44       @vx_plus = @landing.vx
45     end
46   else
47     @vx_plus = 0
48   end
49 end
```

図 6　移動時の更新(ジャンプアクション Ver0.11a)

(2)Game_Player

①概要

Game_Player クラス（図 5 の 2 に位置を参照）は、プレイヤーを扱い、イベントの起動判定やマップのスクロール等の機能を持っているクラスであり、このクラスのインスタンスは $game_player で参照される。このクラスは、Game_Character クラスを経由して Game_CharacterBase を継承している。今回使用するスクリプトでは、ボタン入力によるプレイヤーキャラクターの移動処理が大幅に変更されていて、摩擦による減速処理やジャンプの処理も主にこのクラスでされる。

②ボタン入力による移動処理

まずは図 7 の方向ボタン入力による移動処理から説明する。5 行目の if は VXace の初期状態

の同メソッドと同様にイベント中は動けなくなっている。6 行目の処理は、L ボタンを、キーボードなら Q キーを押せばメンバーチェンジのメソッドが呼び出され、操作キャラが替わる。はしごにつかまっていて@ladder が真ならば上下ボタンを押して移動ができ、上下ボタンを押すと縦方向速度@vy がはしご移動最高速度@ladder_move_power を、上方向なら下回り、下方向なら上回るまで、1 フレーム毎にはしご移動加速度@ladder_accelerator の数値分だけ@vy に代入され加速する。移動カウント変数@move_count はスーパークラスである Game_CharacterBase の移動カウントの処理で毎フレーム 1 ずつ減少し、0 になれば横方向速度@vx が 0 になって失速し、はしごにつかまっているなら@vy も 0 になって失速するので、移動する為に@move_count 変数を 1 以上にする必要がある。このとき停止カウント変数@stop_count を 0 にするのは、この処理が無ければはしご移動の時この値が 1 以上になってしまい、キャラ画像が動かないままはしごを昇降するという不気味な動きをとってしまうからである。その後、昇降し終わってはしごから離れるとはしごから降りるメソッドが呼び出され@ladder が偽になる。@ladder が偽になっている時の左右移動は、ダッシュしていない時に左右のボタンを押す事ではしごにつかまっている時とほぼ同様の移動処理をする。21 行目と 25 行目の Set_direction は 4 なら左方向を、6 なら右方向をキャラクターが向く。30 行目以降は、はしごにつかまっていない状態で上下ボタンを押した場合の処理で、はしごと接触しているかを判定するメソッドが呼び出され、真になればはしごにつかまった状態になり get_on_ladder が真になる。

```
1  #--------------------------------------------------------------------------
2  # ● 方向ボタン入力による移動処理
3  #--------------------------------------------------------------------------
4  def move_by_input
5    return if $game_map.interpreter.running?
6    change_member if Input.trigger?(:L) # 操作キャラクター切り替え
7    if @ladder    # はしごにつかまっている
8      if Input.press?(:UP)
9        @vy = [@vy - @ladder_accelerator, -@ladder_move_power].max
10       @move_count = 4
11       @stop_count = 0
12     elsif Input.press?(:DOWN)
13       @vy = [@vy + @ladder_accelerator, @ladder_move_power].min
14       @move_count = 4
15       @stop_count = 0
16     end
17     get_off_ladder unless collide_ladder?
18   else          # つかまっていない
19     if @dash_count == 0
20       if Input.press?(:LEFT)          # 左移動
21         set_direction(4)
22         @vx = [@vx - @accelerator, -@move_power].max if @vx > -@move_power
23         @move_count = 4
24       elsif Input.press?(:RIGHT)      # 右移動
25         set_direction(6)
26         @vx = [@vx + @accelerator, @move_power].min  if @vx < @move_power
27         @move_count = 4
28       end
29     end
30     if Input.press?(:UP)            # つかまり（上入力）
31       get_on_ladder if collide_ladder?
32     elsif Input.press?(:DOWN)       # つかまり（下入力）
33       get_on_ladder(true) if collide_ladder_down?
34     end
35   end
36 end
```

図 7　方向ボタン入力による移動処理(ジャンプアクション Ver0.11a)

③摩擦の処理

次に摩擦による減速処理（図 8 参照）について説明する。図 6 と図 7 のメソッドだけでは障害物に当たるまで永遠に滑り続けるように動く。そこでこの摩擦の処理の出番である。このメソッドは図 6 の摩擦の処理メソッドにオーバーライドしていて、それを 5 行目で呼び出している。6 行目以降はしごにつかまっている時とそうでない時に分かれる条件分岐がある。つかまっている時は TMACT モジュールの摩擦の強さの定数 FRICTION_DEFAULT を n に代入し、プレイヤーキャラクターが動いている時に n から FRICTION_DEFAULT を引いた正数を、それが負

数なら 0 を n に代入する。降下していて縦の移動速度@vy が 0 より大きければ、0 になるまで@vy から n の分が減算され、逆に上昇していれば 0 になるまで n の分が加算され減速する。11 行目以降ははしごにつかまっていない時のしょりになり、ダッシュ中でなければ横の速度@vx が、12 行目の場合は左方向に、13 行目の場合は右方向に移動していて、最大速度@move_power を上回っていれば、@vx に 256 だけ減速した数値か@move_power の、どちらか 0 から遠い方が@vx に代入される。16 行目から 20 行目は着地しているかしていないかの条件分岐で、着地していなくて@landing が nil の時は摩擦の処理が終わり、着地していて@landing が nil でなければ、16 行目でマップの上に乗っていて@landing に[x,y]の値が代入されている場合、17 行目でその座標のリージョン ID に対応した TMACT モジュールの定数 FRICTION_REGION の値（50 番のリージョンなら 0 を、51 番のリージョンなら 32768）を n に代入する。@landing が nil と[x,y]以外なら、FRICTION_DEFAULT を n に代入する。それが終われば、動いている時に n から FRICTION_DEFAULT が引かれた数値か 0 のどちらか大きい方を代入し、右に動いていて@vx が 0 より大きければ@vx から n が引かれた数値か 0 のどちらか大きい方が@vx に代入され、左に動いていればその逆が代入される。しかし、この時の摩擦の値 n が@vx より高くない限り、はしごにつかまっているなら上下の方向ボタンを、地上なら左右の方向ボタンを押し続けている限り図 6 と図 7 のメソッドにより、このメソッドによって減速が発生しない。

```
1  #--------------------------------------------------------------------------
2  # ● 摩擦の処理
3  #--------------------------------------------------------------------------
4  def update_friction
5    super
6    if @ladder
7      n = TMACT::FRICTION_DEFAULT
8      n = [n - TMACT::FRICTION_DEFAULT, 0].max if moving?
9      @vy = @vy > 0 ? [@vy - n, 0].max : [@vy + n, 0].min if @vy != 0
10   else
11     unless dash?
12       @vx = [@vx + 256, -@move_power].min if @vx < -@move_power
13       @vx = [@vx - 256, @move_power].max if @vx > @move_power
14     end
15     if @landing
16       if @landing.is_a?(Array)
17         n = TMACT::FRICTION_REGION[$game_map.region_id(*@landing)]
18       else
19         n = TMACT::FRICTION_DEFAULT
20       end
21       n = [n - TMACT::FRICTION_DEFAULT, 0].max if moving?
22       @vx = @vx > 0 ? [@vx - n, 0].max : [@vx + n, 0].min if @vx != 0
23     end
24   end
25 end
```

図 8　摩擦による減速処理(ジャンプアクション Ver0.11a)

④ジャンプの更新

次にジャンプの処理（図 9 参照）について説明する。5 行目から 12 行目までは、ジャンプボタン（キーボードなら S キー）入力の残り時間（単位：フレーム）変数@jump_input が 1 以上あるかどうかの条件分岐があり、あるなら 1 減算され、もし、まだジャンプボタンが押されていれば縦速度変数@vy がジャンプ力変数@jump_power の数値分減少し、プレイヤーキャラクターが上昇する。ジャンプボタンの入力をやめれば、ジャンプボタン入力残り時間が 0 になって落下する。13 行目からはジャンプボタンが押されたかどうかの条件分岐があり、押されなければこのメソッドの処理が終わり、押されると残りジャンプ回数@jump_count が 1 以上あればそれを 1 減少させて次の処理へ向かう。@jump_count が 0 以下なら壁ジャンプの処理に入る。壁ジャンプフラグ@wall_jump が nil や false であればこのメソッドの処理が終わり、そうでなければ、向き変数@direction が 4 即ち左を向いているなら横の実座標@real_x から横の当たり判定

@collide_w と 8192 を引いた数だけ引いた数値を、@direction がそれ以外、即ち右を向いているならそれらの数値が加算された数値を、2 進数化して 15 桁右にシフトして 10 進数化し、その数値を x に代入する。y には現在の縦の実座標@real_y の 2 進数化したものを 15 桁右にシフトした 10 進数が代入される。21 行目で壁ジャンプが可能な壁かどうかを判定し、可能なら壁から跳ね返るかのように横速度が壁側と反対方向にプレイヤーキャラクターが跳んでいき、ここでジャンプ回数の条件分岐が終了する。その後、はしごにつかまっていればはしごから手を離し、新たにジャンプされたのでジャンプ入力残り時間にジャンプ入力最大時間が代入される。ダッシュ中ならダッシュの時間延長をして、それから@vy が@jump_power の数値分減少して跳ぶ処理に入り、姿勢の矯正処理がされる。

