キャラ電　コンテンツ開発ガイド

第 2.0 版

平成 17 年　10 月 19 日

株式会社 NTT ドコモ

目次

はじめに 1
概要 1
必要となる知識 1
キャラ電コンテンツ開発者ガイダンス 2
1. キャラ電とは 2
2. キャラ電の種類 2
3. キャラ電のバージョンについて 3
4. キャラ電コンテンツ作成の流れ 4
4.1 2Dキャラ電の場合 4
4.2 3Dキャラ電の場合 6
4.3 コンテンツの配信 8
5. キャラ電コンテンツ作成における制限事項 9
6. キャラ電コンテンツ作成における概念（ビヘイビア、モータ） 13
6.1 キャラクタの動作 13
6.2 アクションの合成 17
6.3 パーツの状態 19
6.4 モータとは 21
6.5 ビヘイビアとは 23
6.6 ビヘイビアとモータの詳細 24
6.7 まとめ 41
7. キャラ電コンテンツ作成におけるガイドライン 42
7.1 アニメーションの定義 42
7.2 モーター（パーツ）の考え方 43
7.3 アクション 44
7.4 パーツ分け 44
7.5 全体アクションの定義 45
7.6 パーツアクションの定義 45
7.7 状態遷移 46
7.7.1 全体アクション 46
7.7.2 パーツアクション 46
7.8 2Dアニメーション 47
7.8.1 2Dアニメーションの制御 47
7.8.2 間隔（TIME） 47
7.9 アニメーションの競合 48
7.9.1 ボーンの衝突 48
7.9.2 テクスチャアニメーションの衝突 48
7.10 イベント文字列 49
7.10.1 ユーザー操作アクション 49
7.10.2 癖アクション 49
7.10.3 簡易リップシンク 49
7.10.4 トラッキング 49

7.10.5 デフォルトアクション 49
7.10.6 アクションリセット 50
7.11 簡易リップシンクの実現方法 51
7.12 テクスチャアニメーションによるリップシンク 52
7.13 癖アクションの実現方法 53
7.13.1 ボーンアニメーションによる癖アクション 53
7.13.2 テクスチャアニメーションによる癖アクション 53
7.14 イベントの競合解決 54
7.14.1 イベントの優先順位 54
7.14.2 イベントのマージ 54
7.15 まとめ 55
7.16 モーターの優先順位 56
7.16.1 ボーンアニメーション 56
7.16.2 テクスチャアニメーション 56
スクリプトリファレンス 57
8. 構文表記法 58
8.1 構文のルール 59
8.2 見出語 59
8.3 選択肢 59
8.4 繰り返し 59
9. 字句要素 60
文字集合 61
9.1 アルファベット 61
9.2 数字 61
9.3 図形文字 61
9.4 特殊文字 61
9.5 水平タブ 61
9.6 改行文字 61
9.7 コメント 62
9.8 トークン 62
9.9 トークンの回避 62
9.10 その他 62
10. 構文規則 63
10.1 ビヘイビア 64
10.2 modelセクション 66
10.3 motorセクション 67
10.4 textureセクション 68
10.5 mapセクション 69
10.6 documentセクション 70
10.7 startセクション 72
10.8 resetセクション 73
10.9 eventセクション 74
10.10 コマンド詳細（ start / reset / event ） 75
10.11 ビヘイビア使用例 77
10.12 モータ 79
10.13 モータ使用例 82

商標について

「iモード」「iアプリ/アイアプリ」「キャラ電」は株式会社NTTドコモの商標または登録商標です。

その他記載された会社名、製品名などは該当する各社の商標または登録商標です。

改版履歴

版数	変更箇所	項版	変更内容
1.0	-	−	新規作成
2.0	P3	3	「キャラ電のバージョンについて」を追加
	P10・11	5	インターレース GIF について追加
	P11	5	環境マッピングデータを追加
	P10	5.2	使用可能なポリゴン属性を追加
	P69	15.5	map セクションを追加
	P75	15.10	light（光源情報設定コマンド）を追加
	全体	−	新機能（ブレンディング・光源・環境マッピング・透視投影）について各機能詳細追記

はじめに

このドキュメントではキャラ電を開発する際のガイダンスを解説します。

概要

本書ではキャラ電を作成する為の手順や、作成する際に必要な概念を説明しています。

※キャラ電コンテンツを作成の際には、別途 NTT ドコモより提供するキャラ電コンテンツ作成ツール「キャラ電スタジオ」をご利用ください。また、「キャラ電スタジオ」に同梱されている「キャラ電スタジオ操作説明書」及び「キャラ電スタジオチュートリアル」もご覧ください。

必要となる知識

キャラ電を開発するにあたり以下の知識を必要とします。

- 3D およびグラフィックスに関する知識
  基本的な 3D やグラフィックスに関する知識を前提としていますが、
  不十分な場合でも一定の理解は可能です。

- ビヘイビア・モータの概念に関する知識
  キャラ電スタジオでデータを作成する際に必須となる知識です。
  本ドキュメントにて説明致します。

- Windows に関する基本的な操作
  本ドキュメントでは特に説明致しませんので、
  Windows に関する操作については別途書籍等で確認して下さい。

キャラ電コンテンツ開発者ガイダンス

ここではキャラ電コンテンツを作成する為に必要な知識や注意しなければならない点について説明します。

1. キャラ電とは

キャラ電とは、携帯端末上のTV電話におけるサービスの名称の事で、TV電話の際にユーザ自身の画像ではなくユーザの指定したキャラクタを通話相手に表示させ、その指定されたキャラクタをユーザの分身として動作させる事ができるサービスの名称です。ユーザはこのサービスを使用するにあたり、使用するデータを自分の携帯端末上にダウンロードして選択し、携帯端末上のキーを操作する事により、TV電話の際にデータに設定されているキャラクタを通話相手の携帯端末上に表示させる事が出来ます。

2. キャラ電の種類

■ キャラ電データ

携帯端末上で動作させるキャラ電のデータには構成されるデータの種類によって2種類あります。一つめは3Dモデルによって表現されたキャラクタをキーの入力に応じてアニメーション動作させる**3Dキャラ電**です。もう一つは3Dモデルを使用せず、写真や絵などの画像を利用してキーの入力と結びつけて表示させる**2Dキャラ電**です。キャラ電を作成する際に使用するツールであるキャラ電スタジオでは、この両方のデータを作成する事が可能です。

■ キャラ電ピクチャ

キャラ電データを使用し、ユーザが操作したアクションを動画又は、静止画でユーザの端末に保存した物の名称です。

3. キャラ電のバージョンについて

キャラ電ではバージョンによって利用できる表現が異なります。
表現する機能については、以下を参照して機種を判別した上で利用してください。

■ **キャラ電 Ver.1.0（キャラ電機能対応全機種）**
キャラ電 Ver.1.0 機能のみで作成したキャラ電データは、全てのキャラ電対応機種で利用可能です。

■ **キャラ電 Ver.1.1（902i 等）**
キャラ電 Ver.1.0 よりも表現力を向上させる為、以下のような機能を拡張しています。
拡張した表現は、Ver.1.1 に対応していないキャラ電端末では拡張機能を無視した形で表示されます。

◆1.ブレンディング
-透明や半透明の表現が可能になります。
・3D モデルの場合にのみ使用できます。
・ポリゴン属性のブレンディングを有効に設定することにより実現します。

◆2.光源
-環境光と平行光源を設定することによりモデルに陰影を付けることができます。
・3D モデルの場合に使用できます。
・ポリゴン属性の光源を有効に設定し、キャラ電データ作成の際に光源情報を設定することにより実現します。

◆3.環境マッピング
-3D モデルで、光源が有効設定されている場合に使用できます。
・ポリゴン属性の環境マッピングを有効に設定し、キャラ電データ作成の際に環境マッピング用の画像を設定することにより実現します。

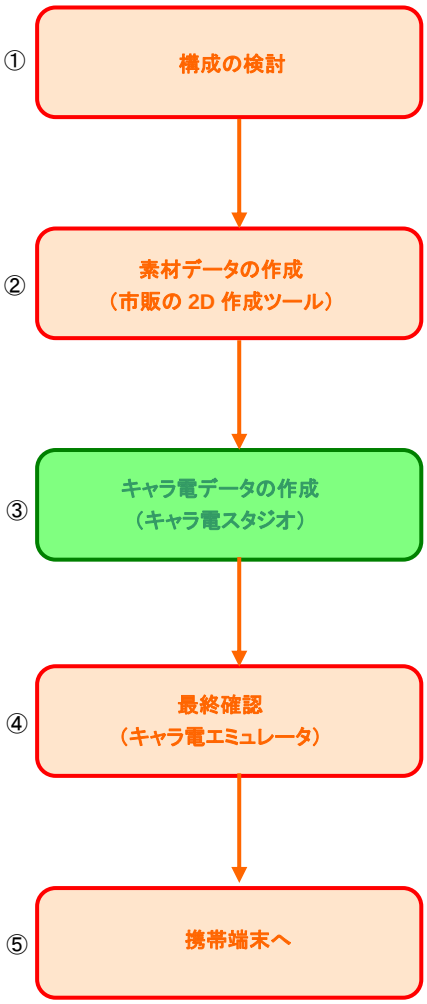
◆4.透視投影
-視点からモデルへと視線を伸ばし、その視線とスクリーンの交点にかたちを描画する方法です。近くの物が大きく遠くの物が小さく表示されるため遠近感が出ます。
・3D モデルの場合に使用できます。
・キャラ電データ作成の際に設定することにより実現します。

4. キャラ電コンテンツ作成の流れ

キャラ電コンテンツを作成するための大まかな流れを示します。コンテンツ作成の流れは作成するキャラ電の種類によって以下の2つに分かれます。

4.1 2Dキャラ電の場合

2Dキャラ電の作成手順は以下の様な流れになります。



どの様な画像を使用して表現するかを検討し、どのキーでどの様な表示をするかの構成を決めます。

構成に基づいて必要な素材を作成します。必要に応じて前景や背景に分けて作成します。

作成した素材や前景、背景等を組み合わせて2Dキャラ電を作成します。また表示タイミングや携帯端末上のキーへの割当て等を行います。

作成した2Dキャラ電のエミュレートを行い、最終的な動作の確認を行います。

ユーザのダウンロードによってキャラ電が携帯端末上で動作します。

① 構成の検討

2Dキャラ電を作成する為には3Dモデルを使用したキャラクタのアニメーションではなく、複数枚の画像を切り替える事によって動いているように見せるため、表現を行う為の画像が必要になります。使用する画像を前景や背景に分けて構成し、背景だけの変更や前景だけの変更で表現を変える事も可能です。これらを携帯端末上の各キーに割当て、そのキーが押された時の動作として表現されます。2Dキャラ電の構成を検討する場合には後述する「ガイダンス」に従って検討を行って下さい。

✓ 使用可能な画像の枚数には制限があります。

② 素材データの作成

検討した構成に基づいて画像データを作成します。作成するツールは市販の2D作成ツールで作成を行なって行きます。

✓ 画像は指定されたフォーマットである必要があります。

③ キャラ電データの作成

作成した素材を用いてキャラ電を作成するツールであるキャラ電スタジオによりデータを作成します。キャラ電スタジオにはプレビュー機能がありますのである程度のデータの確認は出来るようになっています。使用方法については別途「キャラ電スタジオ操作説明書」で解説していますので参照して下さい。

④ 最終確認

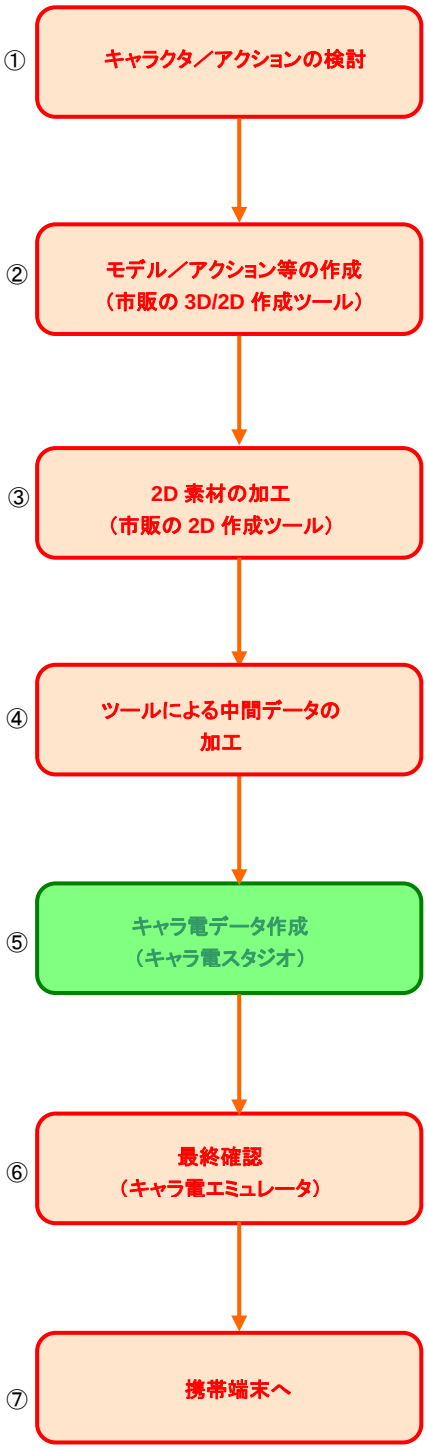
キャラ電エミュレータを用いて最終的なコンテンツの確認を行います。ここでは携帯端末とほぼ同じ環境で動作確認を行う事が可能です。エミュレータ上での確認が終了するとコンテンツの開発は終了となります。

⑤ 携帯端末へ

実際に携帯端末上にキャラ電をダウンロードして操作します。

4.2 3Dキャラ電の場合

3Dキャラ電の作成手順は以下の様な流れになります。



キャラクタのデザインから始まり、そのキャラクタの動作の構成を検討します。また、表現を行う背景なども含め、アクションの構成を検討します。

動作させる3Dモデル及びテクスチャを作成し、必要なアクションの作成を行います。この際、作成した3D データは ExporterPlugin によって中間データとして出力します。また、背景・前景を用意する場合は素材を市販の2Dツール等で用意します。

3Dツールで作成したテクスチャデータを市販の2Dツールで減色します。

テクスチャアニメーション等の編集を行う場合にはツールを使用して中間データを加工します。（この行程は必要ない場合もあります）

作成した素材を組み合わせて3Dキャラ電を作成します。また動作タイミングや携帯端末上のキーへの割当て等を行います。

作成した3Dキャラ電のエミュレートを行い、最終的な動作の確認を行います。

ユーザのダウンロードによってキャラ電が携帯端末上で動作します。

① キャラクタ／アクションの検討

キャラクタのイメージを作成し、そのキャラクタにどの様な動作をさせるのかを検討します。この際、キャラ電を作成する為に必要な「ビヘイビア／モータの概念」をある程度意識しておく必要があります。この段階ではキャラクタの動きのパターンやこの概念に基づいてキーに割当てるアクションを検討します。

- ✓ 表現できるアクションの数には制限があります。後述する「ガイダンス」に従った検討を行って下さい。

② モデル／アクション等の作成

検討した構成に基づいて3Dモデルやテクスチャ、アクションを作成します。3Dモデルを作成する際の注意点や制限などは後述の「ガイダンス」に基づいている必要があります。作成したデータは ExporterPlugin によって出力し、中間データとしておく必要があります。

- ✓ 各種 3D ツールにアドインする ExporterPlugin については該当のマニュアルを参照して下さい。
- ✓ 使用可能なアクション数やモデルの属性については後述の「ガイダンス」を参照して下さい。

③ 2D素材の加工

3Dツールで作成したテクスチャデータを減色します。
BMP であれば 8bit256 色 Max256x256、GIF であれば GIF89a 形式アニメーション GIF 以外 Max256x256 にします。

④ ツールによる中間データの加工

必要であれば ExporterPlugin によって出力された中間データの微調整を行います。
ExporterPlugin については別途マニュアルを参照して下さい。

⑤ キャラ電データの作成

作成した素材を用いてキャラ電スタジオによりデータを作成します。
キャラ電スタジオにはプレビュー機能がありますのである程度のデータの確認は出来るようになっています。使用方法については本書で解説していますので該当ページを参照して下さい。

⑥ 最終確認

キャラ電エミュレータを用いて最終的なコンテンツの確認を行います。ここでは移動機とほぼ同じ環境で動作確認を行う事が可能です。エミュレータ上での確認が終了するとコンテンツの開発は終了となります。

⑦ 携帯端末へ

実際に携帯端末上にキャラ電をダウンロードして操作します。

4.3 コンテンツの配信

<A>タグでのコンテンツ配信

キャラ電のコンテンツは、HTML の<A>タグを用いてリンクを設定します。

<記述例>

<A href="http://www.hogehoge.co.jp/charaden.afd">キャラ電君</A>

※ キャラ電データを配信する Web サーバでは、キャラ電ファイルの Content-Type が "application/x-avatar"に設定されている必要があります。
※ キャラ電データを CGI 経由でダウンロードする場合は必ず Content-Length を付与してください。
※ <A>タグでのデータ配信では、キャラ電ピクチャは、一律再配布不可となります。

