# ダメージ表示画像化

蒼竜 (@soryu_rpmakermv)*

## 1 はじめに

RPG ツクール MV では，戦闘中にダメージをバトラー上に描画する際に図 1 のようにダメージ数字を書き込まれたファイルを読み込み，出力するダメージに応じて画像を処理して描画を行っていた．


図 1: RPG ツクール MV のプロジェクトの system フォルダ下にある Damage.png

一方，RPG ツクール MZ ではこの方式が取りやめられ，ゲーム内で設定されたフォントに基づいて，ゲーム内でダメージ数字を書き込んだ文字列の画像を生成し描画するようになった．

これは，RPG ツクールのゲームエンジン (javascript) における画像読み込み速度の遅さから来る一連の最適化によるものだと推測される．特に RPG ツクール MV 製のゲームは動作が重いとも評判で，プレイ中にメモリ使用量が増大し続ける傾向にあった．RPG ツクール MZ では，使用を終えた画像を明示的に破棄する機構が備えられ，全般に渡ってメモリ消費を極力抑える設計となっている．

一方，画像の読み込みというのは非同期に，つまり一度画像の読み込みを指示した後はその完了を待たずに続く処理を行うようになっている．RPG ツクール MV/MZ におけるダメージ表示の処理は，**ダメージ表示を行う直前に表示する数字の準備を始める**．そして RPG ツクール MZ では，表示が終了したダメージ数字は直ちにメモリ節約のために削除されてしまう．ゆえに，ダメージを表示しようとするたびに Damage.png を毎回読み込まなければならなくなり，画像の準備が間に合わずにダメージが表示されないということが起きてしまう．

本プラグインは，ダメージ表示処理部分のみを RPG ツクール MV の処理を元に再現するものである．図 2 は RPG ツクール MZ のゲーム画面であるが，ダメージ画像は Damage.png を読み込んで処理したものとなっている．


図 2: 用意した Damage.png を用いたダメージ表示

一戦闘中は，ダメージ画像の破棄を毎回行わせないようにすることでダメージ画像の準備が表示に間に合わないという問題を解消している．また，図ではダメージ数字を処理する部分を書き換えてある．これはスクリプトの書き換えが必要であるが，本プラグインでは純粋にダメージの表示・演出処理の部分のみを抜き出した独立

*http://dragonflare.blue/dcave/

したファイルを用意することで実装難易度を緩和している．当該部分のスクリプト記述例やヒントも後の章で説明する．

これによって，RPG ツクール MV 以前に作成された Damage.png の素材を RPG ツクール MZ で再利用することができて，かつての資産を遺産とせずに済むし，ゲームフォントを再利用することによるダメージ数字演出の地味さを回避することができる．

## 2 使用法

本プラグインは以下の 2 つに分かれている．**ダメージ表示演出はデフォルトのままでよい場合は前者のみ**の導入でよい．

- SoR_AlternativeDamagePopup_MZ.js
- SoR_ADP_UpdateModifier.js

導入後は，ダメージ画像を RPG ツクール MV のように**./img/system/Damage.png** として配置する．

### 2.1 プラグインパラメータ

本プラグイン (本体ファイル `SoR_AlternativeDamagePopup_MZ.js`) のプラグインパラメータは，ダメージ表示演出の表現を変更するための (RPG ツクールのシステム上に元々実装されている部分の数値のみを変えられるようにした) ものであり，特にダメージ数字の初期条件 (表示位置の始点など) を調整するためのものである．ダメージ数字表示中の挙動は後に述べる方法で，スクリプトを記述することになる．

### 2.2 主な設定

- Damage_Duration
  - ダメージ数字を画面に描画する時間 (フレーム数)
  - 表示が始まり，消滅するまでの全体時間
- Initial_DamagePositionY
  - ダメージ数字表示位置 $y$ 座標 (対象バトラー位置基準)
- CriticalFlash_Duration
  - クリティカル発生時のフラッシュ発生時間 (フレーム数)
- CriticalFlash_Color
  - クリティカル発生時のフラッシュカラー，RGBA の 4 要素配列形式 ($[R, G, B, A]$)
- DamagePosX_Multiple
  - 連続ダメージ表示時のダメージ数字表示位置 $x$ 座標補正
  - ここでいう「連続ダメージ」とは，前のダメージ数字が消える前に次のダメージ数字が新しく表示される状況のことを言う
  - この値のぶんだけ，連続ダメージの表示位置が**横に**ずれていく
- DamagePosY_Multiple
  - 連続ダメージ表示時のダメージ数字表示位置 $y$ 座標補正
  - この値のぶんだけ，連続ダメージの表示位置が**縦に**ずれていく
- DamagePosX_Random
  - 連続ダメージ表示時のダメージ数字表示位置の**乱数による** $x$ 座標補正
  - `DamagePosX_Multiple` とは違って「だんだんずらす」ではなく，バトラー位置を基準にこの値を用いた乱数ぶんだけずれる
  - 前の数字とあまり重ならずにダメージ数字を散らばらせたい場合はこちらを使う (`DamagePosX_Multiple` との共用は可能)
- DamageFadeOut_Frame
  - ダメージ画像がフェードアウトし始める残り描画フレーム数
  - フレーム数は `Damage_Duration` で設定した値からカウントダウンを始め，この値になった時からフェードアウトが始まる
  - 小さくしすぎると急速に消滅するようになるので注意