```
1  #--------------------------------------------------------------------------
2  # ○ ジャンプの更新
3  #--------------------------------------------------------------------------
4  def update_jump
5    if @jump_input > 0                # ジャンプの追加入力
6      @jump_input -= 1
7      if Input.press?(:Y)
8        @vy = -@jump_power
9      else
10       @jump_input = 0
11     end
12   end
13   if Input.trigger?(:Y)             # ジャンプ
14     if @jump_count > 0
15       @jump_count -= 1
16     else
17       return unless @wall_jump
18       x = (@direction == 4 ? @real_x - @collide_w - 8192 :
19         @real_x + @collide_w + 8192) >> 15
20       y = @real_y >> 15
21       return unless $game_map.can_wall_jump?(x, y, @direction)
22       @vx = @direction == 4 ? @move_power : -@move_power
23       @direction = reverse_dir(@direction)
24     end
25     get_off_ladder if @ladder
26     @jump_input = @jump_input_time
27     if dash?                        # ダッシュ中ならダッシュ時間の延長
28       @dash_count = @dash_count_time
29       @vx = @direction == 4 ? -@dash_power_x : @dash_power_x
30     end
31     @vy = -@jump_power
32     @stop_count = 0
33     straighten
34   end
35 end
```

図 9　毎フレーム行われるジャンプの更新(ジャンプアクション Ver0.11a)

⑤ダッシュの更新

次に、本項で度々出していたダッシュについて説明する（図 10 参照）。もしダッシュの残り時間@dash_count が 1 以上あればジャンプの時と同様に毎フレーム 1 ずつ減少する。ダッシュ後の硬直時間@dash_delay が 1 以上あればこれも毎フレーム 1 ずつ減少して、9 行目で@dash_delay が 0 以下になれば、ダッシュが禁止されていないマップでダッシュボタン（キーボードの Shift キー）を押してダッシュができる。はしごにつかまっていればはしごから離れ、@dash_count と@dash_delay にそれぞれの最大時間を代入する。13 行目でプレイヤーキャラクターが左を向いていればダッシュ力@dash_power_x をマイナスにして横の速度@vx に代入し、それ以外の方向をむいていれば@dash_power_x をそのまま@vx に代入して加速する。縦の速度@vy も縦のダッシュ力@dash_power_y をマイナスした値が代入され、少々キャラクターが浮く。移動時間@move_count にダッシュ時間の半分が代入され、停止時間@stop_count は 0 になり姿勢の矯正の処理が呼び出される。

```
#--------------------------------------------------------------------------
# ○ ダッシュの更新
#--------------------------------------------------------------------------
def update_dash
  @dash_count -= 1 if @dash_count > 0
  if @dash_delay > 0
    @dash_delay -= 1
  else
    if Input.trigger?(:A) && !$game_map.disable_dash?
      get_off_ladder if @ladder
      @dash_count = @dash_count_time
      @dash_delay = @dash_delay_time
      @vx = @direction == 4 ? -@dash_power_x : @dash_power_x
      @vy = -@dash_power_y
      @move_count = @dash_count / 2
      @stop_count = 0
      straighten
    end
  end
end
```

図 10　毎フレーム行われるダッシュの更新(ジャンプアクション Ver0.11a)

第 3 項　ジャンプアクション Ver0.11a

本項では、シューティング関連の処理とダメージの処理を扱うメソッドを含むクラスを詳しく説明する。

(1)Game_Player

①概要

このクラスは移動関連の処理の他に、プレイヤーキャラクター側のダメージの処理を行うメソッドが導入されている。ダメージの処理はジャンプアクション Ver0.11a と JUMPVer0.1a で大幅な違いがあるので本項でジャンプアクション Ver0.11a 側のダメージ処理（図 11 参照）を説明する。

②被弾時のダメージの処理（10 行目まで）

まず、このメソッドの引数の event_id が 0 以下であるか、プレイヤーキャラクターが被弾後の無敵状態であり through?が 1 以上ある場合、プレイヤーキャラクターはダメージを受けずこのメソッドの処理を終える。メソッド処理を続ける場合、user に接触イベント ID を入れた Game_Map クラスの変数 events を代入し、item にデータベースのスキルにある、このメソッドの引数@skill_id 番のスキルが代入される。その後、対応するアクターの取得メソッドの結果がリセットされる。9 行目でクリティカルヒットの判定をして、10 行目で Game_Battler クラスのダメージ計算が行われ、11 行目で Game_Battler クラスのダメージ処理が行われる（図 12）。

③ダメージ計算とダメージ処理

図 12 のメソッドは、VXace の初期状態のものである。ダメージ処理の引数は、 user にスキルやアイテムの使用者を表す Game_Battler オブジェクトが、item にデータベースで指定したスキルやアイテムのオブジェクトが渡される。5 行目で使用者 user（敵キャラクター）、対象者 self（プレイヤーキャラクター）、ゲーム内変数の配列$game_variables を引数として item.damage.eval を呼び出していて、それを value に代入し、後に属性耐性や防御やクリティカル等のダメージ修正が続く。ダメージの処理では、@result.hp_damage がなら、その変数を引数として被ダメージ時の処理メソッドを呼び出し、プレイヤーキャラクターはダメージを受ける[3]。

④被弾時のダメージの処理（12 行目以降）

処理は図 11 のダメージ計算に戻り、12 行目で、横座標に@real_x、縦座標に@collide_h の半分の値を引かれた@real_y、ダメージには図 12 のダメージ処理の 8 行目時点での

@result.hp_damage、敵かどうか判断する@enemy_flag は false を引数として、Game_Map クラスのポップアップの追加メソッドを呼び出し、ダメージ値を画像表示する。その後 14 行目でプレイヤーキャラクターに戦闘アニメ（キャラクターが赤くなる）が再生され、次に被弾後の無敵時間をセットし、最後に操作キャラクターの HP が 0 になり戦闘不能状態になれば戦闘不能の処理をする。戦闘不能処理は、まだ他に生存キャラクターが居れば入れ替えをして、全滅していれば全滅した時のコモンイベントを呼び出すのだが、先に他のクラスのゲームオーバー処理が優先されていて 23 行目の意味が無いという問題がある。これに関しては第 4 章で修正を行う。

```
1  #--------------------------------------------------------------------------
2  # ○ ダメージの処理
3  #--------------------------------------------------------------------------
4  def damage(event_id, skill_id)
5    return if event_id < 1 || through?
6    user = $game_map.events[event_id]
7    item = $data_skills[skill_id]
8    actor.result.clear
9    actor.result.critical = (rand < actor.item_cri(user, item))
10   actor.make_damage_value(user, item)
11   actor.execute_damage(user)
12   $game_map.add_popup(@real_x, @real_y - @collide_h / 2,
13     actor.result.hp_damage, false)
14   @animation_id = TMACT::ANIME_DAMAGE_PLAYER
15   @through_count = @through_count_max   # 被弾後無敵時間のセット
16   dead if actor.dead?
17 end
18 #--------------------------------------------------------------------------
19 # ○ 戦闘不能の処理
20 #--------------------------------------------------------------------------
21 def dead
22   if $game_party.all_dead?
23     $game_temp.reserve_common_event(TMACT::ALL_DEAD_EVENT)
24   else
25     change_member
26   end
27 end
```

図 11　ダメージの処理