アバターピクチャ再配布制限

以下に示すように<OBJECT>タグを利用してキャラ電ファイルを配信することにより、キャラ電ピクチャの再配布可/不可を指定できる事が可能です。

また、キャラ電のコンテンツ自体は、再配布一律不可となっております。

<キャラ電データ配信に必要な要素と属性一覧>

要素	属性	属性値	備考
OBJECT	data	コンテンツファイルの URI	-
	type	application/x-avatar	キャラ電データであることを示す識別子
	id※1	任意の識別子	OBJECT を特定するための識別子
	declare	-	オブジェクト宣言であることを示す識別子
PARAM	name	movie	固定
	value	free	アバターピクチャが再配布可能/再配布不可であることを表す。
		上記以外※2	
	valuetype	data	固定
A	href	参照する OBJECT ID	OBJECT の ID を指定

※1 id 属性の値は半角英字で始まり、半角英数字、"-"(ハイフン)、"_"(アンダースコア)、":"(コロン)、"."(ピリオド)で構成されている必要があります。
※2 対象となる name 属性値に指定可能な値以外の value 属性値(空の値(value="")を含む)が指定されることを示す。

<記述例>

```
<OBJECT declare id="chara.declaration" data="charaden.afd" type=" application/x-avatar ">
<PARAM name="movie" value="" valuetype="data">
</OBJECT>
<A href="#chara.declaration">キャラ電君</A>
```


5. キャラ電コンテンツ作成における制限事項

ここではキャラ電コンテンツを作成する上での制限事項を示します。

◆ 使用可能な3Dツール

3Dキャラ電を作成する際の3Dツールは以下のいずれかである必要があります。

1. 3DStudio Max Ver5.0/5.1
2. SoftImage 3.9/4.0
3. LightWave 7.0/7.5
4. AnimationMaster 9.5
5. MAYA 4.5

また、作成したデータをキャラ電スタジオで使用する為には、上記各ソフトウエアに対応した ExporterPlugin を用意する必要があります。
ExporterPlugin の使用方法等については各 Plugin のマニュアルを参照して下さい。

◆ 中間フォーマットデータ

各 3D ツールの ExporterPlugin によって出力される中間フォーマットデータには以下の種類があります。

1. **BAC6**（モデル）/ **TRA4**（アクション）フォーマット
   最新の ExporterPlugin によって出力された中間ファイルフォーマットです。現時点では AnimationMaster9.5 以外のツールに対応しています。作成した 3D データをキャラ電スタジオで使用する為には、この BAC6/TRA4 形式のデータフォーマットである必要があります。

2. **BAC5**（モデル）/ **TRA3**（アクション）フォーマット
   別途ツール(PAC.exe)を使用する事により BAC6（モデル）/ TRA4（アクション）フォーマットに変換する事ができます。作成した 3D データをキャラ電スタジオで使用する為には、BAC6/TRA4 形式に変換する必要があります。

◆ 作成する3Dモデル

3Dモデルを作成する際には以下の様な点に注意する必要があります。

1. ポリゴン数
   ポリゴンデータ作成時にポリゴン数を増やすと、それだけプログラムで展開した時に必要とされるメモリが多くなります。メモリに展開できればポリゴン数に関係無く表示可能ですが、携帯端末上である事を意識して、ポリゴン数は **750** ポリゴン程度を推奨します。

   以下に参考値としてボーン1個の球体モデルでの各ポリゴン数でのデータの目安を示します。

ポリゴン数	必要メモリ試算(Byte)	データサイズ約(Byte)
750	20,832	12,000
1,000	26,520	19,000
2,000	58,880	39,000
3,000	88,240	60,000
5,000	146,920	101,000
10,000	293,240	209,000

   また、キャラ電における描画方式はポリゴン単位のZソート方式のため、ポリゴンが重なるとちらつく事があります。モデリングを行う際にはなるべくポリゴンが重ならないようにして下さい。

2. 使用可能なポリゴン属性
キャラ電を作成する為に使用可能な3Dデータのポリゴン属性は以下の通りです。

属性		使用可／不可
片面		使用可
両面		使用可
透過		使用可
Muliple Texture		使用可
Flat Color Polygon		使用可
Texture Animation		使用可
ブレンディング	半透過	使用可
	加算	使用可
	減算	使用可
光源		使用可
環境マッピング		使用可

3D モデルの該当するポリゴン属性が無効に設定されていると、キャラ電データに設定しても機能は反映されません。キャラ電で使用する機能は3D モデルを作成する際に機能を有効にするようにポリゴン属性を設定しておく必要があります。

3. 画面への描画方法
画面への描画は**平行投影**または**透視投影**で描画されます。

4. テクスチャ
モデルに設定するテクスチャは3Dツールで作成した後、必要であれば減色する必要があります。テクスチャの種類は BMP または GIF で以下の属性である必要があります。

種類	属性
BMP	8bit、256 色、無圧縮形式 サイズの上限は 256 x 256
GIF	GIF89a 形式、2～256 色、透過色指定あり GIF アニメーションは不可 サイズの上限は 256 x 256

透過は GIF 形式で透過色の指定があるもの、または BMP の場合は0番パレットが該当のポリゴン属性に透過を指定した場合に行なわれます。
GIF の属性としてインターレース属性を設定することは可能ですが、インターレース効果は反映されません。
テクスチャの使用によるメモリの使用量は、テクスチャ1枚につき約 80K を消費します。

5. アクションについて
作成したアクションを ExporterPlugin で出力するとアクションファイルにはモデルの全てのボーンが含まれている状態で出力されます。キャラ電でアクションを再生する際に不都合が生じる場合がありますので後述するキャラ電スタジオにて必要なボーンのみを選択する必要があります。

アクションの使用によるメモリの使用量は、1アクションにつきモデルの 1/5～1/10 程度消費します。

また、アクションを作成する際のフレームレートは 30fps で作成して下さい。

モデルをパーツに分けて各パーツを同時に再生する様なアクションを作成する場合には各パーツのアクションの時間を同じにしてください。

◆ 背景・前景データ

キャラ電で背景・前景データを使用する際は以下の点に注意して下さい。
画像データの種類は BMP または GIF で以下の属性である必要があります。

種類	属性
BMP	8bit、256 色 サイズは Max 256x256、推奨 QCIF
GIF	GIF89a 形式、2～256 色 GIF アニメーションは不可 サイズは Max 256x256、推奨 QCIF

画像の使用によるメモリの使用量は、画像1枚につき約 80K を消費します。
なお、背景データは透過しません。前景データは BMP の場合は 0 番パレットが、GIF の場合は指定した透過色が透過されます。また、透過なし GIF が前景に指定された場合は 0 番パレットが透過されます。
GIF の属性としてインターレース属性を設定することは可能ですが、インターレース効果は反映されません。

◆ 環境マッピングデータ

キャラ電で環境マッピングデータを使用する際は以下の点に注意して下さい。
画像データの種類は BMP または GIF で以下の属性である必要があります。

種類	属性
BMP	8bit、256 色 サイズは Max 64x64
GIF	GIF89a 形式、2～256 色 GIF アニメーションは不可 サイズは Max 64x64

画像の使用によるメモリの使用量は、画像1枚につき約 80K を消費します。
環境マッピングデータは該当のポリゴン属性に環境マッピングを指定した場合に行なわれます。GIF の属性としてインターレース属性を設定することは可能ですが、インターレース効果は反映されません。

◆ メモリ概算

キャラ電を動作させる為に必要な概算メモリは以下の様になります。

素材	使用メモリ(約)
モデル	20K(750 ポリゴンの場合)
テクスチャ	80K(サイズに限らず) x テクスチャ数
環境マッピング	80K(サイズに限らず)
アクション	(モデルの 1/10)K x アクション数
アバタエンジン	100K(エンジン本体 30K + GIF/BMP 変換用 66K)
3Dエンジン	200K

上記の合計値が実際に必要になるメモリとなります。

◆ キャラ電スタジオによる制限

キャラ電データを作成する為に使用するツール、キャラ電スタジオでは以下の制限を設けていますのでキャラ電を設計する際に注意して下さい。

項目	制限内容
キャラ電データサイズ	100 Kバイト以内
テクスチャ総数（背景・前景を含む）	16 ファイル以内
アクションデータ総数	128 ファイル以内
環境マッピングデータ総数	1 ファイル以内
キャラ電タイトル名	60 バイト以内
キーイベント数	100 キー以内
キーアサインヘルプ数 （全体・パーツ合計）	60 個以内
キーアサインヘルプの アクション名文字数	60 バイト以内
隠しアクションのキー数（#を除く）	5 キー以内（#12345#）
モータ名の長さ	64 バイト以内
モータファイル名の長さ	16 バイト以内
ステート名の長さ	16 バイト以内
モータ数	16 モータ以内
ステート数	128 ステート以内
ステートのアクションファイル数	1 ファイル以内
ステートのイベント定義数	16 イベント以内
ステートのイベント定義毎のコマンド数	16 コマンド以内

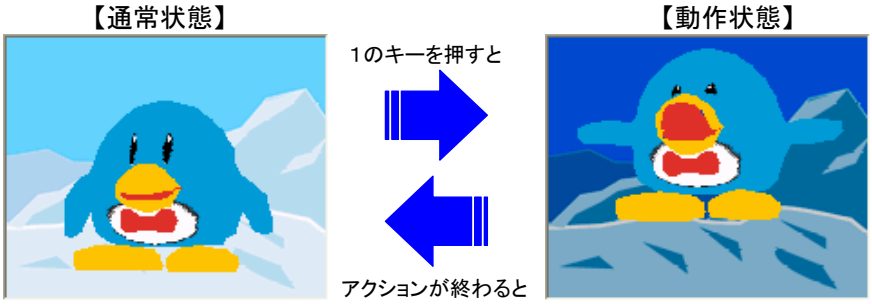
6. キャラ電コンテンツ作成における概念（ビヘイビア、モータ）

キャラ電を作成する際には、これから説明するビヘイビアとモータの概念を理解しておく必要があります。ここではビヘイビアとモータについて具体例を示しながら説明します。

6.1 キャラクタの動作

まずはじめにキャラクタをキャラ電として動作させる場合の動作のさせ方と、その考え方について解説します。キャラ電としてキャラクタを動作させる場合にはキャラクタの各アクションを、キャラクタ全体で動作を行う「全体アクション」、キャラクタの一部を動作させる「パーツアクション」、同じ動作をし続ける「ループアクション」、各アクションへ遷移する為の「補間アクション」として考えます。それぞれについて例を挙げながら考え方を以下に説明します。

1．キャラクタ全体で動作を行うアクションを考える（全体アクション）

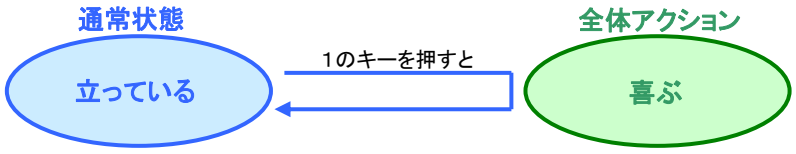


体全体を使って喜んでいる動作

最初の状態ではキャラクタは静止していますが、指定されたキーを押すと喜んでいる様な動作をします。キャラクタの動作としては一連の動作を行なってから元の状態に戻る様な構成として考えます。この様に、指定されたキーを押してからキャラクタ全体で一連の動作を行なって最後に元の状態に戻るような形の動作を**全体アクション**と呼びます。

考え方としてはアクションは状態の遷移として考えられますので、全体アクションの遷移は通常状態を起点にして、以下の様な遷移図で示すことが出来ます。遷移図とは、想定しているキャラクタの状態の関係を図にしたもので、作成しておくとキャラクタの状態を整理するのに非常に役に立ちます。書式が決まっているわけではありませんが、例えば以下の様に表現します。

【遷移図】



通常状態を起点に、全体アクションの動作を行ない、初期状態に戻る

2. キャラクタの一部を動作させるアクションを考える（パーツアクション）

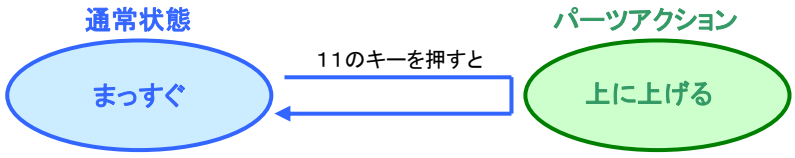
左腕を上に上げる動作

次に、キャラクタの一部だけを動作させる場合を考えます。通常の状態ではキャラクタは静止していますが、指定されたキーを押すとキャラクタの一部（上記の例では左腕）を動作させて元に戻します。左腕の一連の動作が終了すると通常状態に戻ります。この様に指定されたキーが押されてからキャラクタの一部を動作させ、その一連の動作が終了したら元の状態に戻るような動作を**パーツアクション**と呼びます。

パーツアクションも動作の遷移を通常状態を起点にして、以下の様な遷移図で示すことが出来ますが、状態遷移としては左腕だけの遷移と考えられます。従って、パーツアクションを考えるときにはキャラクタのモデルをパーツ毎に分解して考え、パーツだけの遷移として考えます。

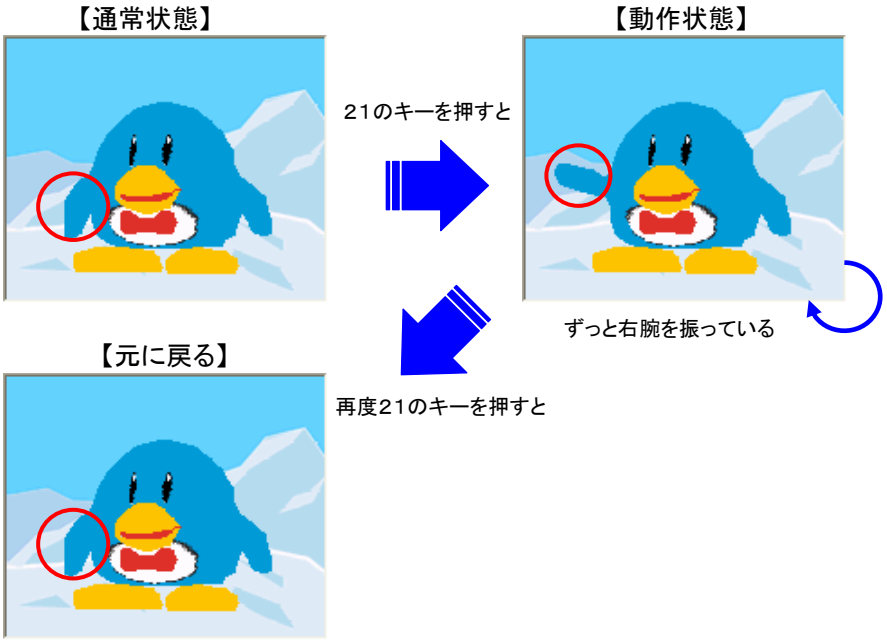
このアクションを遷移図で表すと以下の様に左腕だけの遷移として表せます。

【遷移図（左腕）】



通常状態を起点にパーツアクションの動作を行ない、通常状態に戻る。
ここではパーツ（左腕）だけの遷移である。

3. キャラクタの一部をループ動作させるアクションを考える（ループ動作）

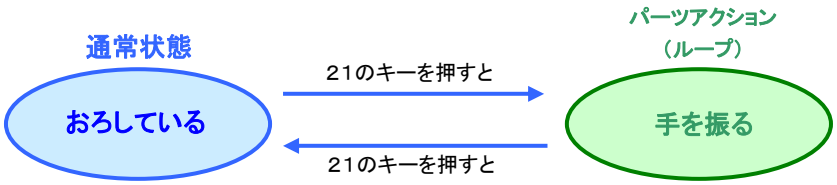


今度はキャラクタの一部だけをループ動作させる場合です。ループ動作とは、通常の状態から動作状態へ移行した後、そこで同じアクションを繰り返し続けるものをいいます。上図では通常の状態ではキャラクタは静止していますが、指定されたキーを押すとキャラクタの一部（上記の例では右腕）をずっと振りつづけています。この場合、右腕の一連の動作が終了しても通常状態には戻らず、再度同じアクションを繰り返します。これはアクションの最初のフレームと最後のフレームの動作を同じにしておき、その一連の動作を繰り返して動作させているためです。この様な繰り返しを前提としたアクションの動作を**ループアクション**と呼びます。

ループアクションは動作の遷移を通常状態を起点にして、以下の様な遷移図で示すことが出来ますが、パーツアクション同様、状態遷移としては右腕だけの遷移と考えられます。従って、ループアクションを考えるときにはキャラクタのモデルをパーツ毎に分解して考え、ある部分のパーツだけの遷移として考えます。（キャラクタをパーツに分解して考えない場合は、キャラクタ全体をひとつのパーツと考える事もできます）

上記の例を遷移図で表すと以下の様に右腕だけの遷移として考えられます。
ここでのポイントは矢印が一方向であるという事です。

【遷移図（右腕）】



各状態を起点にして状態が移行する。
パーツ（右腕）の状態はキーを押す毎に変化する。

4. 次の動作への「つなぎ」のアクションを考える（補間アクション）

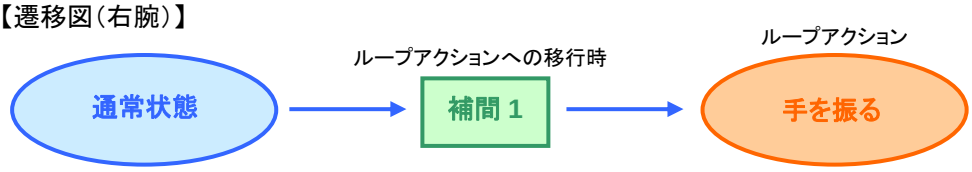
3. で説明したループアクションでの動作をもう少し詳しく見てみることにします。先ほどのアクションでは通常状態から右腕を振り続けていましたが、よく見るといきなり手を振り出すのではなく、下がっていた手が上がってから手を振り始めています。手の部分をアップにして見てみます。