### 2.3　表示基本位置の微調整

基本的には変えなくてよいと思われるが，デフォルトではアクター・エネミーでダメージ位置の基準が微妙ながら異なるため調整できるようにしている．

- Actor_damageOffsetX
- Actor_damageOffsetY
  - アクターに対するダメージ表示の基準座標からのずれ
- Enemy_damageOffsetX
- Enemy_damageOffsetY
  - エネミーに対するダメージ表示の基準座標からのずれ

## 3　ダメージ表示挙動の変更

ダメージ表示挙動を変更するには，スクリプトを記述する他ない．ダメージ数字が表示されている間の挙動は追加ファイル `SoR_ADP_UpdateModifier.js` の中身 (`calcDamageSpritePosition` 関数) を記述するのみで可能なようになっている．最低限の構文さえ記述すれば，RPG ツクール全体に渡るシステムを理解しなくてもダメージ表示演出を変えられるようになっている．

ただし，**自然な美しい描画を行うためには多少の数学・物理の知識が必要**となる．

### 3.1　RPG ツクール MZ デフォルトのダメージ表示挙動の観察

追加ファイル `SoR_ADP_UpdateModifier.js` の内容は，本質的に次のソースコード 1 のようにダメージ表示挙動を規定する `calcDamageSpritePosition` 関数のみとなっている．今，RPG ツクール MZ デフォルトの表示挙動が記述されている．

ソースコード 1: ’デフォルト設定’

```
1 Sprite_Damage.prototype.calcDamageSpritePosition
      = function(sprite,i,len) {
2   sprite.dy += 0.5;
3   sprite.ry += sprite.dy;
4   if (sprite.ry >= 0) {
5     sprite.ry = 0;
6     sprite.dy *= -0.6;
7   }
8 }
```

RPG ツクール MZ でのダメージ表示挙動は，対象バトラーの中くらいの位置にまずダメージを表示してから，基準点に向かって落下させた後，軽くバウンドさせてから停止するというようになっている．その動作はソースコード中にたったの 6 行で記述されている．まず，動作記述のために利用できそうな変数の一覧を表 3.1 に示す．

| 変数 | 意味 |
|---|---|
| sprite | ダメージ画像 |
| sprite.rx | 表示位置 $x$ 座標 |
| sprite.ry | 表示位置 $y$ 座標 |
| sprite.dx | $x$ 方向の変分 |
| sprite.dy | $y$ 方向の変分 |
| sprite.scale.x | ダメージ画像表示サイズ ($x$ 方向) |
| sprite.scale.y | ダメージ画像表示サイズ ($y$ 方向) |
| this._duration | 現在の表示時間 (フレーム数) |
| this.battler | このダメージの描画対象のバトラー |

表 1: ダメージ表示挙動時に使用できる主な変数

現在の対象バトラーを示す `this.battler` 以外は，RPG ツクール側で標準で利用できるものであるため，このまま利用する．特に重要なのは，現在座標 $(rx, ry)$ を決定するために差分 $(dx, dy)$ を用いていることである．この `calcDamageSpritePosition` 関数は，ダメージ数値がバトラー上に表示されている間 (つまり，プラグインパラメータ `Damage_Duration` で表示されているフレーム数)，毎回実行されて表示位置を動かしていくというものである．画像の最終的な位置 ($y$ 座標) を大真面目に書くと式 (1) のようになる．

$$\begin{aligned} y &= \int_0^{duration} f(t)dt \\ &= \sum_{t=0}^{duration} dy \end{aligned} \tag{1}$$

これは，$y$ 座標を直接書き換えているのではなく，1 フレームごとにダメージ数字が上下方向にどれだけ移動するかという値 $dy = f(t)$ を与えて，それを $y$ 座標の値に足し続けることを行うことで，滑らかにダメージ画像を移動させようとするものである．(小学校で習ういわゆる「速さ × 時間=道のり」というものである．)

間違っても $y+ = 1$ などと書くことがあれば，**ただ直線的にダメージ数字が移動する非常に出来の悪いゲーム**のように見えてしまうだろう．

改めてソースコード 1 を見てみると，$dy = 0.5t$ となっていて，それを $y$ 座標の値に加えている．つまり **1 フレームごとに移動速度がだんだん上がっている**ということになる．しかしこのままでは，ダメージ数字が無限に画面下方へ落ちて行ってしまうため，**4 行目の if 文**にて基準点よりも下に進んでしまったらダメージ画像を基準点で止めるようにしている．

一方，$dy$ に-0.6 がかけられている．これによって，一時的に反対方向に向かって画像を移動させるように働くが，元々 $dy = 0.5t$ であるため，すぐ下へ落ちるような振動する挙動となる．