```
1  #--------------------------------------------------------------------------
2  # ● ダメージ計算
3  #--------------------------------------------------------------------------
4  def make_damage_value(user, item)
5    value = item.damage.eval(user, self, $game_variables)
6    value *= item_element_rate(user, item)
7    value *= pdr if item.physical?
8    value *= mdr if item.magical?
9    value *= rec if item.damage.recover?
10   value = apply_critical(value) if @result.critical
11   value = apply_variance(value, item.damage.variance)
12   value = apply_guard(value)
13   @result.make_damage(value.to_i, item)
14 end
```

```
1  #--------------------------------------------------------------------------
2  # ● ダメージの処理
3  #    呼び出し前に @result.hp_damage @result.mp_damage @result.hp_drain
4  #    @result.mp_drain が設定されていること。
5  #--------------------------------------------------------------------------
6  def execute_damage(user)
7    on_damage(@result.hp_damage) if @result.hp_damage > 0
8    self.hp -= @result.hp_damage
9    self.mp -= @result.mp_damage
10   user.hp += @result.hp_drain
11   user.mp += @result.mp_drain
12 end
```

図 12　Game_Battler クラスのダメージ計算と処理

⑵Game_Character

①概要

Game_Character クラス（図 5 の 3 に位置を参照）は、主に移動ルートなどの処理を追加したキャラクターのクラスで、Game_Player や Game_Event 等 のスーパークラスとして使用される。今回使用するスクリプトでは、敵キャラクターのショットの方向等を決めるメソッドが導入されている。

②敵ショット

n 方向と全方向のショットの引数は、n がショットの方向数で、space が弾の間隔（角度）で、angle が弾の発射角度で、speed は弾速、aim はプレイヤーキャラクターを狙うかを決め、count は弾が消えるまでの時間で、type は弾のタイプで、index は呼び出さず、敵の弾かを判断する @enemy_flag は真となっている。まず、n 方向と全方向の両方は、弾の発射角度引数 angle を a に代入し、そこから引数 aim が nil や false 以外なら弾を発射する敵キャラクターから見たプレイヤーキャラクターの居る方向（角度）を angle_to_player メソッドで算出し、それを a に加える。n 方向ショットの 14 行目は図 14 のような a を中心とした扇状の弾幕を張る為の角度調整をして、全方向ショットの 28 行目は 360 度を方向数で除算して間隔を作る。n 方向ショットの 15 行目以降と全方向ショットの 29 行目以降は、i が 0 から方向数 n になるまで、Game_Map クラスの弾の追加を行うメソッドを呼び出し、間隔@space の数値分弾の角度が加算される。

```ruby
#--------------------------------------------------------------------------
# ○ プレイヤーのいる方向（角度）を取得
#--------------------------------------------------------------------------
def angle_to_player
  Math.atan2($game_player.real_y - $game_player.collide_h / 2 -
    (@real_y - @collide_h / 2), $game_player.real_x - @real_x)
end
#--------------------------------------------------------------------------
# ○ n方向ショット
#--------------------------------------------------------------------------
def nway(n, space, angle, speed, aim, count, type, index = nil, enemy_flag = true)
  a = angle
  a += angle_to_player if aim
  a = a - ( space * ( n - 1 ) / 2 )
  (0...n).each do |i|
    $game_map.add_bullet(@real_x, @real_y - @collide_h / 2, 200 + i,
      (Math.cos(a) * speed).to_i, (Math.sin(a) * speed).to_i,
      a, count, type, index, @id, enemy_flag)
    a += space
  end
end
#--------------------------------------------------------------------------
# ○ 全方位ショット
#--------------------------------------------------------------------------
def nall(n, angle, speed, aim, count, type, index = nil, enemy_flag = true)
  a = angle
  a += angle_to_player if aim
  space = Math::PI * 2 / n
  (0...n).each do |i|
    $game_map.add_bullet(@real_x, @real_y - @collide_h / 2, 200 + i,
      (Math.cos(a) * speed).to_i, (Math.sin(a) * speed).to_i,
      a, count, type, index, @id, enemy_flag)
    a += space
  end
end
```

図 13　敵ショットの発射前の処理

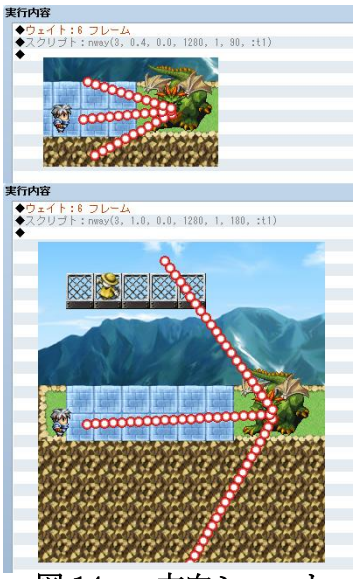


図 14　n 方向ショット

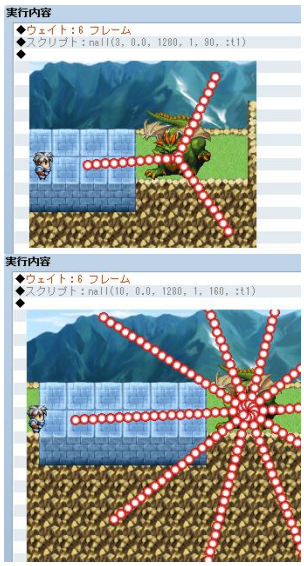


図 15　全方向ショット

(3)Game_Bullet

①概要

Game_Bullet クラス（図 5 の 4 の位置を参照）は、弾を扱うクラスで、主に弾のタイプを設定したり、弾の移動処理をしたり、衝突判定の処理をしたりする。

②弾タイプの設定

まずは弾のタイプ設定（図 16 参照）の説明をする。図 14 と図 15 のイベントコマンドの 2 行目の最後に:t1 と入力されているのは、図 16 の 7 行目から、タイプ 1 のメソッドを呼び出す為である。弾タイプのメソッドの処理は、1 行目で Graphics ファイル内の System ファイル内にある画像をファイル名で指定して、2 行目と 3 行目で当たり判定の広さを設定する。次に 4 行目で被弾時に発動するスキルを決める。これは、@skill_id に代入されている数値を、データベースのスキルのスキル番号に合わせていて、データベースのスキルが未編集の場合、1 番のスキルは攻撃（通常攻撃）である。このアクションスクリプトでの被弾時に発動するスキルは図 17 の赤い丸で囲まれているダメージ処理のみが適応され、他の設定は無視される。ダメージタイプが HP（MP）ダメージなら被弾者の HP（MP）が減少し、HP（MP）回復なら被弾者の HP（MP）

が回復し、HP（MP）吸収なら被弾者の HP（MP）を、その弾を発射したキャラクターの HP（MP）に加える。ダメージ計算式も分散度も属性も会心があるかどうかも適応される。16 行目の@map_collide は、弾がマップを貫通するかどうかを返し、真なら貫通せず偽なら貫通する。

```
1 #==============================================================================
2 # □ Game_Bullet
3 #==============================================================================
4 class Game_Bullet
5   # タイプごとの呼び出し設定
6   HANDLER = {}
7   HANDLER[:t1] = [:init_t1, :update_t1]    # タイプ1
8   #--------------------------------------------------------------------------
9   # ○ タイプ1
10  #--------------------------------------------------------------------------
11  def init_t1
12    @character_name = "Bullet_0"      # グラフィックのファイル名
13    @collide_w = 6                    # 当たり判定の幅（画像中心から辺までの距離）
14    @collide_h = 6                    # 当たり判定の高さ（画像中心から辺までの距離）
15    @skill_id = 1                     # スキルID
16    @map_collide = true               # マップとの衝突フラグ
17  end
18  def update_t1
19  end
20 end
```

図 16　弾タイプの設定

図 17　データベースのスキル

③フレーム処理

次に、フレーム処理（図 18 参照）について説明する。まずは、移動処理の更新を行い、弾の移動をする。この時の@x と@y は、キャラクターが移動する際の理論座標@x と@y とは違い、移動速度@vx と@vy 基準になっている。この@vx と@vy は Game_Map クラスの弾の追加メソッドで設定されている引数を使用する。12 行目で、図 16 のメソッド、弾タイプ 1 なら init_t1 のメソッドが呼び出され、結果が返される。13 行目で接触判定を行い、弾が消えるまでの時間が 1 フレーム減少し、それが 0 になればその弾が消える。

④弾の接触判定

弾の接触判定（図 19 参照）は、まず、マップとの接触判定を行い、図 16 の弾タイプのメソッドで設定したマップとの衝突フラグが真で、弾がマップと接触すると弾が消滅するが、弾が接触したマップオブジェクトにステータスが振られている場合、そのマップオブジェクトにダメージを与えるメソッドが呼び出され、このメソッドの処理が終了する。マップに接触しなかったら、次の敵の弾かどうかの条件分岐に入り、それが真であり、プレイヤーに弾が接触した場合、図 11 のメソッドが呼び出され、プレイヤーキャラクターがダメージを受け、弾が消えてメソッド処理が終了する。敵の弾かどうかが偽であって敵キャラクターとするイベントに接触した場合、その敵キャラクターにダメージを与えて弾が消滅してメソッド処理が終了する。