これはパーツアクションで手を振り始めるまでのアクションの様子を表しています。通常状態で下がっていた手が動作状態に移行する前に間の動作として手をあげる途中の動作（補間）を行っています。

この様にループアクションにおいて、目的とする動作の状態になるまでの間の動作を**補間アクション**と呼びます。キャラ電の動作はこの補間アクションを考える事でよりスムーズなアクションの動作を実現する事ができます。

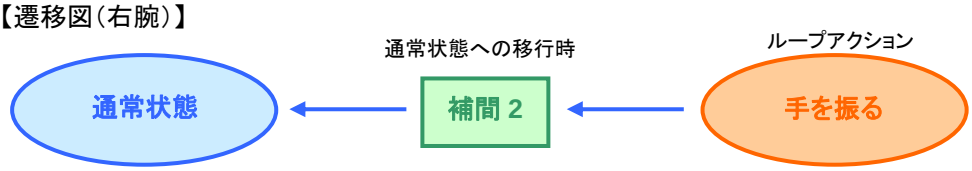
先ほどの右手を振り続けるループアクションをもう一度遷移図で表すと以下の様になります。ここでのポイントは遷移先の状態までの間に「**補間アクション**」を経由している事です。

【遷移図（右腕）】



通常状態を起点に補間アクションを経て「手を振る」状態に移行する

【遷移図（右腕）】



「手を振る」状態を起点に補間アクションを経て通常状態に移行する

ここまでの説明で、キャラクタの動作を考える手順としては、

1. 全体アクションの動作を決める
2. パーツアクションの動作を決める
3. ループアクションの動作を決める
4. ループアクションでは状態を遷移する際に補間アクションを考える

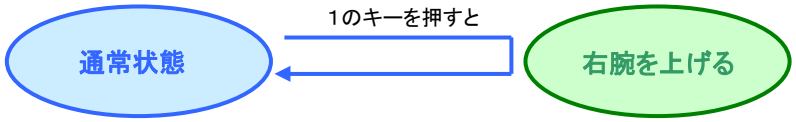
である事がわかりました。

6.2 アクションの合成

ここではパーツの動作を同時に行なわせる事を考えてみます。右腕と左腕を同時に動かす例をみてみましょう。今度はパーツの遷移図の方から考えてみます。

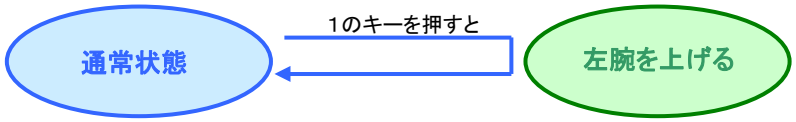
遷移を考えるときはパーツ毎に右腕の遷移と左腕の遷移を分けて考えます。

【右腕の遷移図】



通常状態を起点にパーツアクションの動作を行ない、通常状態に戻る

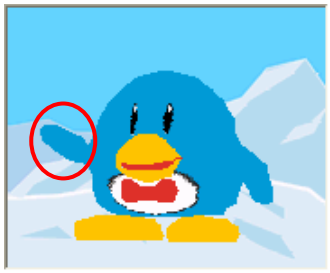
【左腕の遷移図】



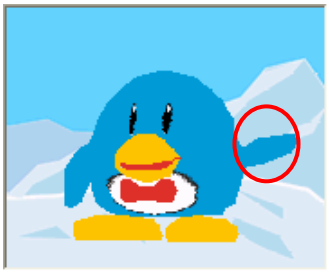
通常状態を起点にパーツアクションの動作を行ない、通常状態に戻る

以下の様に各パーツ毎のアクションを作成します。

【右腕を上げる】



【左腕を上げる】



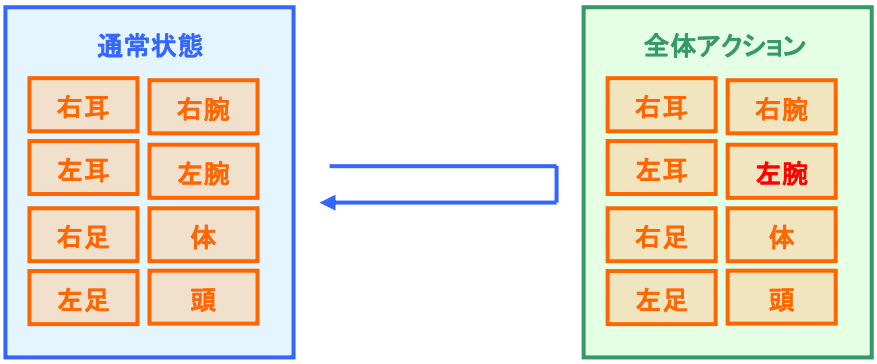
このそれぞれのパーツの動作を同じキーに割り当てる事でこれらを同時に再生する事が出来ます。

【通常状態】

1のキー

【両腕を上げる】

パーツ毎のアクションが同時に動作するので、両腕を上げているアクションになります。この事から、キャラクタ全体を動作させる場合にもパーツ毎の状態の遷移であるとして考え、全体アクションは各パーツ毎のアクションの集まりであるという事がわかります。



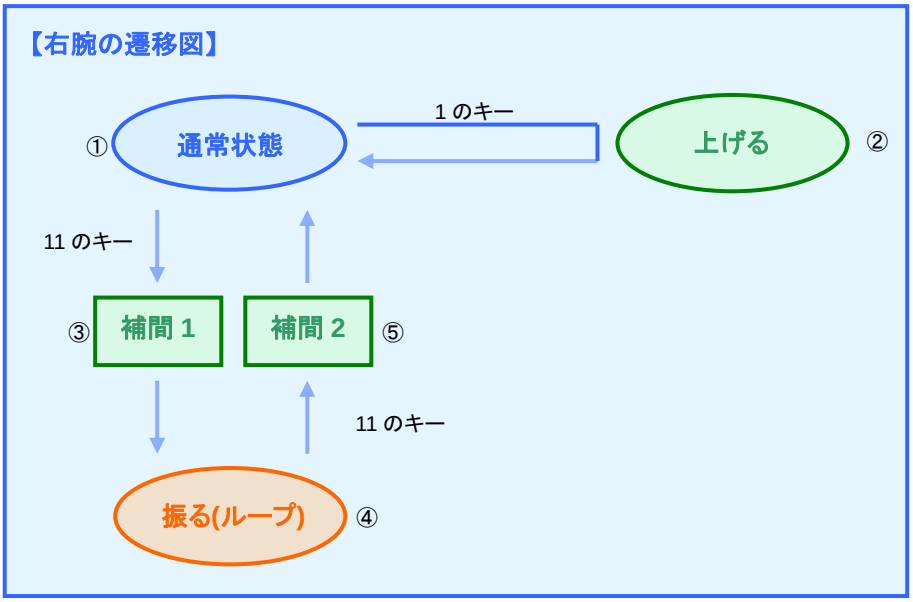
全体アクションも各パーツ毎のアクション毎の遷移であるとして考えられます。

＜補足＞
　この例では、あえて右腕と左腕を別々のパーツとして考えましたが、実際に両腕をあげるだけの動作であれば両腕をひとつのパーツとして考える事も可能です。ここでは考え方を説明をする為の便宜上のものと捉えて下さい。

6.3 パーツの状態

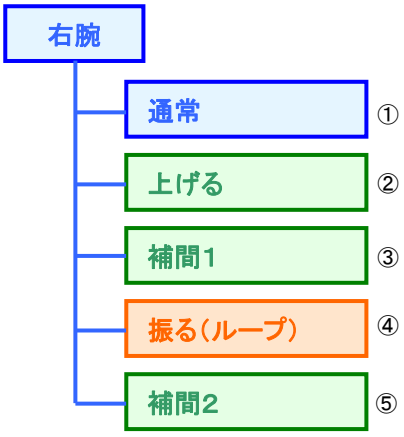
これまで説明してきました様にアクションを考える際にはキャラクタをパーツ毎に分解して考え、各パーツ毎に動作させてこれらを合成する事で一つのアクションを実現出来る事がわかりました。ここではこれらのパーツの状態について考えてみます。

例としてこれまでの説明に出てきた右腕だけの遷移について考えてみます。
右腕の動作としては「右腕を振る（ループ）」と「右腕を上げる」の2つの動作がありました。
これらを右腕の遷移図として全体を表現してみます。



上の遷移図は、右腕は「通常状態」から1のキーを押されると、「上げる」状態に遷移し、動作が完了すると「通常状態」に戻ります。また、「通常状態」から11のキーを押されると補間アクションを行って「振る（ループ）」状態に遷移します。「振る（ループ）」状態で11のキーを押されると補間アクションを行って「通常状態」に戻ります。と、表現されている事になります。

この遷移図からわかる様に、パーツには「**状態**」が存在することになります。
補間アクションも状態の一つと考えると、①～⑤の5つの状態が存在することになります。



この様にパーツには「状態」が存在していることがわかります。

同様にキャラクタの他の部分もパーツとして考えると左腕、頭、足などの各パーツにも状態が存在することになります。

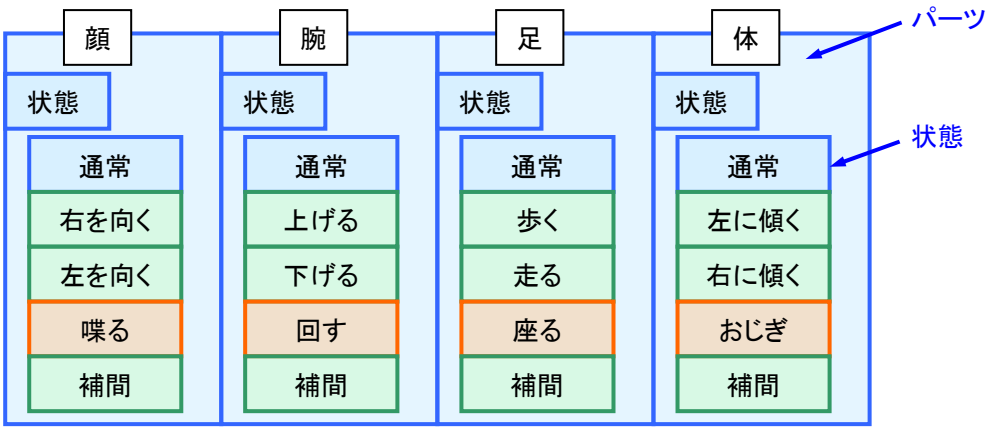
このことから前項の結果に引き続き、キャラクタの動作を考える場合には、

1. 全体アクションの動作を決める
2. パーツアクションの動作を決める
3. ループアクションの動作を決める
4. ループアクションでは状態を遷移する際に補間アクションを考える
5. 各アクションで動作させるパーツを決定する。
6. 各パーツ毎に遷移図を作成する
7. 遷移図を元に各パーツ毎に状態を洗い出す

という手順でキャラクタの動作を考えて行くという事がわかります。

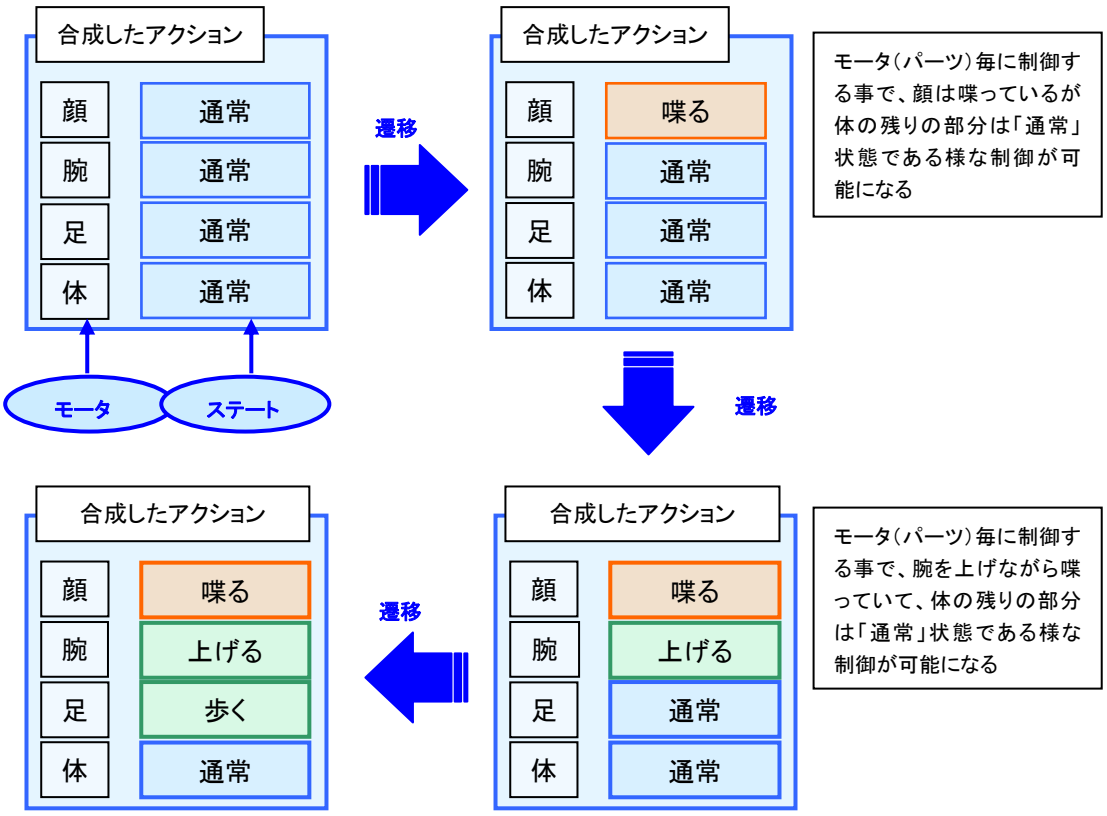
6.4 モータとは

今までの説明でキャラクタの動作の考え方について説明してきましたが、ここではモータについて説明します。モータとは、ひと言でいうとこれまでに説明してきた「パーツ」の事をいいます。キャラクタの動作を「各パーツ毎のアクションの合成」としてとらえ、各パーツ毎の「状態」の遷移によりひとつの動作を実現することになります。



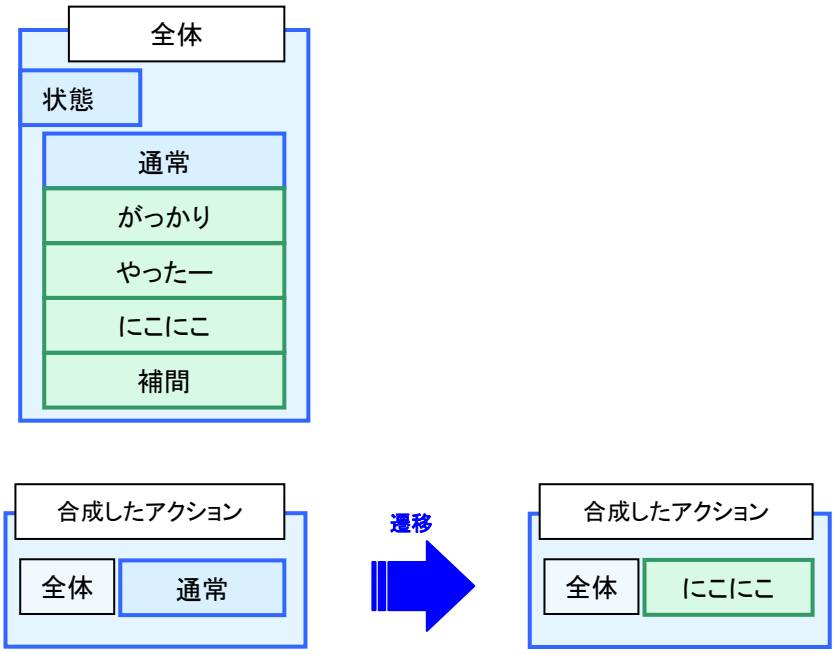
「パーツ」毎に「状態」がある

上記の様に各パーツ毎に状態を用意した場合、この各パーツの事を**モータ**と呼びます。
また、各モータに用意した状態の事を**ステート**と呼びます。
実際にアクションを行う際は各**モータ**（パーツ）毎の**ステート**（状態）を合成する事でアクションを実現します。



モータ（パーツ）毎のステート（状態）を合成する事で、様々な動作が実現できます

モータは必ずしも複数用意する必要はなく、一つのモータで動作を実現しても構いません。これはキャラクタをパーツに分けずに、一度に全てのパーツの状態を遷移させて、一つ一つのアクションとして作成する事になります。