### 3.2　放物線を描いて飛ばす

独自のダメージ表示方式を作るための例として，図 3 のように**ダメージ数字を放物線状に飛ばす**という表現方法を示す．


図 3: 放物線を描くように飛ぶダメージ表示

ソースコード 2 は，図 2 に示したものを表現するために作成したコードとなる．アクター・エネミーに応じて左右どちらかへ放物線を描いてダメージ数字を飛ばし，ある程度飛んだところからフェードアウトさせるようにしている．

ソースコード 2: ’放物線を描くように飛ばす’

```
1  Sprite_Damage.prototype.calcDamageSpritePosition
       = function(sprite,i,len) {
2   const t = 120−this._duration;
3
4   sprite.dx = this.battler.isActor()? 1 : −1;
5   sprite.dy = −3.0+0.063*t;
6   sprite.scale.x −= 0.001;
7   sprite.scale.y −= 0.001;
8   sprite.rx += sprite.dx;
9   sprite.ry += sprite.dy;
10  if(sprite.dy < 0) sprite.opacity−=1;
11 }
```

まず，放物線を描くように滑らかに飛ばすためには高校程度の初等力学の知識が必要である．図 3 のようにダメージ数字を打ち上げてから落下させる運動は「斜方投射」に相当する．

$$v_x = v_{x0} \tag{2}$$

$$v_y = v_{y0} - gt \tag{3}$$

$v_{x0}, v_{y0}$ は各方向に対する初速度となる定数である．また，$g = 9.81(m/s^2)$ は重力加速度を表す定数である．斜方投射は，$x$ 方向は最初に与えた速度 $v_{x0}$ のまま等速直線運動を行い，$y$ 方向は初速度 $v_{y0}$ で投げ上げる自由落下となる．

単純には，$v_{x0}, v_{y0}$ を好きに決めたものを `sprite.dx` と `sprite.dy` に代入するだけでよい．

4 行目では，$v_{x0} = 1$ として，アクターかエネミーかを判定して向きを変えている．5 行目では，$v_{y0} = -3.0$ としている．2D ゲームの画面では，実際の物理定数を使った計算値を描画することは現実的ではないため，適当に $g$ を調整して (ゲーム画面内に自然に収まるように)$g = 0.063$ を用いている．ゆえに，ソースコード上での $y$ 座標の変分 (上下方向の速度) は $dy = -3.0 + 0.063t$ となっている．ここでゲーム画面は座標系が上下反転しているため，$-3.0$ とは始めに上方向に飛ばすことを意味する．

時刻 $t$ は 2 行目で求めた現在のフレーム時間である (`this._duration` は残りフレーム数であることに注意)．またこの例では，総フレーム数を 120 と設定して処理している． $t$ は徐々に増えていくため，$dy$ はしばらくすると符号が反転して画面下方に向かって落ちていくようになる．

加えて，遠くにダメージ数字が飛んでいく様子を表すために，`sprite.scale.x` および `sprite.scale.y` を徐々に小さくすることでダメージ数字が小さくなるようにした．併せて，ダメージ数字が遠くへ飛んでいくタイミングを見計らって，10 行目では明示的にだんだんフェードアウトさせるようにしている．

このように，$dx$ は常に一定速度，$dy$ はフレーム時刻 $t$ に依存する速度が与えられて，毎回実際の座標 $(rx, ry)$ に加えられている．これによって，ダメージ画像が放物線を描くように滑らかに移動する表現ができる．

### 3.3 ダメージ表示挙動操作まとめ

ダメージ表示挙動を記述する上で重要なことは，1 フレームごとにダメージ数字の位置の増加量・変分 $(dx, dy)$ を指定することであった．その変分を $(rx, ry)$ に足していくことで滑らかな挙動が実現できる．おおまかな作成のための要点は以下のようになる．

1. `const t = XXX -this._duration` でフレーム時間取得
   - XXX は自分で指定した (プラグインパラメータ `Damage_Duration`) の値
2. `sprite.dx`, `sprite.dy` を指定 (時間に依存する場合は $t$ を使う)
   - $t$ を使わずに数値を指定すると，一定速度の移動となる．
   - $t$ を使うことで，時間に比例して速度が変化するようになる．
3. 作った `sprite.dx`, `sprite.dy` を反映するために `sprite.rx`, `sprite.ry` へ足す．
4. 途中で挙動を変えたい場合には $t$ を使って，`if` 文で分岐させる．

結局，$dx, dy$ をうまく操ることができれば，どんな表現をすることも可能になる．

## 4 実装（競合）情報

本プラグインは基本的にダメージ表示機能を改修しているため，全般が上書きとなる．
普通は考えられないが，どうしても他のダメージ表示系に手を加えるプラグインと共用する場合には別途個別の改修が必要になる．

### 4.1 上書き定義

ダメージ表示演出変更 (表示処理のみ)

- `Sprite_Damage` クラス全般
- Sprite_Battler.prototype.createDamageSprite

ダメージ画像の逐次破棄を抑止

- Sprite_Battler.prototype.destroyDamageSprite

### 4.2 処理継ぎ足し

ダメージ画像準備処理

- `BattleManager.setup`
- `Sprite_Damage.prototype.initialize`

## 5 バージョン情報

- ver 1.00 (Oct. 8, 2020) 公開