```ruby
#--------------------------------------------------------------------------
# ○ オブジェクトが存在しているかどうか
#--------------------------------------------------------------------------
def exist?
  @character_name != ""
end
#--------------------------------------------------------------------------
# ○ フレーム更新
#--------------------------------------------------------------------------
def update
  update_move
  send(HANDLER[@type][1])
  update_collide
  @count -= 1
  erase if @count == 0
  exist?
end
#--------------------------------------------------------------------------
# ○ 移動処理の更新
#--------------------------------------------------------------------------
def update_move
  @x += @vx
  @y += @vy
end
```

図 18　弾のフレーム更新

```ruby
#--------------------------------------------------------------------------
# ○ マップと接触しているかどうかを返す
#--------------------------------------------------------------------------
def collide_map?
  !$game_map.check_passage_bullet(@x >> 15, @y >> 15)
end
#--------------------------------------------------------------------------
# ○ 接触判定
#--------------------------------------------------------------------------
def update_collide
  if @map_collide && collide_map?  # マップとの接触判定
    $game_map.tile_damage(@x >> 15, @y >> 15)
    erase
    return
  end
  if @enemy_flag
    if collide?($game_player)
      $game_player.damage(@event_id, @skill_id)
      erase
    end
  else
    $game_map.events.values.each do |event|
      next unless collide?(event)
      event.damage(@event_id, @skill_id)
      erase
      break
    end
  end
end
```

図 19　弾の接触判定

(4)Game_Map

①概要

Game_Map クラス（図 5 の 5 の位置を参照）は、マップを扱い、スクロールや通行可能判定などの機能を持っているクラスである。このクラスのインスタンスは $game_map で参照される。ジャンプアクション Ver0.11a では、マップの耐久力セットや弾の追加等の処理を行うメソッドが追加されている。

②弾の追加メソッド

まず、弾の追加メソッド（図 20 参照）を説明する。この弾の追加メソッドは、図 13 の敵ショットのメソッドから呼び出される。まず、敵の弾かプレイヤーキャラクターの弾かの条件分岐があり、敵の弾（true）の場合、敵の弾の表示できる残りの数@blank_enemy_bullets が 0 でなければ、その変数を i に入れ、@enemy_bullets[i].setup オブジェクト（Game_Bullet クラスのセットアップメソッド）に、図 13 の敵ショットの処理で呼び出された時のこのメソッドの引数を、敵フラグを真にしてそのオブジェクトの引数に代入する。敵弾表示数配列 @alive_enemy_bullets に i を追加する。プレイヤーキャラクターのショットにあたる 11 行目以降にも、同様の処理がされる。

```ruby
#--------------------------------------------------------------------------
# ○ 弾の追加
#--------------------------------------------------------------------------
def add_bullet(x, y, z, vx, vy, angle, count, type, index, id, enemy_flag)
  if enemy_flag
    return if @blank_enemy_bullets.empty?
    i = @blank_enemy_bullets.shift
    @enemy_bullets[i].setup(x, y, z, vx, vy, angle, count, type, index, id, true)
    @alive_enemy_bullets.push(i)
  else
    return if @blank_player_bullets.empty?
    i = @blank_player_bullets.shift
    @player_bullets[i].setup(x, y, z, vx, vy, angle, count, type, index, id, false)
    @alive_player_bullets.push(i)
  end
end
```

図 20　弾の追加処理

③マップ耐久力のセットアップ

次にマップ耐久力のセットするメソッド（図 21 参照）を説明する。まず、マップオブジェクトの HP に空の配列を生成する。6 行目から 10 行目まではループを利用した、マップの全てのマス（タイル）のチェックを行い、TMACT モジュールで設定された定数 REGION_HP[]（[] の中にはリージョン ID が入る）で指定されたリージョン ID のあるタイルに、REGION_HP[] に代入されている HP を代入する。その通行不可オブジェクトは、弾を当てる事で図 19 の弾の

接触判定処理によって消滅する。

```
#--------------------------------------------------------------------------
# ○ マップ耐久力のセットアップ
#--------------------------------------------------------------------------
def setup_tile_hp
  @tile_hp = []
  width.times do |x|
    height.times do |y|
      @tile_hp[[x, y]] = TMACT::REGION_HP[region_id(x, y)]
    end
  end
end
```

図 21　マップ耐久力のセット

(5)Sprite_MapStatus

①概要

Sprite_MapStatus クラス（図 5 の 6 の位置を参照）は、主に画面左上の HP の描画（図 3 参照）と更新を行うクラスである。なお、ジャンプアクション Ver0.11a と JUMPVer0.1a ではマップ上の HP 描画方法に差異があり、JUMPVer0.1a にはこのクラスは無い。

②HP 描画

まずは HP 描画メソッド（図 22 参照）の説明をする。まず、Rect オブジェクト（引数は x=0, y=TMACT::MAP_STATUS_HIGHT-24, width=96, hight=24）を生成し、変数 rect に代入し、次の行から描画処理に入る。まず、変数 rect と同じ四角形の画像をクリアしてから、操作プレイヤーキャラクターの顔アイコン（x=0, y= TMACT::MAP_STATUS_HIGHT-24, src_bitmap=@face_bitmap, src_rect=rect）を描画する。その後、変数 w に現 HP の 94 倍を最大 HP で割った値を代入し、HP バー（x=1, y= TMACT::MAP_STATUS_HIGHT-7, width=w, hight=6,color(red=64, green=224, blue=64, alpha=224)）を表示する。11 行目でフォント名設定をする。アリアル Black が一番希望となって優先的に設定されるが、それがシステム上存在していなければ VL ゴシックが選択される。次に変数 rect の要素を変更し、最後に現在の HP の値を rect の範囲内の右揃えで描写する。

```
#--------------------------------------------------------------------------
# ○ ＨＰを描画
#--------------------------------------------------------------------------
def draw_hp
  rect = Rect.new(0, TMACT::MAP_STATUS_HEIGHT - 24, 96, 24)
  self.bitmap.clear_rect(rect)
  self.bitmap.blt(rect.x, rect.y, @face_bitmap, rect)
  w = @hp * 94 / @mhp
  self.bitmap.fill_rect(1, TMACT::MAP_STATUS_HEIGHT - 7,
    w, 6, Color.new(64, 224, 64, 224))
  self.bitmap.font.name = ["Arial Black", "VL Gothic"]
  rect.set(4, TMACT::MAP_STATUS_HEIGHT - 24, 88, 24)
  self.bitmap.draw_text(rect, @hp, 2)
end
```

図 22　HP の描写

第 4 項　JUMPVer0.1a

本項では、水中処理やダッシュ押し出しの処理を詳しく説明する。

(1)水中表現

①概要

水中表現は JUMPVer0.1a のシステムの一つで、一言で表すと水中として扱うリージョンにプレイヤーキャラクターが入ると動きが全体的に鈍り、ダッシュできなくなるが、泳ぐように無限ジャンプができるものである。この処理を行うメソッドを含むクラスが複数ある。

②Game_CharacterBase

Game_CharacterBase クラス（図 5 の 1 の位置を参照）には水中にいるかどうかを判定する

メソッド（図 23 参照）がある。これは、プレイヤーキャラクターが TMACT クラスで設定した水中として扱うリージョン ID のあるタイルに触れている場合、このメソッドは真になり、そうでなければ偽になる。