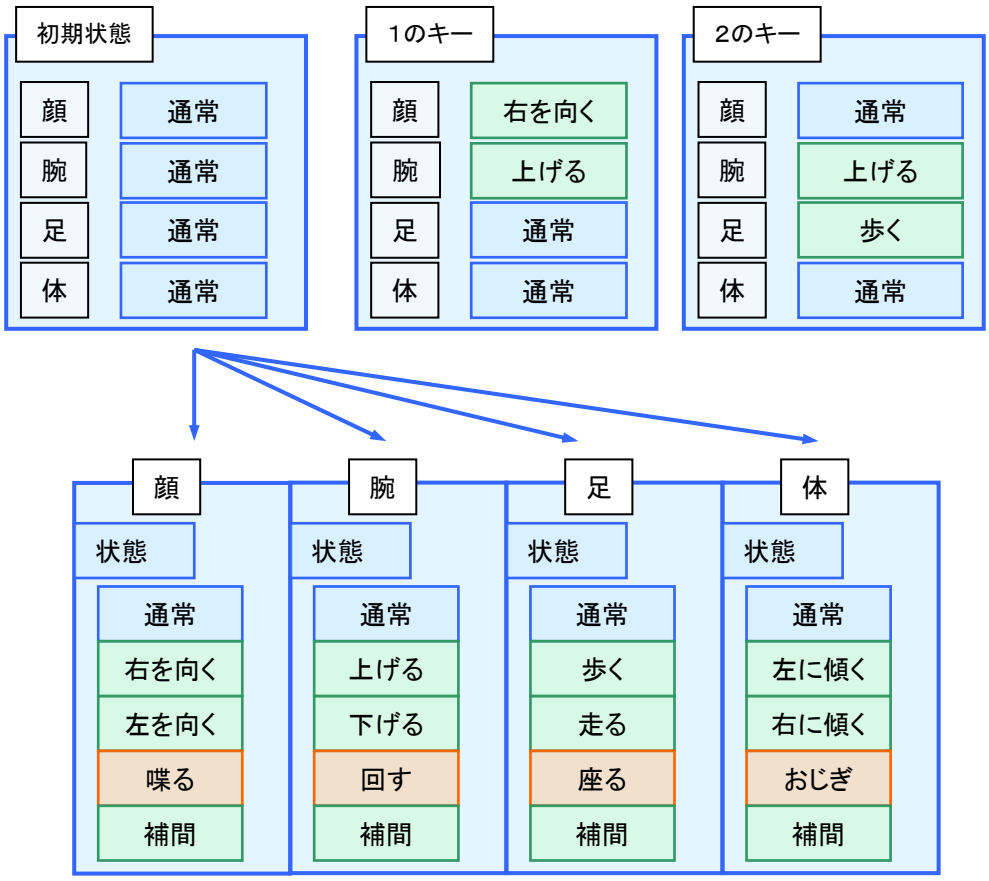


モータを一つとして考える場合はキャラクタの表現を一つのステート（状態）で用意する事になります。版権のある動き方の決まったキャラクタを動作させる場合にはモータを一つにした方がそのキャラクタ本来の動きに忠実になる場合もあります。

このドキュメントの後半で説明するキャラ電スタジオでは、モータ（パーツ）に対するステート（状態）を定義して行くことでキャラ電の動作を作り上げて行くことになります。

6.5 ビヘイビアとは

続けてビヘイビアについて説明します。ビヘイビアとはイベントに応じたキャラクタの振る舞いの事を指します。イベントというのは携帯端末からのキー入力や、ある一定の時間間隔など、キャラクタの状態を遷移させるためのトリガーになるものと言えます。キャラ電では、キャラクタの最初の状態（初期状態）、リセットされた状態（リセット状態）、各キーを受け付けた時、をイベントとして捉えてこれをビヘイビアから設定します。具体的には、今までの説明に出てきた各モータのステートをそれぞれのイベントに応じて設定する事になります。



ビヘイビアでは「初期状態」等のキャラ電の状態に応じて各モータ（パーツ）のステート（状態）を定義することにより、キャラ電全体の制御が可能になります。

キャラ電の状態には以下の様な状態があります。

1. **初期状態**
   キャラ電が携帯端末上にロードされた直後の状態で、この時の各パーツの状態を定義する事になります。
2. **リセット**
   携帯端末上でリセットキー（0）を押された時の状態で、この時の各パーツの状態を定義する事になります。
3. **イベント**
   携帯端末上の各キー（1～9や11等）を押された時の状態で、この時の各パーツの状態を定義する事になります。実際にはこの「イベント」の各キーに対応した動作を設定する事でキャラ電の様々な表現をさせる事になります。

6.6 ビヘイビアとモータの詳細

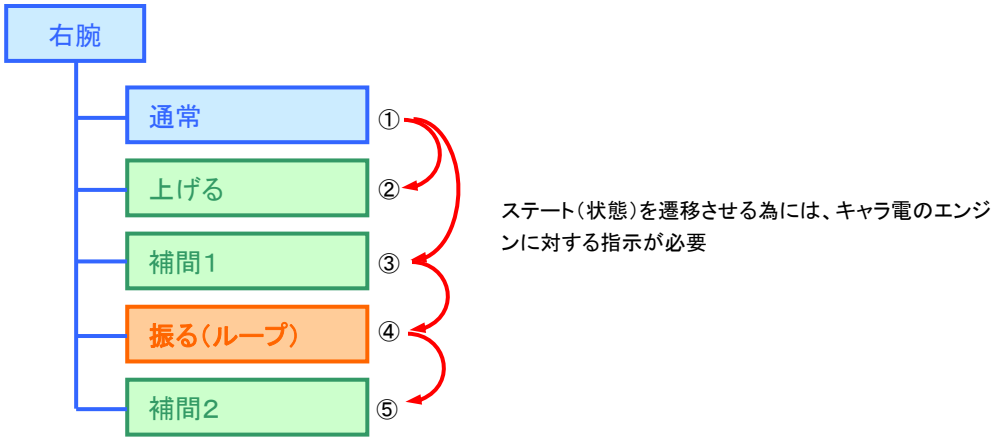
ここまでで概念的なところは一通り終わりになります。ここでは前回までに登場したビヘイビアとモータについてもう少し詳しく見てみることにします。これから説明する詳細の内容としては、ビヘイビアがキャラ電全体の制御を行っており、携帯端末からのキーイベントに応じてイベントの発行を行なっている事や、モータのステートの遷移の方法、また、ステートと共にその中で指定できるイベントやコマンドについての説明になります。

前項までの内容でキャラクタの動作を考える時には、キャラクタをパーツに分けて考え、それぞれのパーツの動作の組み合わせによって各々の動作させる様に考えるか、それともキャラクタ全体をひとつのパーツとして考え、動作をさせる様に考えるかの2通りの考え方があることが分かりました。いずれにしてもモータにステート（状態）を用意し、その状態を遷移させることでキャラクタの動作が行われる事になります。ここではまずモータ（パーツ）におけるステート（状態）の遷移について説明します。

1. イベントに応じたステート（状態）の遷移

キャラ電の内部ではキャラ電の動作を制御しているキャラ電のエンジンがあります。このキャラ電のエンジン内部では、キャラ電を動作させる為にモータ（パーツ）毎の現在のステート（状態）を保持しています。従って、実際にステートを遷移させるためには、キャラ電のエンジンに対してステートの遷移を指示する事によりステートの遷移を行わせる事になります。

＜モータ（右腕）での例＞

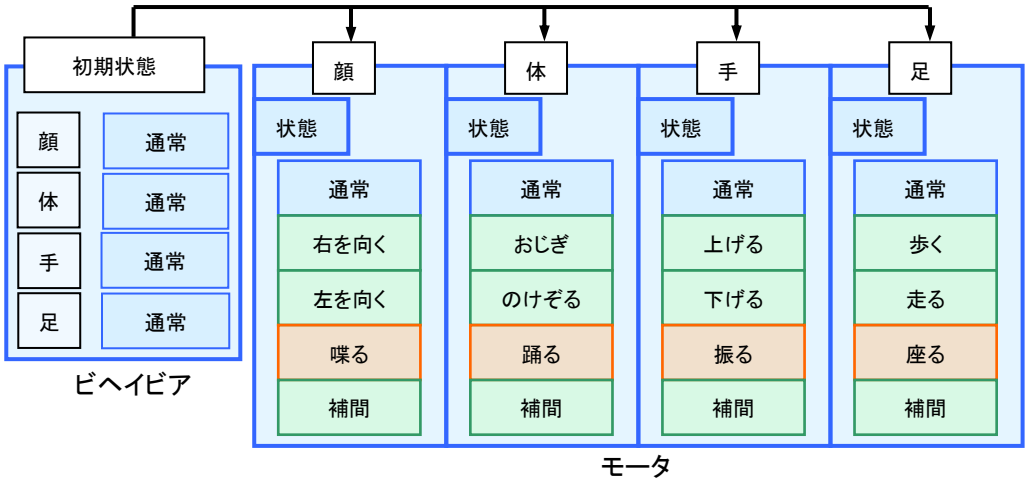


ステートの状態を遷移させるにあたり、「いつ」「どのように」ステートを遷移させるかを指定する必要があります。キャラ電ではこの遷移のタイミングをビヘイビアとモータの各々のタイミングで制御することになります。ここではビヘイビアから指定するステートの遷移のタイミングを説明します。ビヘイビアから指定するステートの遷移タイミングとしては以下のものがあります。

＜ビヘイビアから指定するステートの遷移タイミング＞
(1)初期状態
(2)リセット時
(3)キー押下時
(4)癖イベント発生時
(5)リップシンクイベント発生時

1. 初期状態

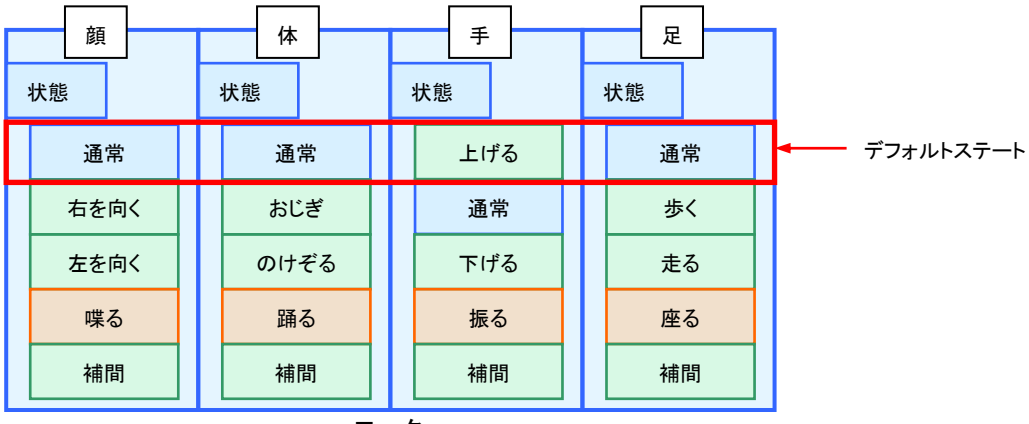
初期状態とは、キャラ電が携帯端末上に読み込まれた直後の状態を言います。ここでの動作の指定は通常、各モータ毎に用意した通常状態のステート（状態）になる様に状態を設定します。



上図の例では「初期状態」で「顔」「体」「手」「足」の各モータのそれぞれの「通常」ステートを設定しているため、キャラ電が携帯端末上に読込まれた直後にはキャラクタの各パーツは「通常」のアクションを行います。

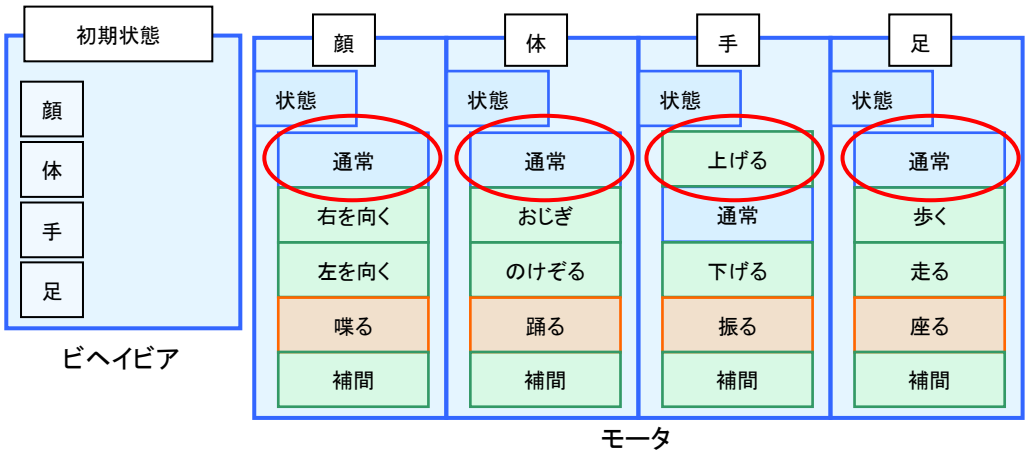
＜**デフォルトステート**について＞

ここで、注意しておかなければならない点があります。キャラ電で作成するモータ内のステートのうち、先頭に定義するステートには意味があります。各モータ内の先頭のステートは自動的に**デフォルトステート**として設定される様になっています。**デフォルトステート**とは、暗黙の初期ステートとして設定されるステートの事で、特に指定しなくても初期のステートとして登録されるステートの事をいいます。従って、上記の例では明示的に各モータ毎の先頭ステートを指定していますが、各モータのステートの順番が上記の様な例である場合には特に指定しなくても、各モータの先頭ステート（すなわち「通常」）が指定された事になります。例えば、ステートを以下の様な順番で設定した場合には明示的に初期状態のステートを指定しないと意図した動作が行われない場合があります。



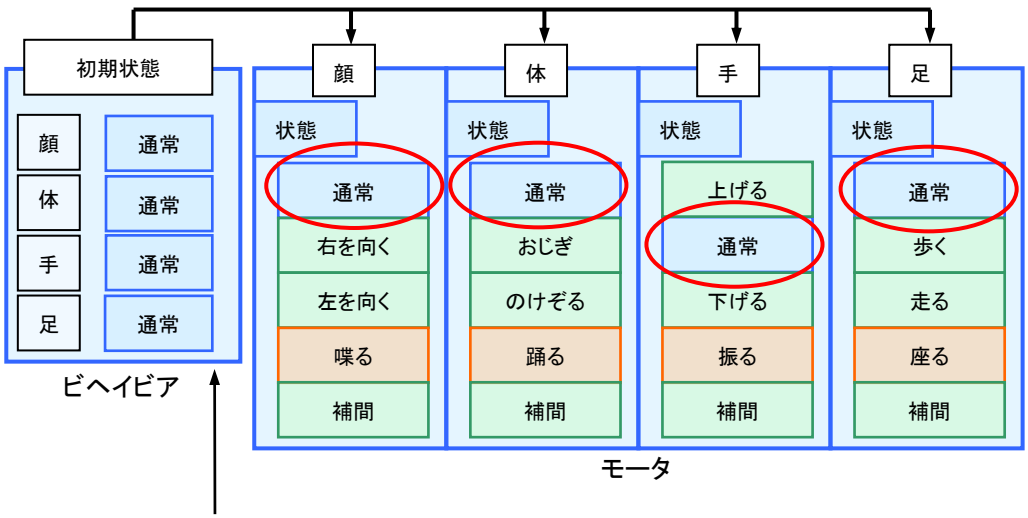
上記の「手」のモータの様に、「通常」の状態として指定したいステートを「手」のモータ内の先頭以外の場所に定義してしまうと、「手」のモータのデフォルトステートは「上げる」ステートに設定されてしまいます。従って、「手」のモータを「初期状態」で「通常」の動作を行いたい場合にはビヘイビアから明示的に定義する必要があります。

＜明示的に指定していない場合＞



特に初期状態を定義しなかった場合には、「手」のモータは「上げる」ステートが設定されてしまい、キャラ電を読込んだ直後から「手をあげる」動作を行ってしまい、想定していた動作とは異なる動作を行ってしまう。

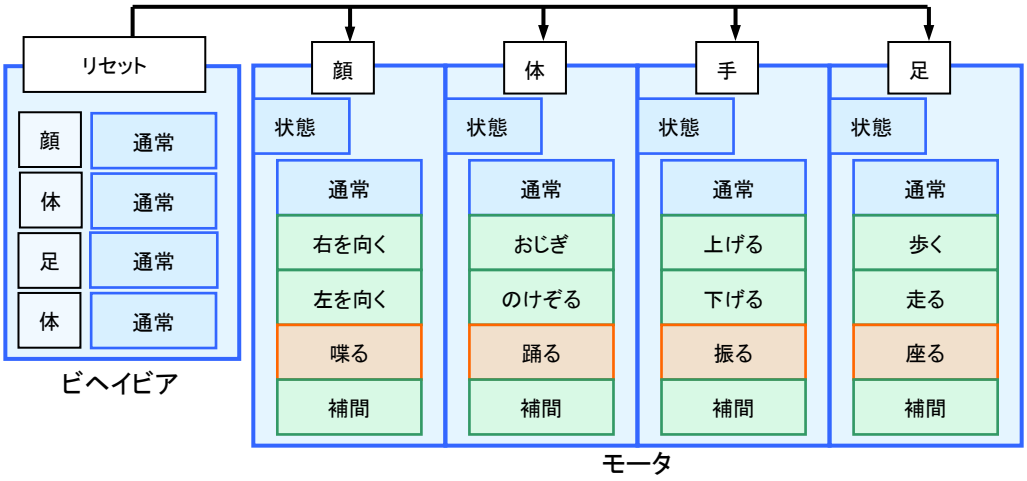
＜明示的に指定した場合＞



初期状態を明示的に指定する事によりデフォルトステートではないステートを初期状態として設定している。この場合、初期状態は期待した動作（すべてのモータが通常動作）を行う。

2. リセット

リセットとは、キャラ電が携帯端末上のキーで「0」を押された場合（実行中アクションの中断があった場合等）の事です。リセットではこの時に行う動作を指定しますが、普通は特別な事がない限り「初期状態」と同じ設定にしておきます。



リセットの場合も初期状態と同様に一度、全てのモータがデフォルトステートに設定されてから、設定された動作を行う様になっています。

3. キー押下時

キー押下時のイベントには細かく分けると4種類あります。前の方でも書きましたが、「全体アクション」、「パーツアクション」、「癖アクション」、「リップシンクイベント」です。携帯端末上でそれぞれのキーが押されると、そのキーに応じてキャラクタを動作させる訳ですが、それぞれのアクションによって割当てるキーにはルールがあります。

【定義可能なキー】

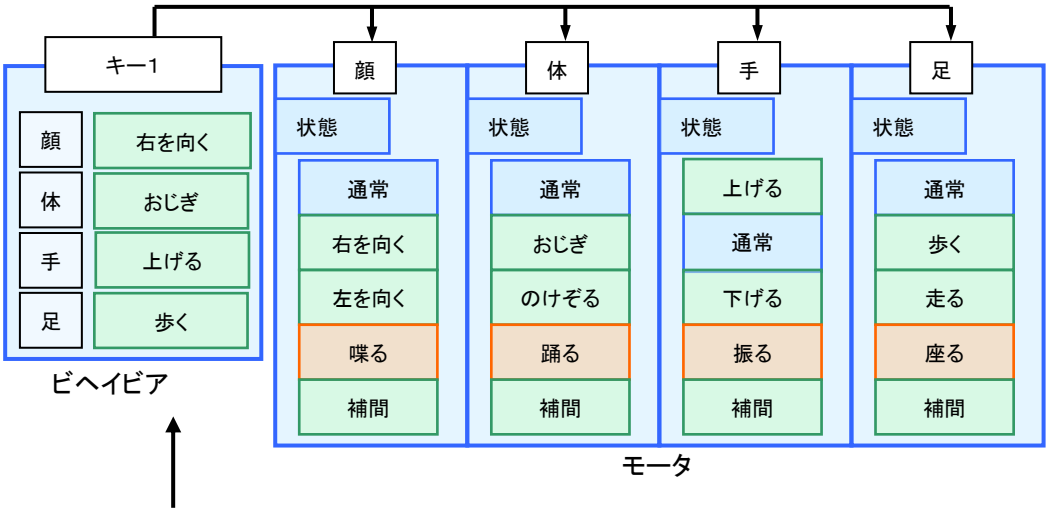
モード		キー押下ルール
癖アクション		#*20
リップシンク		#*00（開始）、#*01（終了）
全体アクション		1～9の短押し ＃1～＃9の短押し
パーツアクション	パーツ	1～9の2桁キーの短押し組合せ （11～19、21～29、･･･91～99）
	隠し	＃から始まり＃で終わるキーの短押し組み合わせ （＃12345＃） ＃間のキーは1～9のみであり押下回数は5個とする

定義を行う際には上記のキーを意識して設定する必要があります。

一般的にパーツを複数用意した場合（モータを複数用意した場合）には全体アクションでは各パーツの状態を定義する事でアクションする様に作成します。また、パーツが一つだけの場合（モータが1つの場合）は一つの状態でアクションを行う様に作成し、その状態を定義します。

＜モータが複数ある場合＞

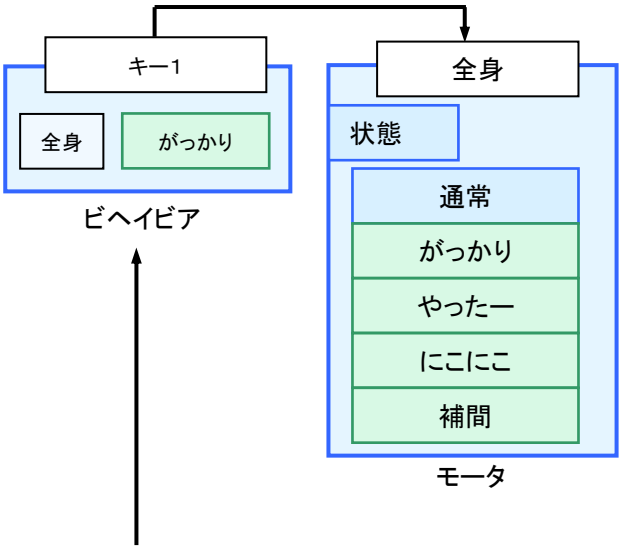
モータが複数ある場合は各モータにそのパーツ毎の状態を用意し、ビヘイビアから各モータ毎の状態を定義する事によりアクションを実現します。



モータに用意した各状態をビヘイビアから定義する事により「キー1」が押された時の状態を設定する事が出来る。

＜モータが1つの場合＞

モータが1つの場合は1つのパーツに複数の状態を用意し、ビヘイビアからはその特定の状態を指定する事でアクションを設定します。

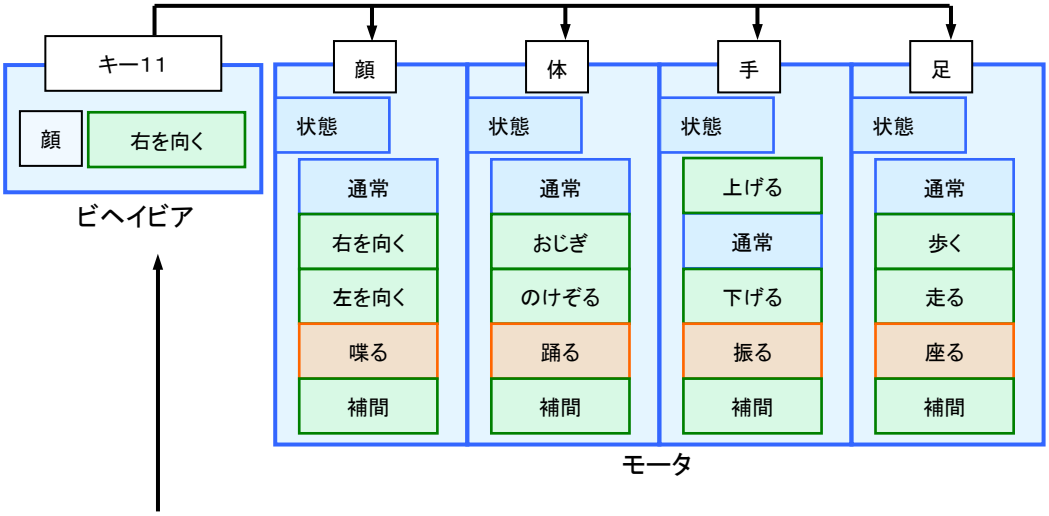


モータがひとつの場合はモータに用意した特定の状態をビヘイビアから定義する事により「キー1」が押された時の状態を設定する事になる。

次にパーツアクションですが、パーツアクション（腕だけを上げるなど）を定義する場合には以下の様に各パーツの特定箇所のみを定義する事になります。

＜モータが複数ある場合の注意＞

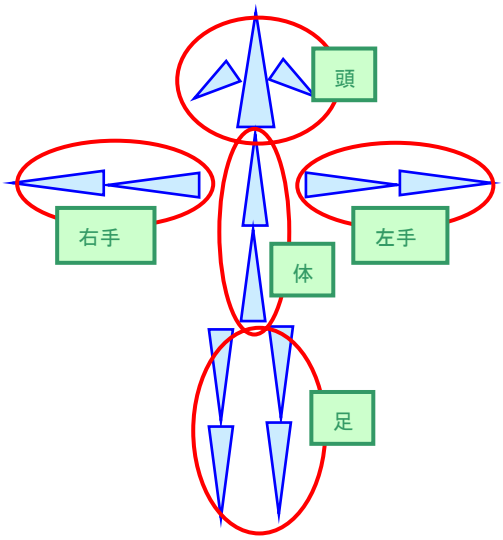
モータが複数ある場合はパーツとして動作させるパーツのみの指定をおこなう事でそのパーツだけの状態を変化させます。



ビヘイビアからは操作したい特定のパーツの状態を設定する事により、他のパーツの状態は変更せずにそのパーツの状態だけを変化させる事でパーツのアクションをさせることができる。

上記の様に各パーツ毎に動作を定義する場合にはアクションを作成する際に以下の様な点に注意する必要があります。

例えば、以下の様に複数のパーツ（モータ）を使用して各パーツを管理する際には重複したキーフレームのあるボーンを作成しない様にする必要があります。



各パーツ毎に動かすボーンを決めて頭のアクション作成時には、頭として決めた部分以外のボーンにキーフレームを作成しない様にする必要があります。

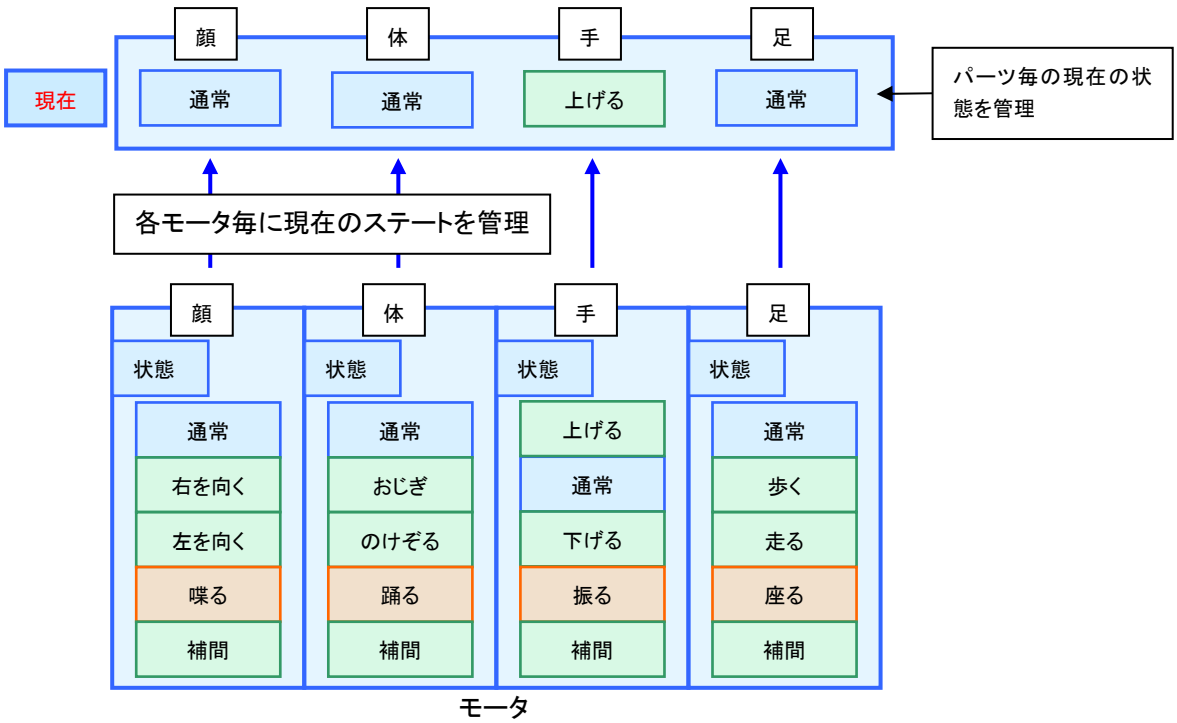
2. ステート(状態)遷移の種類とその方法

上記までの説明でビヘイビアから各モータのステート(状態)を変更する仕組みが理解できたと思います。ここではさらにビヘイビアからの各モータの状態を変更する方法の種類の違いについて説明します。

前項にも出てきましたが、キャラ電の内部ではキャラ電の動作を制御しているキャラ電のエンジンがあります。このキャラ電のエンジン内部ではキャラクタの各モータについて、「**現在状態(現在のステート)**」と「**目的状態(遷移先のステート)**」という2つの状態を管理しています。キャラ電ではこの2つの状態を変更する事によりキャラクタのパーツ毎のステートの遷移を実現しています。実際にキャラ電のデータを作成する際にはこの概念についてもある程度の理解が必要になってきます。ここではまずこの2つの状態の管理方法について説明します。

(1)キャラ電が管理する「**現在状態(現在のステート)**」
キャラ電の内部では、キャラクタのパーツ(モータ)が今どのステートなのかを各パーツ毎に管理しています。

キャラ電内部の「**現在状態(現在のステート)**」



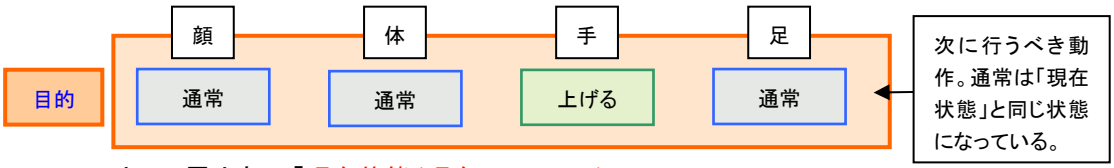
モータ

キャラ電が内部で管理している「**現在状態**」とは各パーツ毎の現在のステートの事で、現在画面に表示されている「現在のパーツ毎の動作」の事になります。上記の例では現在のステートに設定されている動作、すなわち「キャラクタの顔・体・足は通常の状態で、手を上げている状態」が画面に表示されており、「現在のステート」である事になります。

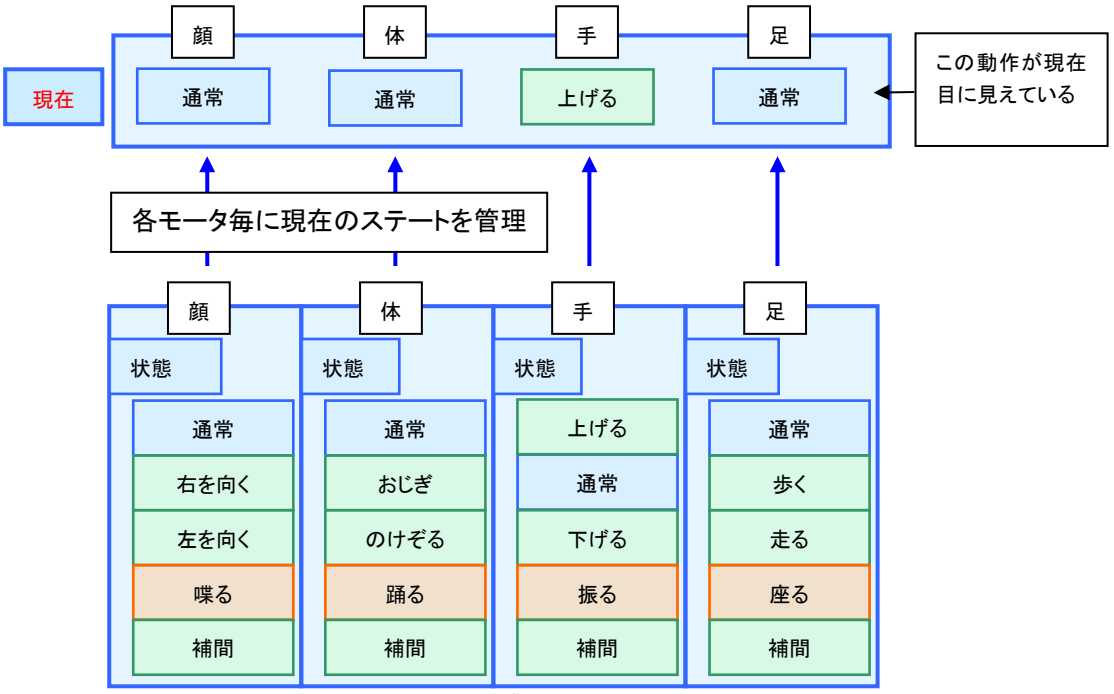
(2)キャラ電が管理する「**目的状態（遷移先のステート）**」

キャラ電では、「現在状態」とは別に「次にどの動作をすべきか」を管理するために、キャラ電の内部に「目的状態（遷移先のステート）」というものをパーツ毎に管理しています。これはキャラクタの各パーツが次にどの状態に遷移すべきかを管理しており、補間アクションの様に複数の動作を経てある状態に達する動作を行いたい場合に設定を行うものです。**「目的状態」**が変化するとキャラ電は**「現在状態」**から**「目的状態」**に遷移しようとします。これにより現在の状態からの動作が開始される事になります。通常、何も設定されていない場合は「現在状態」と「目的状態」は同じ状態になっています。