```
#--------------------------------------------------------------------------
# ○ 水中にいるかどうかを返す
#--------------------------------------------------------------------------
def in_water?
  $game_map.region_id(@x, @y) == TMACT::WATER_REGION
end
```

図 23　水中にいるかを判定するメソッド

③Game_Player

Game_Player クラス（図 5 の 2 の位置を参照）の、ジャンプの更新と摩擦の更新と重力の処理に水中処理が導入されている。

③-①ジャンプの更新

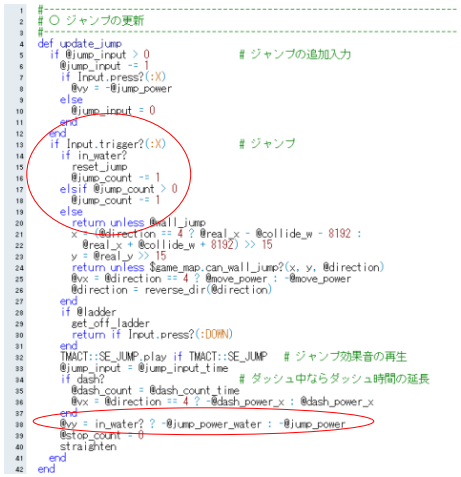
ジャンプの更新（図 24 参照）は、14 行目から 18 行目までに水中に居るか居ないかの条件分岐が追加されており、水中フラグが真ならば常にジャンプカウントが最大まで回復してそこから -1 され、偽なら通常通りジャンプカウントが 1 つ減少する。そして、ジャンプした時に 38 行目で水中フラグが真なら水中ジャンプ力@jump_power_water の数値分プレイヤーキャラクターが上昇する。この変数は通常のジャンプ力@jump_power よりも大幅に低く設定されており、上昇量も約 1.5 マス程度となっている。水中フラグが偽なら通常のジャンプで上昇する。

③-②摩擦の処理

摩擦の処理（図 25 参照）は、12 行目から 14 行目に水中に居る場合の処理がある。水中フラグが真なら、水中での移動力@move_power_water を変数 mp 代入し、偽なら通常の移動力での処理になる。水中移動力は通常の移動力よりも低く設定されている。

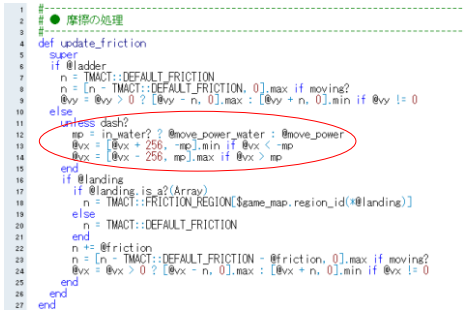
③-③重力の処理

重力の処理（図 26 参照）は、はしごにつかまっていない時に水中に居て水中フラグが真ならば、TMACT モジュールの水中落下最高速度 MAX_FALL_SPEED_WATER が n に代入され、落下速度が通常の落下より低下する。



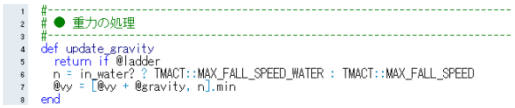
```
#--------------------------------------------------------------------------
# ○ ジャンプの更新
#--------------------------------------------------------------------------
def update_jump
  if @jump_input > 0                  # ジャンプの追加入力
    @jump_input -= 1
    if Input.press?(:X)
      @vy = -@jump_power
    else
      @jump_input = 0
    end
  end
  if Input.trigger?(:X)               # ジャンプ
    if in_water?
      reset_jump
      @jump_count -= 1
    elsif @jump_count > 0
      @jump_count -= 1
    else
      return unless @wall_jump
      x = (@direction == 4 ? @real_x - @collide_w - 8192 :
        @real_x + @collide_w + 8192) >> 15
      y = @real_y >> 15
      return unless $game_map.can_wall_jump?(x, y, @direction)
      @vx = @direction == 4 ? @move_power : -@move_power
      @direction = reverse_dir(@direction)
    end
    if @ladder
      set_off_ladder
      return if Input.press?(:DOWN)
    end
    TMACT::SE_JUMP.play if TMACT::SE_JUMP   # ジャンプ効果音の再生
    @jump_input = @jump_input_time
    if dash?                          # ダッシュ中ならダッシュ時間の延長
      @dash_count = @dash_count_time
      @vx = @direction == 4 ? -@dash_power_x : @dash_power_x
    end
    @vy = in_water? ? -@jump_power_water : -@jump_power
    @stop_count = 0
    straighten
  end
end
```

図 24　ジャンプの更新(JUMPVer0.1a)



```
#--------------------------------------------------------------------------
# ● 摩擦の処理
#--------------------------------------------------------------------------
def update_friction
  super
  if @ladder
    n = TMACT::DEFAULT_FRICTION
    n = [n - TMACT::DEFAULT_FRICTION, 0].max if moving?
    @vy = @vy > 0 ? [@vy - n, 0].max : [@vy + n, 0].min if @vy != 0
  else
    unless dash?
      mp = in_water? ? @move_power_water : @move_power
      @vx = [@vx + 256, -mp].min if @vx < -mp
      @vx = [@vx - 256, mp].max if @vx > mp
    end
    if @landing
      if @landing.is_a?(Array)
        n = TMACT::FRICTION_REGION[$game_map.region_id(x@landing)]
      else
        n = TMACT::DEFAULT_FRICTION
      end
      n += @friction
      n = [n - TMACT::DEFAULT_FRICTION - @friction, 0].max if moving?
      @vx = @vx > 0 ? [@vx - n, 0].max : [@vx + n, 0].min if @vx != 0
    end
  end
end
```

図 25　摩擦の処理(JUMPVer0.1a)



```
#--------------------------------------------------------------------------
# ● 重力の処理
#--------------------------------------------------------------------------
def update_gravity
  return if @ladder
  n = in_water? ? TMACT::MAX_FALL_SPEED_WATER : TMACT::MAX_FALL_SPEED
  @vy = [@vy + @gravity, n].min
end
```

図 26　重力の処理(JUMPVer0.1a)

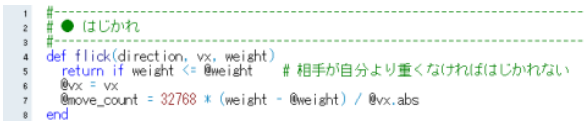
(2)ダッシュ押し出し

①概要

ダッシュ押し出しとは、プレイヤーキャラクターがダッシュした時、前方のキャラクターに前方から当たれば相手を突き飛ばす事である。このシステムが導入されているのは Game_CharacterBase のみである。

②Game_CharacterBase

Game_CharacterBase（図 5 の 1 の位置を参照）の、はじかれメソッド（図 27 参照）は、左右のキャラクターとの衝突判定で、ダッシュ中フラグが真の場合呼び出されるメソッドである。プレイヤーキャラクターの重さのステータス（@weight）が相手のキャラクターの重さのステータス（引数 weight）以上の時に、横速度@vx に引数 vx が代入され、移動カウント@move_count に、相手の重さから自分の重さを引いた数値を 32768 倍にした数値を、@vx の絶対値を割った数値を代入し、相手は勢いよく弾かれる。