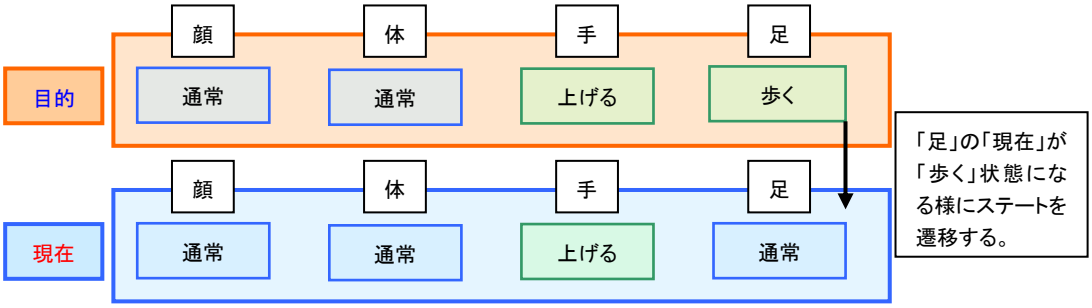
キャラ電内部の「目的状態（遷移先のステート）」



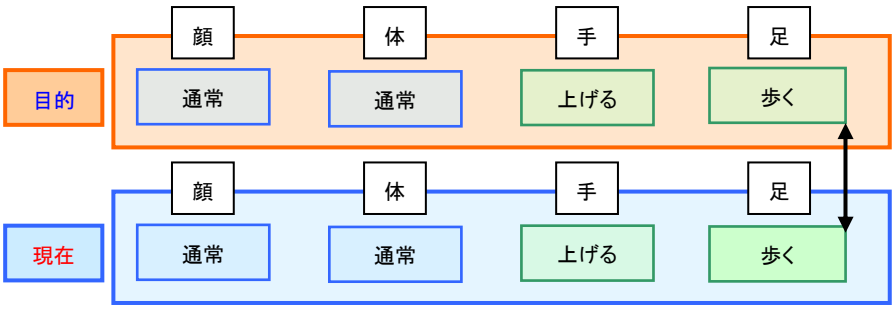
キャラ電内部の「現在状態（現在のステート）」



キャラ電は「目的状態」が変更されると、「現在状態」と「目的状態」が同じになる様に状態の遷移を開始します。（状態を変更し始めます）

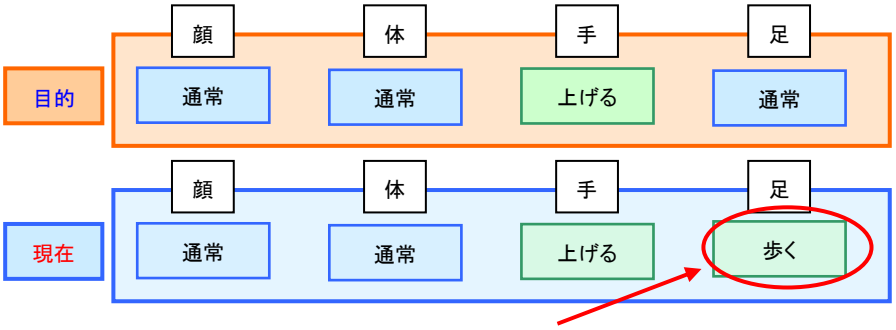


この様に「目的状態」に遷移先のステートを設定する事で画面に見えているキャラクタの動作が開始される事になります。上図の例では動作を開始して、「足」が「歩く」状態になるように状態の遷移し続けます。



「足」の「現在状態」が「歩く」状態になった時点でステートの遷移は終了します（厳密には、「歩く」状態への遷移を繰り返します）。この時画面に見えているキャラクタは「顔・体は通常の状態で、手は上げており、足は歩いている」動作をしている事になります。

「目的状態」に次の状態を設定するとキャラクタの状態を遷移させる事が分かりましたが、「**現在状態**」を変更する事によっても状態を遷移させる事が出来ます。



「現在状態」を変更した場合には、キャラクタは指定されたステート（状態）にすぐに遷移します。画面に見えているキャラクタは指定された動作を即時に開始します。「目的状態」を変更した場合との違いは、以前に出てきた「目的の動作を行うまでの補間アクションの動作を伴わない」という点です。

上記までの実際の例では以下の様な違いが出てきます。

a.「**現在状態**」の「足パーツ」を「歩く」状態に変更した場合
画面に見えているキャラクタの足は最初は通常状態だったものが突然歩く動作を行っている状態になります。（急に歩きだす）

b.「**目的状態**」の「足パーツ」を「歩く」状態に変更した場合
画面に見えているキャラクタの足は最初は通常状態だったものが、まず、補間アクションの動作（この例であれば歩きだす様な動作）を行ってから歩いている状態になります。（スムーズに歩きだす）
※補間アクションはモータで遷移先を設定する事により実現します。（後述）

ここまでの説明でキャラクタの各パーツの状態を遷移させるには、「現在状態」を変更する方法と、「目的状態」を設定する方法の2種類があることが分かりました。

キャラ電スタジオでは、キャラクタのパーツの状態を遷移させる方法をそれぞれ以下の様に呼び、区別しています。

状態の変更方法	意　味
「実行」(set)	「現在状態」の状態を変更します。
「設定」(play)	「目的状態」の状態を変更します。

一般的なガイダンスとしては、ビヘイビアの各イベントから各モータのステートを設定する際には以下の変更方法を使用して状態を設定する事になります。

イベント	状態の変更方法
初期状態	「実行」(set)により状態を変更する。
リセット	「実行」(set)により状態を変更する。
全体アクション	「実行」(set)により状態を変更する。
パーツアクション	「設定」(play)により状態を変更する。
パーツアクション（ループ）	「設定」(play)により状態を変更する。
癖アクション	「実行」(set)により状態を変更する。
リップシンク	「設定」(play)により状態を変更する。

この様に、補間アクションを伴うループアクションは「設定」によって次の状態を指定する事になります。

3. モータにおけるステート（状態）遷移

今までの説明ではビヘイビアでは「現在状態」や「目的状態」に各モータのステート（状態）を変更する事で状態が遷移するという事について説明してきましたが、今度はその状態の変化に応じたステートの遷移について説明します。

ステートでは以下の様に各ステートのタイミングに応じたイベントを設定する事が出来る様になっています。

＜ステートで設定できるイベント＞

イベント種類	意　味
開始(start)	「現在状態」が指定されたステートに到達した時
終了(end)	「現在状態」のアクションが終了した時
イベント(event)	「目的状態」が変更された時
間隔(time)	「現在状態」がステートに到達してからの経過時間を迎えた時

各イベントでは状態の遷移のタイミングを取得する事ができますので、そのタイミングで何をするかの指定を行う必要があり、これを「コマンド」と呼びます。各イベントで指定可能なコマンドは以下の通りです。

＜ステートのイベントで指定できるコマンド＞

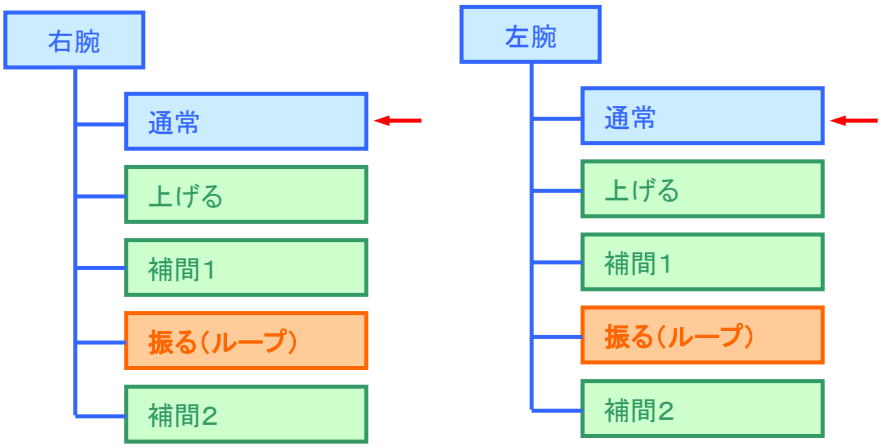
イベント種類	意　味
前景(setfg)	前景画像を設定する(透過が行なわれます)
背景(setbg)	背景画像を設定する(透過は行なわれません)
遷移(goto)	指定されたステートに遷移する
遷移先設定(next)	遷移先のステートを設定する（実際の遷移は行わない）
遷移先へ(gotonext)	保存されている遷移先のステートに遷移する（設定されている遷移先がない場合は現在のステートに戻る）

・ステートの遷移方法

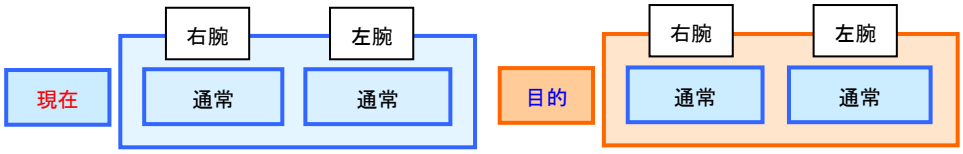
各ステートではイベントの指定を行う事で、そのステートの中で「いつ」「何をするか」という事が指定できます。また、ビヘイビアでの「現在状態」や「目的状態」の変更によってもステートの遷移が行われますので、このあたりも合わせて説明します。以下に例としてよく使用されるパターンを挙げて説明します。

以下の例では、右腕と左腕の両モータについて前項で説明したキャラ電内部の管理方法と併せてどのように遷移するのかを説明します。

＜右腕と左腕モータのステートの例＞

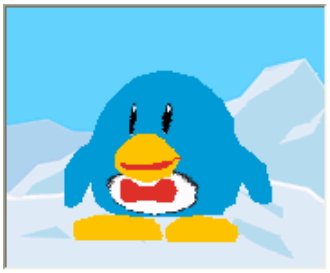


上の図では、右腕と左腕のモータに存在するステートを記述してあります。また、下の図では前項で説明したパーツ（モータ）毎の「現在状態」と「目的状態」がそれぞれ「通常」である状態を表しています。



この状態ではキャラ電の画面にはそれぞれ右腕・左腕ともに「通常」に設定されているアクションが実行されて表示されていることになります。

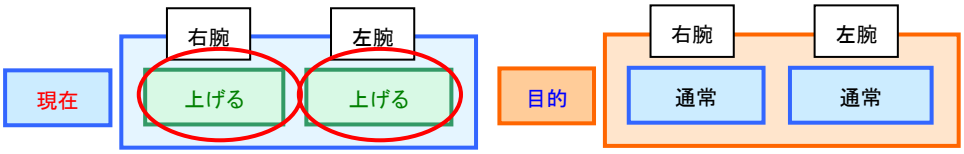
【右腕、左腕が「通常」ステートにある状態】



注：実際には右腕と左腕だけのモータという事は有り得ませんが説明のため便宜上2つのモータで説明します。

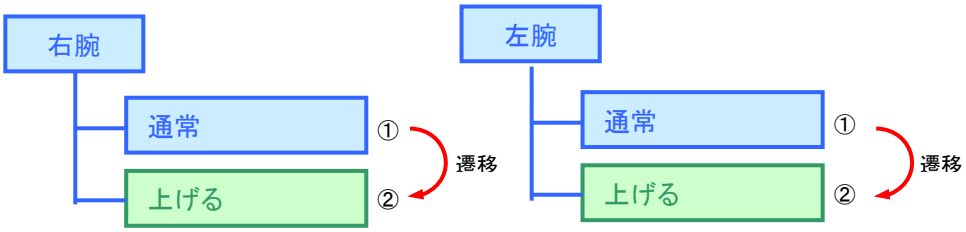
ここで、携帯端末からのイベントとして1のキーを押された場合を考えます。ビヘイビアでは携帯端末からのイベントを取得してキャラ電内部の「現在状態」や「目的状態」を変更しますので、ここでは「現在状態」の「右腕」「左腕」のそれぞれの状態を「上げる」に変更したとします。

＜例として1のキーが押された. . . ＞



ビヘイビアで「実行(set)」によって「現在状態」の「右腕」「左腕」を「上げる」に変更します。すると、「現在状態」が変更されたので、ステートの遷移が行われます。

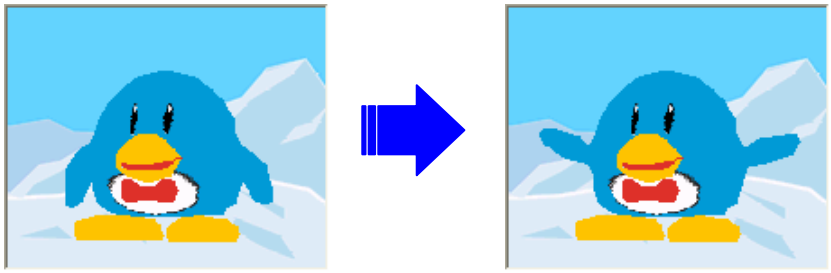
＜右腕と左腕モータのステートの例＞



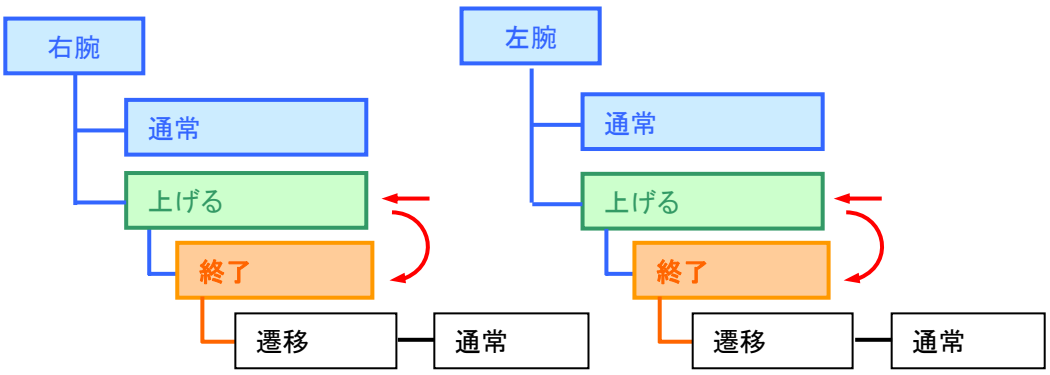
「現在状態」の変更によって各モータの遷移が行われます。この例では両腕のモータがそれぞれ「上げる」状態に遷移することになります。これによりキャラ電の画面にはキャラクタが両腕をあげている動作が表示されます。

各ステートの遷移により、アクションが再生される

【通常ステート】　　【上げるステート】

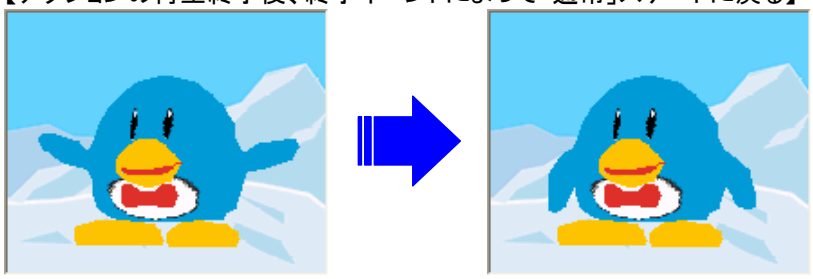


次に、キャラクタが両腕を上げ終わった後にまた通常の状態に戻す事にします。
その場合にはステートにアクションの終了時のイベントを設定する事で状態の遷移を行います。

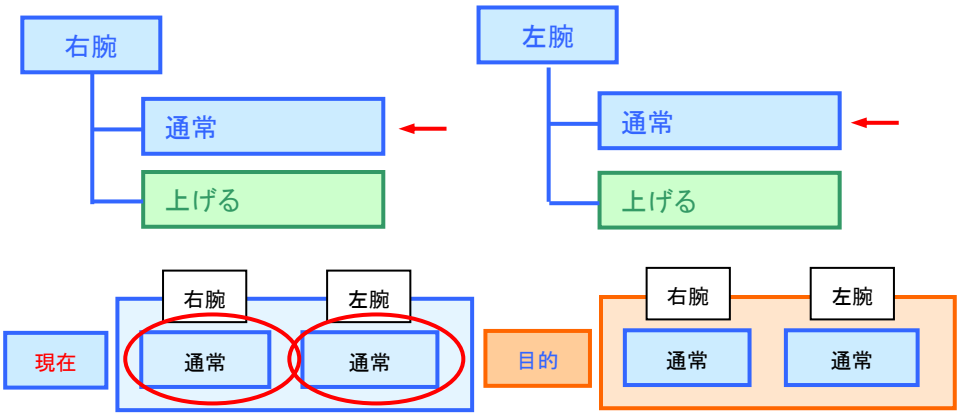


ステート「上げる」に**イベント**として「終了」を設定し、コマンド「遷移」の遷移先として「通常」を指定します。この事により「上げる」ステートにおけるアクションの終了時に「通常」に遷移することになります。

【アクションの再生終了後、終了イベントによって「通常」ステートに戻る】

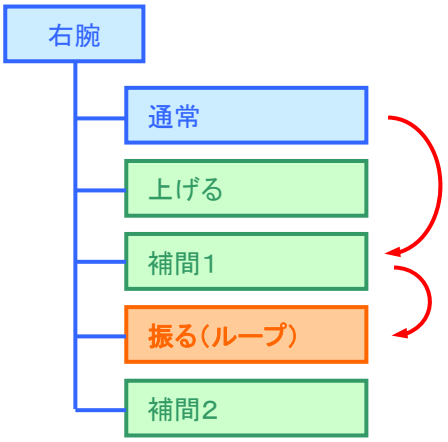


ステートでの終了イベントに「通常」ステートへの遷移の指定がある為、各モータのステートは「通常」に戻っています。



次に、ループするパーツアクションの動作を見てみます。
今度は携帯端末から「右腕だけを振る」というパーツのループアクションのキー入力があったとします。(例:キー11の入力など)
以前にも説明しました様にループアクションでは補間のアクションを行う必要がありますので、ステート側での遷移は以下の様に遷移させる必要があります。

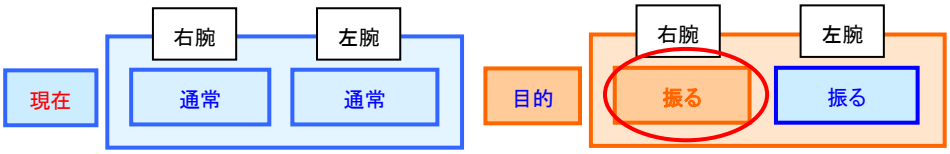
＜右腕のモータ＞



補間を行うアクションのステートの遷移は「通常」から「補間1」を通り「振る(ループ)」へと遷移させる必要がある。

「通常」ステートから「補間1」アクションを行い「振る(ループ)」アクションを行う事でスムーズなループアクションへの移行となります。
補間アクションが存在する場合はビヘイビアではキーイベントを取得した時に「目的状態」を設定する様にしておきます。