```
#--------------------------------------------------------------------------
# ● はじかれ
#--------------------------------------------------------------------------
def flick(direction, vx, weight)
  return if weight <= @weight    # 相手が自分より重くなければはじかれない
  @vx = vx
  @move_count = 32768 * (weight - @weight) / @vx.abs
end
```

図 27　はじかれメソッド

第 4 章　2DACT スクリプトの改造

第 1 節　スクリプトの変更の概要

本論文で、スクリプト素材を改造するにあたって変更する点は、ジャンプアクション Ver0.11a のスクリプトをベースとして、以下のシステムを修正・追加実装を行った。

(1)アクション移動システム変更

移動加速度を撤廃し、空中でも摩擦の影響を受けるように修正を行った。このシステムは Game_CharacterBase と Game_Player と Game_Map を改造して変更した。

図 28　水中を往くプレイヤーキャラクター

(2)RPG 操作との切り替え

アクションスイッチによって RPG 操作との切り替えができるシステムを追加した。このシステムは Game_Map に導入していて、TMACT モジュールがアクションスイッチを扱う。

図 29　操作モード切り替え

(3)アクションの被弾により全滅しても直接ゲームオーバーにはならないでコモンイベントを呼び出す処理

直接ゲームオーバーになる原因を断つ事によって全滅時コモンイベントを実行できるように修正を行った。このシステムは Scene_Base に導入している。

図 30　全滅時コモンイベントによって爆発するプレイヤーキャラクター

(4)画面外へ出た時の処理

画面外へプレイヤーキャラクターがでてしまった場合、救済処置として全滅時コモンイベントを実行するシステムを追加した。このシステムは、Game_Map に導入している。

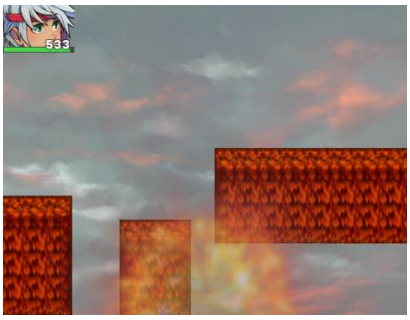


図 31　画面外へ落下して爆発するプレイヤーキャラクター

改造した素材のクラス図を図 32 に記す。無編集のクラスは無色で表示し、使用素材から改造したクラス・モジュールは赤色で表示し、新たに改造したクラスを青色で表示している。

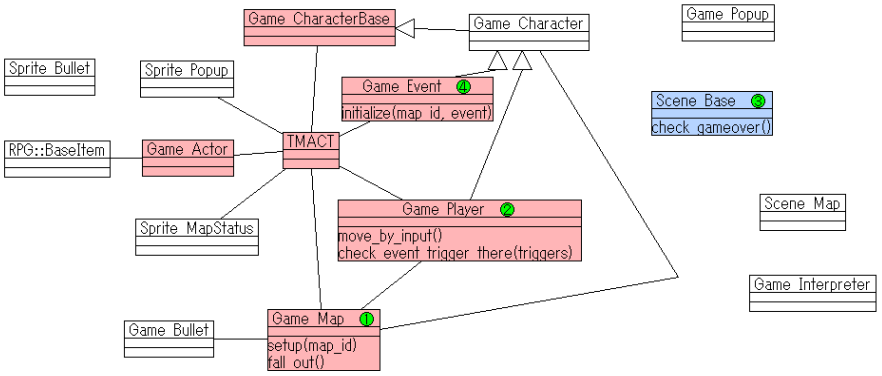


図 32　変更したスクリプトのクラス図

第 2 節　変更箇所の説明

本節では、改造した主なメソッドをクラス別で説明する。定数を一つ追加したのみの TMACT モジュールと if $game_switches[@action_switch]（又は[act]）を導入したのみのクラスの説明は省略する。

(1)Game_Map

Game_Map クラスでの改造メソッド（図 32 の 1 を参照）には、アクションスイッチの ON・OFF の設定と、マップの座標外に出た際の処理を追加した。

①アクションスイッチの ON・OFF

アクションスイッチの ON・OFF は、セットアップメソッド（図 33 参照）に導入した。赤色の枠内がアクションスイッチの ON と OFF を操作する。まず変数 act に、TMACT モジュールに新たに導入したアクションスイッチとして使用するスイッチ ID 定数 ACTION_SWITCH（初期状態では 1）を代入する。その後、8 行目で今いるマップのデータのメモ欄を参照し、その中に<act>と入力されていれば、アクションスイッチを ON にして、そうでなければ OFF にする。

```
1  #--------------------------------------------------------------------------
2  # ● セットアップ
3  #--------------------------------------------------------------------------
4  def setup(map_id)
5    @map_id = map_id
6    @map = load_data(sprintf("Data/Map%03d.rvdata2", @map_id))
7    act = TMACT::ACTION_SWITCH
8    if @map.note =~ /<act>/
9      $game_switches[act] = true
10   else
11     $game_switches[act] = false
12   end
13   setup_tile_hp
14   @tileset_id = @map.tileset_id
15   @display_x = 0
16   @display_y = 0
17   setup_events
18   clear_bullets
19   clear_popups
20   setup_scroll
21   setup_parallax
22   setup_battleback
23   @need_refresh = false
24 end
```

図 33　アクションスイッチの切り替え処理

②場外へ飛んだ際の処理

VXace の通常の操作ではマップの座標外へキャラクターを移動させる事はできないが、本論文で挙げた 2DACT スクリプトを使った操作では、重力の処理で下方向の画面外に出てしまったり、ダッシュ力を過剰な数値に設定してのダッシュで場外に出てしまったりすると、全く動きが取れないままになり、F12 キーを押してリセットしたり、VXace 自体を終了しなければならなくなる。そこで、このメソッド（図 34 参照）を導入した。5 行目から 6 行目へと続く if 文で、プレイヤーの横座標の現在位置 Game_Player クラスの@x が、マップの幅の値以上になって右側の場外に行くか、0 を下回って左側の場外に行くか、プレイヤーの縦座標の現在位置 Game_Player クラスの@y が、マップの高さの値以上になって下側の場外に行くか、0 を下回って上側の場外に行くかをすると、完全に動きが止まり、2DACT の被弾時に発生する全滅時のイベントを発生させる。全滅時のイベント中にマップ移動をするとその移動先でも同じイベントが発生するため、7 行目にイベント実行中はそのイベントは発生しない処理を追加した。

```
1  #--------------------------------------------------------------------------
2  # ○ 場外へ飛んだ際の処理
3  #--------------------------------------------------------------------------
4  def fall_out
5    if $game_player.x >= @map.width || $game_player.x < 0 ||
6      $game_player.y >= @map.height || $game_player.y < 0
7        return if $game_map.interpreter.running?
8        $game_player.vx =0
9        $game_player.vy =0
10       $game_temp.reserve_common_event(TMACT::ALL_DEAD_EVENT)
11   end
12 end
```

図 34 画面外へ出た時の処理

(2)Game_Player

Game_Player クラス（図 32 の 2 を参照）には、非アクション時の移動方法の追加と、正面のイベントの起動判定の四方向対応化と、その他のアクションの処理メソッドにアクションスイッチの条件分岐による挙動の変更の追加を行った。

①非アクション時の移動方法

方向ボタン入力による移動処理メソッド（図 35 参照）に、非アクション時の移動処理を加えたものが、赤色の枠の中になる。19 行目でアクションスイッチが OFF なら、4 方向の移動が可能になり、縦方向の移動も、横方向と同じ移動力を使って移動を行う。図 7 では 4 になっていた@move_count が 1 になっているのは、更に細かな移動を可能にするためである。そして、移動加速度の撤廃により、方向ボタンを押したその瞬間から最大の移動力で移動を行うようにした。

```
1  #--------------------------------------------------------------------------
2  # ● 方向ボタン入力による移動処理
3  #--------------------------------------------------------------------------
4  def move_by_input
5    return if $game_map.interpreter.running?
6    change_member if Input.trigger?(:L) # 操作キャラクター切り替え
7    if @ladder    # はしごにつかまっている
8      if Input.press?(:UP)
9        @vy = -@ladder_move_power
10       @move_count = 1
11       @stop_count = 0
12     elsif Input.press?(:DOWN)
13       @vy = @ladder_move_power
14       @move_count = 1
15       @stop_count = 0
16     end
17     get_off_ladder unless collide_ladder?
18   else          # つかまっていない
19     if $game_switches[@action_switch] == false    #非アクション時
20       if Input.press?(:UP)            # 上移動
21         return if @vx != 0
22         set_direction(8)
23         @vy = -@move_power
24         @move_count = 1
25         @stop_count = 0
26       elsif Input.press?(:DOWN)       # 下移動
27         return if @vx != 0
28         set_direction(2)
29         @vy = @move_power
30         @move_count = 1
31         @stop_count = 0
32       elsif Input.press?(:LEFT)         # 左移動
33         return if @vy != 0
34         set_direction(4)
35         @vx = -@move_power
36         @move_count = 1
37         @stop_count = 0
38       elsif Input.press?(:RIGHT)      # 右移動
39         return if @vy != 0
40         set_direction(6)
41         @vx = @move_power
42         @move_count = 1
43         @stop_count = 0
44       end
45     elsif @dash_count == 0 && $game_switches[@action_switch] == true
46       if Input.press?(:LEFT)          # 左移動
47         set_direction(4)
48         @vx = -@move_power
49         @move_count = 1
50       elsif Input.press?(:RIGHT)      # 右移動
51         set_direction(6)
52         @vx = @move_power
53         @move_count = 1
54       end
55     end
56     if Input.press?(:UP)            # つかまり（上入力）
57       get_on_ladder if collide_ladder?
58     elsif Input.press?(:DOWN)       # つかまり（下入力）
59       get_on_ladder(true) if collide_ladder_down?
60     end
61   end
62 end
```

図 35　非アクション時の移動を含めたボタン入力による移動処理

②イベントの 4 方向対応

2DACT スクリプトでは、正面イベント起動判定メソッドに上下方向の判定が実装されていないため、上下方向の判定を導入した。2DACT 中は上下を向かないので、このメソッドはアクション時と非アクション時を分ける必要は無い。

```ruby
#--------------------------------------------------------------------------
# ● 正面のイベント起動判定
#--------------------------------------------------------------------------
def check_event_trigger_there(triggers)
  if @direction == 4 || @direction == 6
    last_real_x = @real_x
    @real_x += @direction == 4 ? -@collide_w : @collide_w
  else
    last_real_y = @real_y
    @real_y += @direction == 8 ? -@collide_h : @collide_h
  end
  start_map_event(triggers, true)
  if !$game_map.any_event_starting? &&
      $game_map.counter?((@real_x + 16384) >> 15, @y)
      @real_x += @direction == 4 ? -32768 : 32768
  elsif !$game_map.any_event_starting? &&
      $game_map.counter?(@x, (@real_y + 16284) >> 15)
      @real_y += @direction == 8 ? -32768 : 32768
  end
    start_map_event(triggers, true)
  if @direction == 4 || @direction == 6
    @real_x = last_real_x
  else
    @real_y = last_real_y
  end
end
```

```ruby
#--------------------------------------------------------------------------
# ● 正面のイベント起動判定
#--------------------------------------------------------------------------
def check_event_trigger_there(triggers)
  last_real_x = @real_x
  @real_x += @direction == 4 ? -@collide_w : @collide_w
  start_map_event(triggers, true)
  if !$game_map.any_event_starting? &&
     $game_map.counter?((@real_x + 16384) >> 15, @y)
    @real_x += @direction == 4 ? -32768 : 32768
    start_map_event(triggers, true)
  end
  @real_x = last_real_x
end
```

図 36 正面のイベント起動判定（修正後← →修正前）

(3)Scene_Base

Scene_Base クラス（図 32 の 3 を参照）には、新たに編集を加えたクラスで、ゲーム中の全てのシーンのスーパークラスである。なので、このクラスの全滅後のゲームオーバー処理によってジャンプアクション Ver0.11a と JUMPVer0.1a の全滅時処理（図 11 参照）が機能しない。そこで、このクラスのゲームオーバー処理を変更した。それが、図 37 のメソッドである。もしパーティが全滅状態になれば、8 行目でコモンイベントが呼び出される。7 行目の処理は、そのコモンイベント中に同じイベントが発生するのを防ぐためである。なお、イベントコマンドによって全滅した時とバトルで全滅した時は別のメソッドでゲームオーバーとなる。

```ruby
#--------------------------------------------------------------------------
# ◎ ゲームオーバー判定
#     パーティが全滅状態ならゲームオーバー画面へ遷移する。
#--------------------------------------------------------------------------
def check_gameover
  if $game_party.all_dead?
    return if $game_map.interpreter.running?
    $game_temp.reserve_common_event(TMACT::ALL_DEAD_EVENT)
  end
end
```

図 37　被弾して全滅した時の処理

(4)Game_Event

Game_Event クラス（図 32 の 4 を参照）には、オブジェクト初期化の処理に、マップ移動をすると撃破した敵キャラクターが復活する処理を追加した。それが、図 38 の赤い枠に囲まれた 10 行目のセルフスイッチ操作の処理である。何故セルフスイッチ D を操作する事で敵キャラクターが復活するのかというと、敵キャラクターの HP が 0 になった時に、そのイベントのセルフスイッチ D が操作され、セルフスイッチ D が ON の時、それが条件で優先される別のイベントページのイベントとなり、それが空白のイベントの時に何もない状態になるからだ。