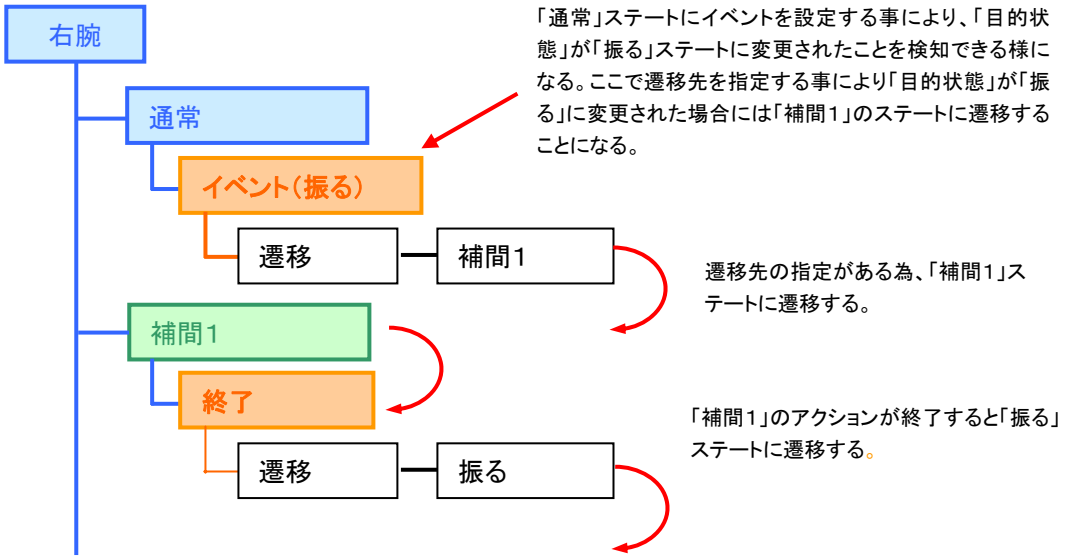
＜例として11（パーツ・ループ）のキーが押された．．．＞



ビヘイビアで「設定(play)」によって「目的状態」の「右腕」を「振る」に変更します。すると、変更された「右腕」について、ステートの遷移が行われます。この時、変更された内容が「目的状態」であるため、ステートの遷移はそのモータに設定されている内容に従って遷移を行います。

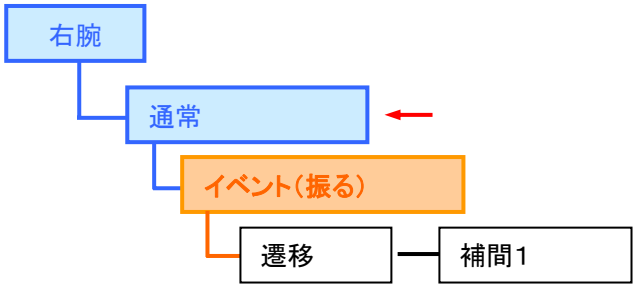
その為、ステートでは目的状態が変更された時のイベントを取得してステートを遷移させます。

＜右腕モータのステートの例＞



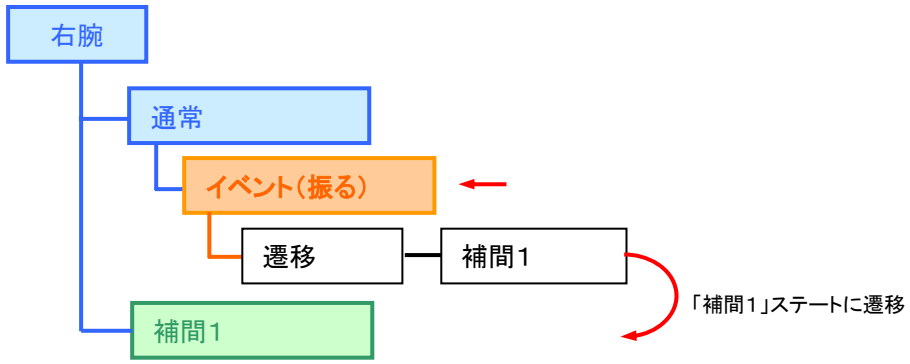
この場合、「通常」ステートには目的状態の変更が行われたかどうかを検知するイベントを設定しておきます。イベントとして「振る」への目的状態の変更が行われたかどうかのイベントを設定しておくと、ビヘイビアで目的状態が変更された場合に変更されたことをハンドリングしてステートを遷移させる事が可能になります。

＜「通常」状態＞

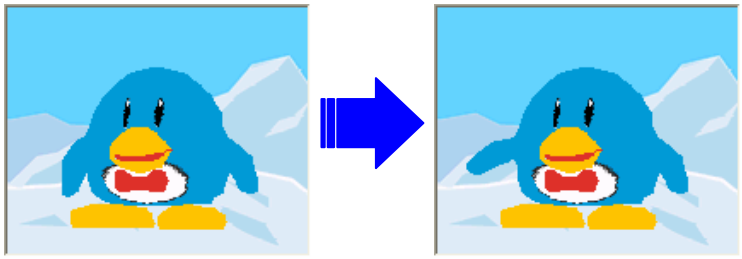


ビヘイビアより「目的状態」が「振る」に変更されると、「通常」ステートでこのイベントをハンドリングして「補間1」に遷移します。

＜キー１１が押された直後＞

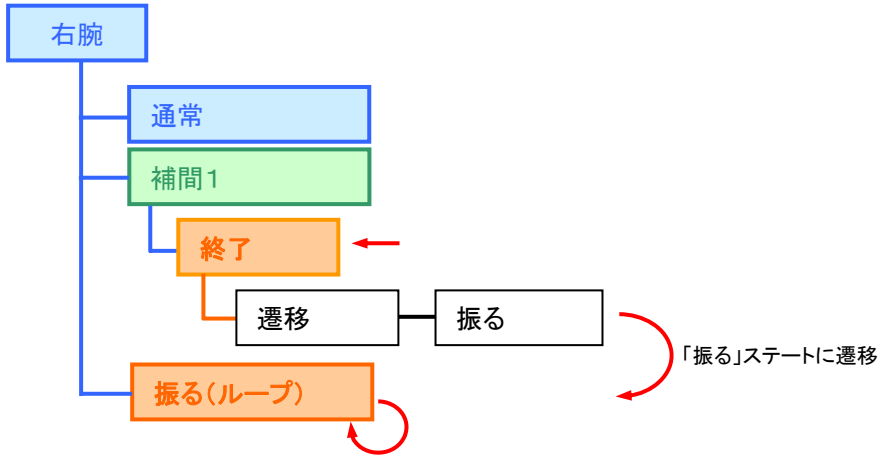


【通常状態】　【補間１】

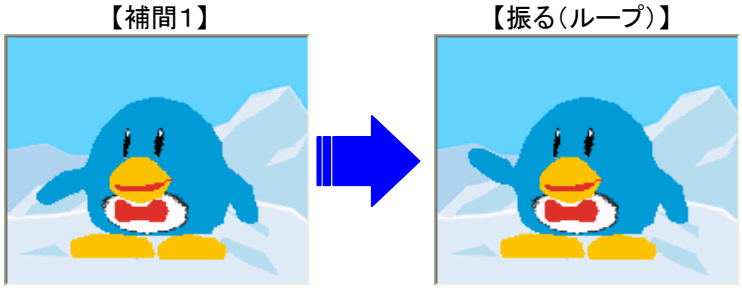


状態が遷移し、「補間１」のアクションが行わます。
「補間１」のアクションが終了すると同時に「補間１」ステートの「終了イベント」により次の状態に遷移します。

＜補間アクションが終了した直後＞

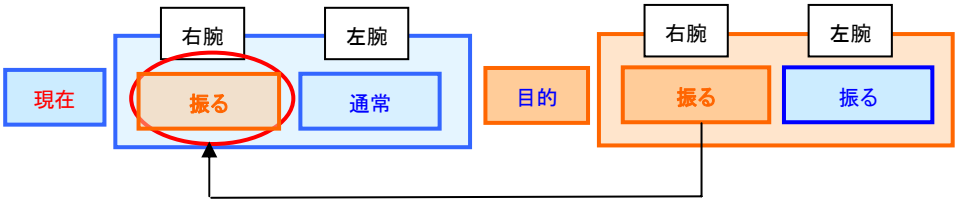


【補間１】　【振る（ループ）】



補間１のアクションが終了すると終了イベントにより目的状態である「振る」ステートへの遷移が行われます。この遷移が行われた時点で「現在状態」と「目的状態」が同じになり、キャラ電内部のエンジンは状態の遷移を終了します。

＜現在状態と目的状態＞



キャラ電は「目的状態」に設定された「振る」ステートへ到達すると「現在状態」に「振る」を設定し、ステートの遷移を終了します。

この様な設定を行うことによりキャラ電が以下の様に動作します。

＜キー11（パーツ・ループ）を押されると…＞

【通常状態】　【補間1】　【振る（ループ）】

通常状態から補間動作を経て振る動作を行います。

ここまでで説明してきました様に携帯端末からのキー入力によりビヘイビアで各モータのステートへの遷移先を指定して、各モータのステートではそれぞれのイベントをハンドリングしながら必要な遷移先を指定する事でアクションを連続して再生することがわかりました。

さらに、背景や前景を各タイミングで変更したりすることも可能になっていますが、このあたりの説明は別紙チュートリアルを参照してください。

6.7 まとめ

これまでのことからキャラクタの動作を作成するときには以下の様なガイダンスを元に作成する事になります。

1. 全体アクションの動作を決める
2. パーツアクションの動作を決める
3. ループアクションの動作を決める
4. ループアクションでは状態を遷移する際に補間アクションを考える
5. 各アクションで動作させるパーツを決定する。
6. 各パーツ毎に遷移図を作成する
7. 遷移図を元に各パーツ毎に状態を洗い出す
8. ビヘイビアとモータの概念を元にイベントとアクションを設定する

また、アクションについては以下の様なガイダンスとなります。

1. 全体アクション
   キャラクタ全体を使用して表現するアクションの事で基本的には複数のパーツ（モータ）から構成されます。ビヘイビアからは実行（set）により全パーツの状態を指定します。
2. パーツアクション
   キャラクタのパーツを動作させるアクションの事で基本的には単一のパーツ（モータ）で構成されます。（尚、キャラクタ全体が単一のパーツ（モータ）で構成される場合はどのアクションも全体アクションとなる）ビヘイビアからは設定（play）によりパーツの状態を指定します。

その他、ビヘイビア、モータを扱うにあたり注意しなければならない点は以下の通りです。

1. モータの優先順位
   これまで説明してきました通り、各モータのステートにおいてキャラクタの各々のパーツを制御することになりますが、各パーツでの制御の際に他のパーツを操作する様な動作がモータ間を跨いで同時に発生した場合には先に設定しているモータでの設定が有効になります。（ボーンの衝突）

2. ステートの優先順位
   前で説明しましたが、先頭のステートはデフォルトステートとなります。

3. 背景・前景を連続で設定した場合
   背景を連続で設定した場合は毎回描画が行なわれませんので、画面には最後の背景が設定されます。

4. テクスチャアニメーション
   モータを跨いでテクスチャアニメーションの設定を行う場合にはテクスチャアニメーションが衝突（同時に再生）しない様に注意して下さい。衝突した（同時に再生が行なわれた）場合にはモータの優先順位に従って動作します。

5. 作成するアクションの長さ
   キャラクタのアクションを作成する際には、同時に行うパーツ毎のアクションの長さ（時間）は同じ長さになるように作成することをお勧めします。これは、例えば両腕を同時に上げる場合に右腕と左腕のパーツで各々の腕を上げるアクションの時間が異なるとその次の状態遷移を行ったときに各パーツがずれて再生されてしまうからです。（設定により回避可能ですが設定が複雑になるので基本的には同時に再生するパーツ毎のアクションの長さは同じにしておくことをお勧めします。）

7. キャラ電コンテンツ作成におけるガイドライン

前章までで、キャラ電に関する動作の原理が把握できたと思います。
ここではキャラ電コンテンツを作成する際のガイドラインをまとめておきます。

7.1 アニメーションの定義

以下にキャラ電データが動作するアニメーションの定義を行ないます。

ボーンアニメーション（3D）

モデルデータを構成するボーン(骨)を動作させることにより、アニメーションを行ないます。

テクスチャアニメーション（3D）

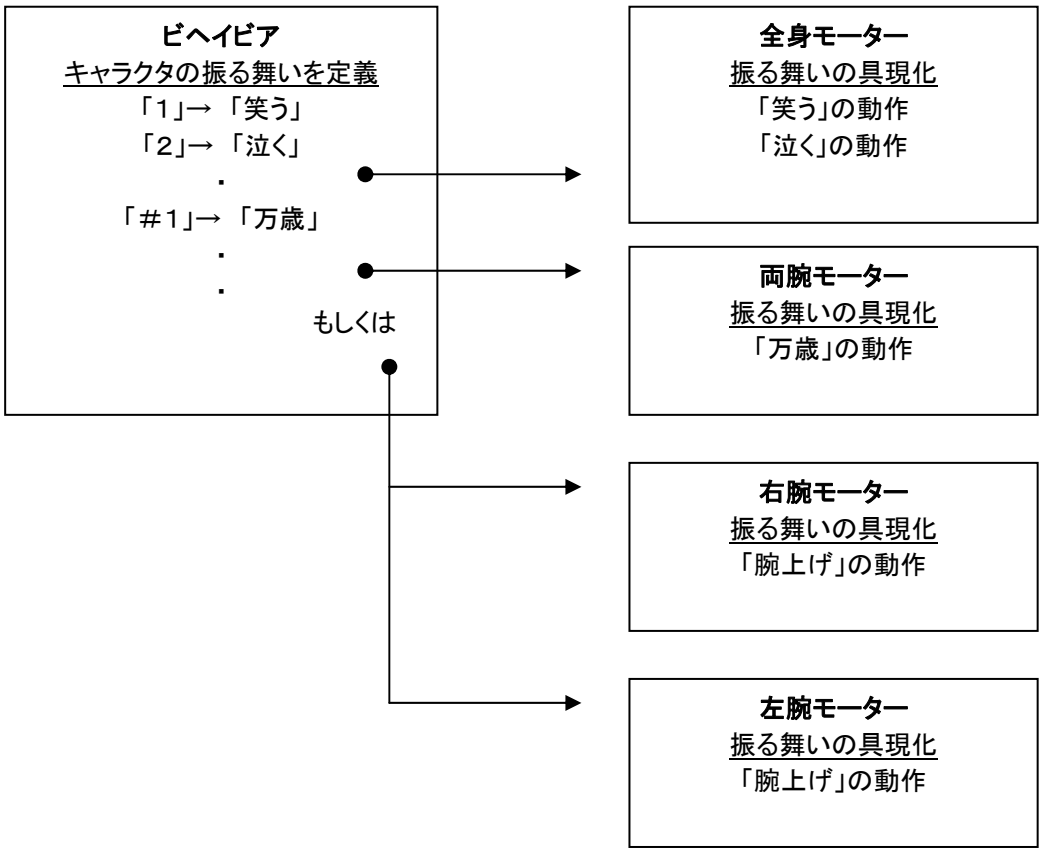
モデルデータに貼り付けたテクスチャのポリゴンをダイナミックに貼りかえることにより、アニメーションを行ないます。

2D アニメーション

モデルデータを用いないでパラパラ漫画的に、ある間隔でイメージの表示を行ないます。

7.2 モーター(パーツ)の考え方

キャラ電を作成する際には、ビヘイビア・モーターという概念を使用します。
ビヘイビアとは、モデルの状態を定義し、モーターとは、ビヘイビアで定義した振る舞いを具現化する動作を定義するものとなります。
例えば、「1」キーが「万歳」と関連付けることがビヘイビアにあたり、「万歳」という動作がモーターに相当します。この「万歳」という動作は「両腕を上げる」という動作により具現化されますが、1つのモーターで「両腕」を、また、2つのモーターでそれぞれ「左腕」「右腕」を扱うことができます。複数のモーターを扱うことをマルチモーターと呼びます。
このように1つのモーターの動作を1つのパーツとして捉えることが可能となります。



7.3 アクション

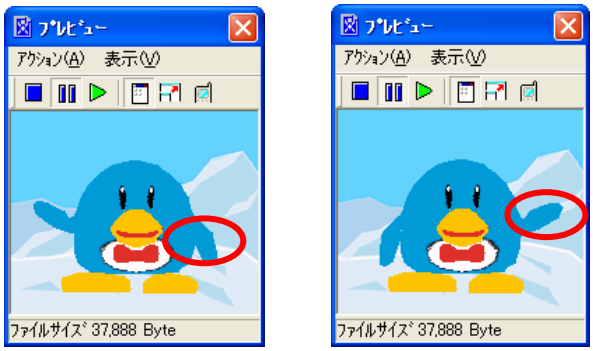
モーターで実現するモデルの動作のことをアクションと呼びます。
キャラ電は、ユーザーの操作により様々なアクションを行ないますが、そのアクションのさせ方により、モーター（パーツ）の捉え方が変わってくきます。
全身を扱うアクションのみのキャラ電は、「腕」「足」といった部品のアクション合成の必要がないため、全身を1つのモーターとして扱うことになります。
また、全身も扱うが、「腕」「足」といった部品のアクションも行うキャラ電は、各々アクション合成が必要になるため、複数のモーターに分けて扱うこととなります。