```ruby
#--------------------------------------------------------------------------
# ● オブジェクト初期化
#     event : RPG::Event
#--------------------------------------------------------------------------
def initialize(map_id, event)
  super()
  @map_id = map_id
  @event = event
  @id = @event.id
  $game_self_switches[[$game_map.map_id, @id, "D"]] = false
  $game_map.need_refresh = true
  @enemy_id = TMACT::ENEMY_DEFAULT
  @hp = 1
  @mp = 0
  @result = Game_ActionResult.new(self)
  moveto(@event.x, @event.y)
  refresh
end
```

図 38　敵キャラクター復活処理

第 3 節　スクリプトエディタ外の設定

本節では、スクリプトエディタ外での編集を説明する。ジャンプアクション Ver0.11a や JUMPVer0.1a と編集する項目はほぼ同じだが、ここでも変更・追加要素がある。

第 1 項　データベースでの設定

①メモ欄

アクターと武器のメモ欄にステータスを設定する。アクター側の設定項目は、ジャンプアクション Ver0.11a の状態から削除されているものがある。図 39 の左がジャンプアクション Ver0.11a のアクターメモで、右が改造したスクリプト素材のアクターメモである。ジャンプアクション Ver0.11a のダッシュ硬直の下には被弾無敵がある。そして、図 40 は武器のメモ欄で、これが書かれてある武器をプレイヤーキャラクターに装備して、初めてプレイヤーは弾を撃つことができる。これらの設定は全て RPG::BaseItem クラスを経由して、Game_Actor クラスでそれぞれに対応するメソッドに反映され、また別のクラスに参照される。

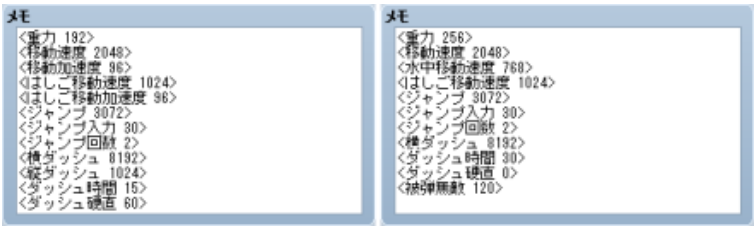


図 39　アクターメモ

図 40　武器メモ

図 41　RPG::BaseItem クラスの一部

図 42　Game_Actor クラスの一部

②コモンイベント

図 34 と図 37 から呼び出される全滅時のコモンイベントは図 43 のようになっている。BGM をフェードアウト（停止ではない）させ、プレイヤーキャラクターを透明にして爆発させ

る。これによりプレイヤーキャラクターが死亡したという事を強調させる。そして、画面を暗くして、その間に透明化を解除して、マップの移動を行い、体力全快させてから画面のフェードインを行う。この素材を使うツクラーは自由に編集できる。例えばアクションスイッチが OFF になっていればゲームオーバーになりタイトル画面へ行き、ON になっていればスクリプトを使ってアクションエリアに入る前のマップに戻ったり、リトライしたりする事もできる。

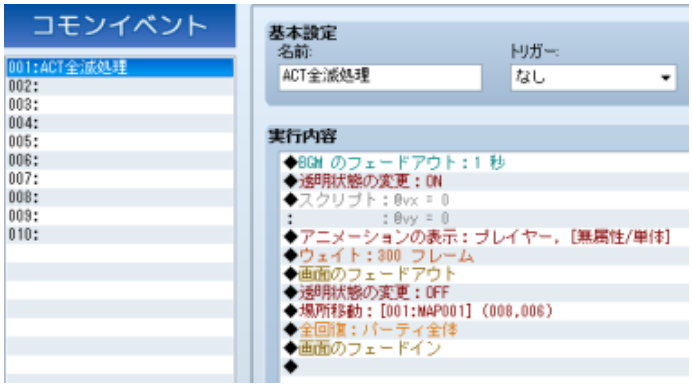


図 43　全滅時のコモンイベント

③アニメーション

アニメーションでは、被弾時のエフェクトが追加されている。これは TMACT モジュールでエフェクトの変更が可能だが、1 番邪魔になりにくいエフェクトは、111 番と 112 番に追加されたアクションダメージで、対象の色を 3 フレーム変色させるだけのエフェクトとなる。

④タイルセット

本論文で新たに公開する素材は、通常の RPG 移動とアクションの移動では様々なシステムが異なるので、通常の RPG のセットとアクションのセットと分けた。この素材のスクリプトを使う際は、セット自体のコピー&ペーストを推奨する。だが、Graphics フォルダ内の Tilesets にこの素材のタイルセットの画像があるので、先にそちらをコピー&ペーストしなければ使えない。

第 2 項　マップ上での設定

①マップの設定

マップの設定でするべきことは、アクションエリアなら ACT 用タイルセットを選び、メモ欄に<act>と入力し、通常の RPG エリアなら ACT 用ではないタイルセットを選ぶだけである。図 40 での ACT 用タイルセットは 5 から 7 までの名前に ACT を含むセットとなる。

図 44　タイトルセット一覧

②マップの描画

マップの描画では､アクションエリアなら最初にレイヤーA の通行可能な透明タイルで塗りつぶしてからオブジェクトを描画し、リージョンはオブジェクトを塗りつぶすように塗る。

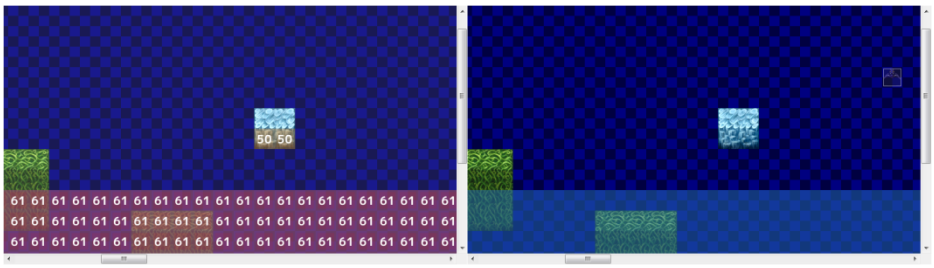


図 45　マップの描写（リージョン← →マップ）

③アクションエリアでのイベント設置

アクションエリアでの画像のあるイベントの全てに HP が割り振られている。たとえ敵キャラクターとして設定してなくても、イベント名に<貫通>と書かなければそのキャラクターは撃破されてしまう。ただ、セルフスイッチ D が発生条件のイベントページが無ければ敵キャラクターでも生存したままとなり、HP ダメージも与えられなくなる。図 42 は、サンプルステージのボス敵キャラクターのようなものである。

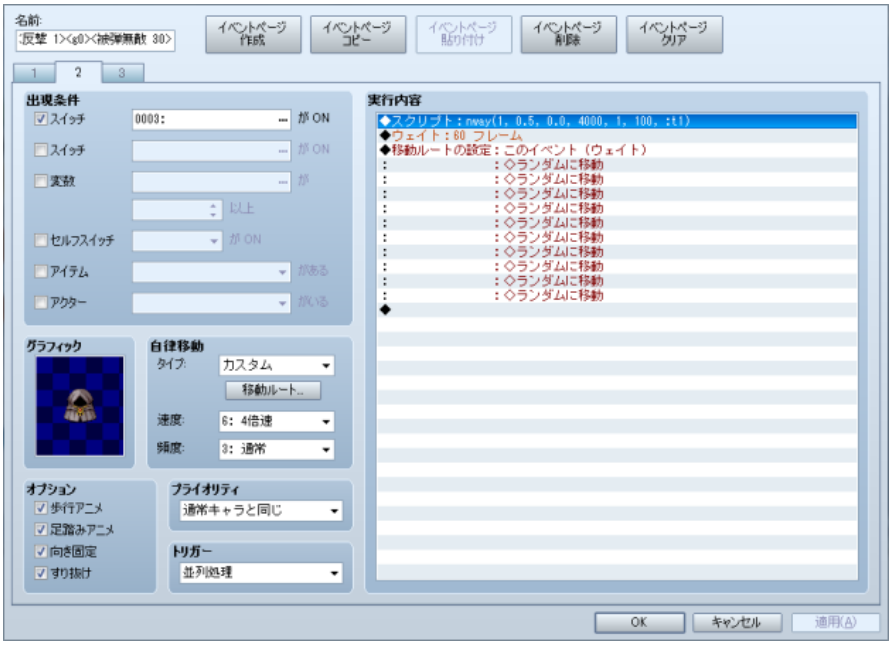


図 46　敵イベントの例

第5章　まとめ

今回私は、ネット上に上げられたRPGツクールVXAceのRGSS3アクションスクリプト素材を改造して新たなアクションスクリプト素材を作成した。その内容としては、元ある素材が『スーパーマリオ』シリーズのような操作感を持っていて、私が作成した素材が『ロックマン』シリーズのような操作感を持つ。そして、新たにRPG操作との切り替えができるようになり、RPGをメインにしながらアクションをミニゲームとして添えたり、逆にアクションをメインにしてRPGをミニゲームとして扱ったりをする事によって、RPGツクールVXAceのゲーム制作の幅が広がり、今後のRPGツクールVXAce製アクション作品の発展に繋がると考えられる。だが、飽くまでもRPGツクール作品なので、アクションゲームとしてのクオリティはアクションゲームツクール作品には劣る。しかし、容量はこちらの方が軽く、RPGができる点においてはこちらの方が優れている。

今後の課題として、今回改造したアクションスクリプト素材の不具合の修正等の改良を行う必要がある。左ボタンとジャンプボタンを押しながらショットボタンを押すと弾が出るが、右ボタンとジャンプボタンを押しながらショットボタンを押すと弾が出なかったり、RPG操作時に接触時や決定ボタンを押す事によって発生するイベントに上から接触したり話しかけたりする際の挙動の異常があったりしたので、まずはそれの修正を行う。特に右ボタンとジャンプボタンを押しながら弾を撃てない不具合はアクションスクリプト素材を改造する前からあったが、修正ができていない。また、不具合の修正の他に、ボスのHPバーの表示をしたり、一時的に無敵になったり重力の影響を受けなくなったり更に複雑な行動を起こす敵キャラクターの作成をする予定である。

RPGツクールVXAceは、過去にRPGツクールVX触れたツクラーにとっては大きな革命である。だが、目玉機能の一つであるRGSSの敷居が高く、値段も12,800円とRPGツクールシリーズの中では極めて高額な為、過去のシリーズを使い続けているツクラーも少なからず存在する。そこで、プログラムが苦手な人の為に、RGSSが備わっているRPGツクールを作品別に、初期状態にある全てのスクリプトを1行ずつ説明するヘルプを作成する事が望まれる。それを作成する事によって、目玉機能であるRGSSを少しでも多く理解し、RPGツクールXPやRPGツクールVXやRPGツクールVXAceを購入するツクラーが増加し、それらの作品が発展する事に繋がると考えられる。

参考文献

[1]高橋征義・後藤裕蔵:「たのしい Ruby 第 3 版」ソフトバンククリエイティブ(2010)
[2]ひきも記は閉鎖しました。 http://hikimoki.sakura.ne.jp/
[3]ツクリブルノート http://toghurt.net/

謝辞

2 年半の間、花川ゼミで共にプログラミング能力を伸ばしたゼミ生の皆さんに感謝します。特に、同じイラストコミック部（現・MANGA イラスト創作部）に所属し、共に部誌作りやパネルにイラストを描いたり、幹部として部をまとめたりしていた井上幹也氏に感謝します。そして、最後まで本論文の添削等の指導をしてくださった花川教授に感謝します。

付録

アクションスクリプトの利用規約

本論文で新たに作成したアクションスクリプト素材は､ひきも記のアクションスクリプト素材を使用して作成したため、利用規約はひきも記によるものとする。

このスクリプト素材は、連絡やクレジット表記や当サイトやひきも記へのリンクをせず、自由に使う事が出来る。完成作品の配布方法（無料・有料）や年齢制限も問わない。ただし、このスクリプトの著作権は tomoaky 氏にある。

改造の有無を問わず、素材の再配布をすることができるが、クレジット表記をする必要があるが、それは無料での配布に限るものとする。

スクリプト素材はサポートの対象外となっている為、自己責任で利用すること。