左図のように、上げた両腕を1つのパーツとして扱うか、左腕・右腕を別々のパーツとして扱うか、でモーターの構成が変わります。

7.4 パーツ分け

モデルに対するアクションは、全身のアクションとして作成する場合や、ある意味を持つパーツ単位で作成する場合が存在します。この場合に注意しなければいけないのは、ボーンアニメーションの衝突です。例えば、「万歳」アクションを実現する場合に「両腕」を1つのモーターとするケース、「左腕」「右腕」として2つのモーターとするケースが考えられます。前者の場合、「両腕」モーターが動作する上で、他のモーターで「両腕」と同じボーンアニメーションが再生されると「ボーンの衝突」が発生します。
競合解決は後述するとして、意図しないアクションとなる可能性があります。
また、後者の場合、「左腕」モーターの中に「右腕」モーターと同じボーンが、または、「右腕」モーターの中に「左腕」モーターと同じボーンが存在すると同じ結果となります。

例えば、下図のように「右腕を上げる」と「左腕を上げる」アクションがあり、同時に再生した場合に、「右腕を上げる」アクションの中に「左腕」ボーンのアクションが存在すると、「左腕」ボーンの衝突が発生します。



全身のアクションのみの作成時は問題ないが、パーツ単位のアクションを作成する必要がある場合は、モーター単位で使用するボーンを決定しボーンが衝突しないようにパーツ分けすることを推奨します。

キャラ電では、全体アクションとパーツアクションの2つのアクションモードが存在します。
上記モーターとアクションの考え方に基づき、アクションモードの定義を行ないます。

7.5 全体アクションの定義

・全身アクションのみで1つのモーター（パーツ）から構成されるアクション
・複数のモーター（パーツ）から構成されるアクション

7.6 パーツアクションの定義

・単一のモーター（パーツ）から構成されるアクション
※ 隠しアクションもパーツアクションに含まれます

7.7 状態遷移

キャラ電を作成する際の全体アクションとパーツアクションの状態遷移について定義すします。

7.7.1 全体アクション

即座にアクションの切り替えを要求される可能性が高いため、各モーターの状態指定は、ビヘイビアの「実行（set）」で行うようにします。
例）

上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

7.7.2 パーツアクション

アニメーションの補間が必要となる可能性が高いため、モーターの状態指定は、ビヘイビアの「設定（play）」で行うようにします。
例）

上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

7.8 2Dアニメーション

2Dアニメーションは、3Dアニメーションと違いモデルデータ・アクションデータを使用しません。使用するのは前景・背景のみです。

7.8.1 2Dアニメーションの制御

2Dアニメーションにおいて、モーターで制御する方法を以下に示します。

7.8.2 間隔（TIME）

ビヘイビアで定義されたアクションが複数枚のイメージで表現する場合、間隔（TIME）の設定を行ないます。
間隔（TIME）は、そのステートに遷移してからの経過時間が、間隔（TIME）で指定する時間（ミリ秒）に達したときに実行されます。
例）

上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

※ 2D はアクションファイルがないため、終了（END）は使用できません
※ 2D キャラ電の詳細な作成方法はｷｬﾗ電ｽﾀｼﾞｵﾁｭｰﾄﾘｱﾙ P.13 を参照してください。

7.9 アニメーションの競合

各モーター間のボーンの衝突やテクスチャアニメーションの衝突について定義します。

7.9.1 ボーンの衝突

前述のようにマルチモーターの場合に、ボーンの衝突が発生することがあります。
基本的には、パーツ分けの時点で各モーター（パーツ）間でのボーンアニメーションが衝突しないようにアクションを定義すべきです。
ボーンアニメーションの衝突が発生した場合に、モーターの優先順位により決定されます。

例）優先度が体→リップ順になる。

上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

※ 但し、ボーンアニメーションによっては、基本姿勢（アクションしない）ボーンも含まれます。この場合は、それを除いた優先度の高いアニメーションが優先されます。

7.9.2 テクスチャアニメーションの衝突

テクスチャアニメーションは、１つのモーターで実現するようにします。
テクスチャアニメーションの衝突が発生した場合に、モーターの優先順位により決定されます。

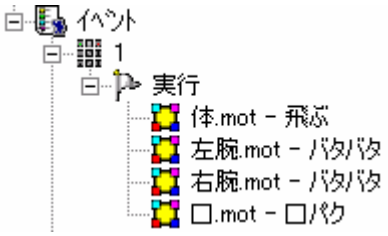
※ 但し、ボーンアニメーションとは異なり、テクスチャアニメーションは、その位置に関係なく衝突が発生します。例えば、口パクと目パチ等

7.10 イベント文字列

キャラ電の動きをイベント文字列で制御を行ないます。

7.10.1 ユーザー操作アクション

ユーザーのキー操作文字列を、イベント文字列としてビヘイビアのイベント（event）で定義します。
例）



上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

7.10.2 癖アクション

内部から発生させるイベントとなり、イベント文字列は「＃＊20」となり、ビヘイビアのイベント（event）で定義します。

7.10.3 簡易リップシンク

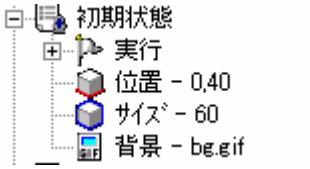
移動機のアプリケーションが口の動きを判別し、リップシンク開始は「＃＊00」、リップシンク停止は「＃＊01」をエンジンへ送出するので、ビヘイビアのイベント（event）で定義します。

7.10.4 トラッキング

※ トラッキングはサポートしません。

7.10.5 デフォルトアクション

キャラ電データのデフォルトの状態を、ビヘイビアの初期状態（start）で定義します。
基本的にはモデルの位置やサイズ、初期状態のアクションを設定します。
例）



上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

7.10.6 アクションリセット

キャラ電データのアクションリセット時の状態を、ビヘイビアのリセット（reset）で定義します。
例）

上図はキャラ電スタジオでの設定方法です。
（詳細はキャラ電スタジオ操作説明書を参照して下さい。）

※ ビヘイビア・モーターの記述方法詳細は P57 以降のスクリプトリファレンス参照の事

7.11 簡易リップシンクの実現方法

簡易リップシンク（口パク開始・停止）をサポートします。
簡易リップシンクの実現手段には、ボーンアニメーション及びテクスチャアニメーションによる方法があります。

ボーンアニメーションによるリップシンク
口の動きをボーンアニメーションで実現するようなモデルが該当します。

1）リップシンク用アクションの作成
　リップシンクするボーンのみのアクションを作成し、1つのモーターとして実現します。

2）デフォルトアクションの作成
　デフォルトアクションのデータを作成し、1つのモーターとして実現します。

3）モーターの優先順位設定
　ユーザー操作アクションのモーターより、低い優先順位にリップシンク用モーター、優先度最下位にデフォルトアクションのモーターとします

このような作りにすることにより、リップシンクはデフォルトアクションより優先されて、キー操作アクションより優先されないので、ユーザー操作アクションとのボーンアニメーションの衝突がないケースのみ口パクを表示するようになります。
注意）全体アクションは、口の動きも含めるように作成します。

7.12 テクスチャアニメーションによるリップシンク

口の動きをテクスチャアニメーションで実現するようなモデルが該当します。

1）リップシンク用アクションの作成
リップシンクするテクスチャアニメーションのみのアクションを作成し、1つのモーターとして実現します。

2）デフォルトアクションの作成
デフォルトアクションのデータを作成し、1つのモーターとして実現。

3）モーターの優先順位設定
ユーザー操作アクションのモーターより、低い優先順位にリップシンク用モーター、優先度最下位にデフォルトアクションのモーターとします。

このような作りにすることにより、リップシンクはデフォルトアクションより優先されて、キー操作アクションより優先されないので、ユーザー操作アクションとのテクスチャアニメーションの衝突がないケースのみ口パクを表示するようになります。
注意）全体アクションは、口の動きに限らず何らかのテクスチャアニメーションを含めるように作成します。

7.13 癖アクションの実現方法

癖アクションの実現手段には、ボーンアニメーション及びテクスチャアニメーションによる方法があります。

7.13.1 ボーンアニメーションによる癖アクション

1）癖アクションの作成
癖を行うボーンのみのアクションを作成し、1つのモーターとして実現します。

2）デフォルトアクションの作成
デフォルトアクションのデータを作成し、1つのモーターとして実現します。

3）モーターの優先順位設定
ユーザー操作アクションのモーターより、低い優先順位に癖用モーター、優先度最下位にデフォルトアクションのモーターとします

このような作りにすることにより、癖はデフォルトアクションより優先されて、キー操作アクションより優先されないので、ユーザー操作アクションとのボーンアニメーションの衝突がないケースのみ癖アクションを表示するようになります。
注意）リップシンクモーターより優先させる事

7.13.2 テクスチャアニメーションによる癖アクション

1）癖アクションの作成
癖を行うテクスチャアニメーションのみのアクションを作成し、1つのモーターとして実現します。

2）デフォルトアクションの作成
デフォルトアクションのデータを作成し、1つのモーターとして実現します。

3）モーターの優先順位設定
ユーザー操作アクションのモーターより、低い優先順位に癖用モーター、優先度最下位にデフォルトアクションのモーターとします。

このような作りにすることにより、癖はデフォルトアクションより優先されて、キー操作アクションより優先されないので、ユーザー操作アクションとのテクスチャアニメーションの衝突がないケースのみ癖アクションを表示するようになります。
注意）リップシンクモーターより優先させる事

7.14 イベントの競合解決

モーターの優先順位による競合解決を行うため、それを考慮した上でキャラ電データの作成を行う必要があります。

7.14.1 イベントの優先順位

イベントの優先順位を以下に示します。

1. ユーザー操作アクション
2. 癖アクション
3. リップシンク
4. デフォルト

7.14.2 イベントのマージ

排他関係にあるイベントが競合した場合の解決結果を以下に示します。

各イベント			イベントの競合解決結果
キー操作	癖	ﾘｯﾌﾟｼﾝｸ	
×	×	○	リップシンク
×	○	×	癖
○	×	×	キー操作
×	○	○	癖、もしくは、（ボーン及びテクスチャアニメーションの衝突がない場合は）癖とリップシンクのマージ
○	×	○	キー操作
○	○	×	キー操作
○	○	○	キー操作

7.15 まとめ

以下に示す方法が基本的なキャラ電データ作成方法となります。

モーターの構成
キャラ電データは以下のモーターから構成されます。

全体アクション
全身のアクションを、1つのモーター、もしくは、複数のパーツアクションのモーターで実現します。

パーツアクション
ある意味をもつパーツ毎のモーターとして実現します。

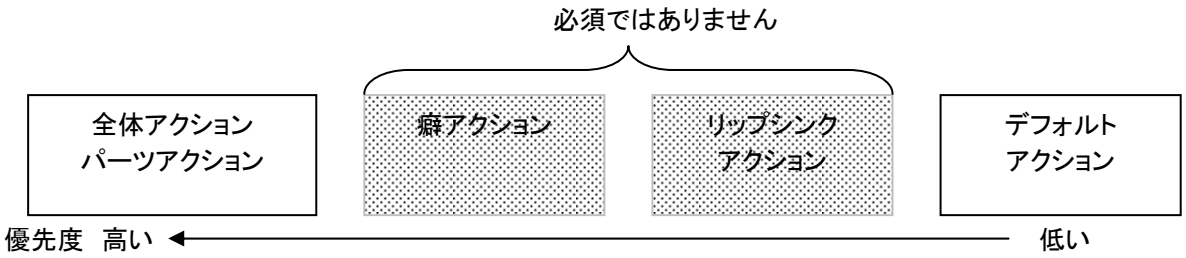
癖アクション
癖のアクションを癖モーターとして実現します。

リップシンクアクション
リップシンクのアクションをリップシンクモーターとして実現します。

デフォルトアクション
全身のデフォルトのアクションをデフォルトアクションモーターとして実現します。

7.16 モーターの優先順位

各モーターの優先順位を以下に示します。



7.16.1 ボーンアニメーション

優先順位が下位のモーターより、上位のモーターが優先的に実行されるため、
上位モーターでは、必要に応じて、下位モーターで定義されているボーンアニメーションを打ち消すようなアクションにしなければなりません。

例えば、全体アクションとデフォルトアクションを考えた場合、キー操作を何もしないときには、デフォルトアクションが実行されています。キー操作により全体アクションが動作するが、全体アクション表現するボーンアニメーション以外のボーンがデフォルトアクションにあると合成されてしまいます。
また、パーツアクションとデフォルトアクションを考えた場合、キー操作によりパーツアクションが動作しますが、パーツアクションで動作しているボーン以外とデフォルトアクションが合成されて問題ないケースもあります。

このように全体アクションでは、デフォルトアクションで使用しているボーン（パーツ）を必ず操作する必要があります。

7.16.2 テクスチャアニメーション

優先順位が下位のモーターより、上位のモーターが優先的に実行されるため、
上位モーターでは、下位モーターのテクスチャアニメーションで使用している部位を扱うようにしなければなりません。

スクリプトリファレンス

本章ではビヘイビアとモータのスクリプト文法について説明します。

8. 構文表記法

本章では、文法を説明するために使用している表記法について説明します。スクリプトの文法は以下の形式で定義します。

8.1 構文のルール

まずその構文の名称があり、それにイコール(=)が続いています。イコールの右辺は、構文のルールを定義しています。小なり(<)と大なり(>)で囲まれた語は特定の構文の要素を示しています。

次の例は、<A>という構文のルールを説明しています。

<A> = <B> | <C>

8.2 見出語

太字で示された語は、見出語を表しています。スクリプト中でもそのままの語が使用されます。次の例では、"play"という見出語はスクリプト中にそのままの語を記述することを示しています。

play { [command...] }

8.3 選択肢

選択肢を構成する垂直バー(|)は、垂直バーで区切られたうちのいずれかの値を取る、という選択肢を表しています。選択肢が一部分だけを構成する場合は、左カッコ(と右カッコ)で囲んでいます。まず最初の例では、<A>は<x>または<y>であることを示しています。また、<B>は「<x> 1 <y>」、「<x> 2 <y>」、または「<x> 3 <y>」のいずれかになります。

<A> = <x> | <y>
<B> = <x> (1 | 2 | 3) <y>

8.4 繰り返し

繰り返しは次のように表現しています。 左カギカッコ[と右カギカッコ]で囲まれた部分は、省略可能であることを示しています。a [a...] という表現は、a を 1 回以上繰り返すことを示しています。ただし a を 2 回以上繰り返す場合は空白で区切ります。例外がある場合には随時補足してあります。

次に上記の例を示します。

<A> = x [y]
<B> = x [x...]

この例では、それぞれ次のようになります。

<A> は「x」または「x y」のいずれか
<B> は「x」、「x x」、「x x x」など

また、以下の文字 1 - 文字 2 という表現は、文字 1 から文字 2 までの範囲に含まれる文字のいずれかを示しています。 例えば次の例は<A>も<B>も a、b、c、d、e、f のいずれかの 1 文字となることを示しています。

<A> = a - f
<B> = (a | b | c | d | e | f)

9. 字句要素

本章では、構文を構成する字句要素について説明します。

文字集合

ビヘイビア・モータスクリプト内では、以下の文字集合を使用することができます。

9.1 アルファベット

大文字と小文字のアルファベット52文字

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

アルファベットの大文字と小文字は区別されます。

9.2 数字

10進数字10個

0 1 2 3 4 5 6 7 8 9

9.3 図形文字

図形文字として以下の文字が使用可能です。

文字	意 味	文字	意 味
!	感嘆符	~	チルダ
#	シャープ	'	一重引用符
$	ドル記号	^	べき乗
&	アンパサント	`	コマンド実行
(	左カッコ	*	アスタリスク
)	右カッコ	¥	バックスラッシュ
-	マイナス	|	垂直バー
+	プラス	;	セミコロン
=	イコール	:	コロン
[	左カギカッコ	″	二重引用符
]	右カギカッコ	,	カンマ
{	左中カッコ	<	小なり
}	右中カッコ	>	大なり
@	アットマーク	/	スラッシュ
.	ピリオド	?	クエスチョン

9.4 特殊文字

以下の文字は Windows の制限により、ファイル名に使用する事はできません。

文字	意 味	文字	意 味
*	アスタリスク	″	二重引用符
¥	バックスラッシュ	,	カンマ
|	垂直バー	<	小なり
;	セミコロン	>	大なり
:	コロン	/	スラッシュ
?	クエスチョン		

9.5 水平タブ

スクリプト中に現れる水平タブは、文字の区切りとして処理され、水平タブは無視されます。

9.6 改行文字

スクリプト中に現れる改行文字は、文字の区切りとして処理され、改行は無視されます。た

だし、以下のケースは別扱いとなります。

コメントでは、改行文字がコメントの終わりを表します。
二重引用符に囲まれた文字列定数内に改行が現れた場合はシンタックスエラーとなります。

9.7 コメント

文字列定数以外の中に感嘆符(!)が現れた場合、それ以降の文字列はコメントとして、スクリプト上は空白と同等に処理されます。コメントには文字数の制限はありません。感嘆符から次の改行文字までがコメントとなります。

〈コメント〉 = ! 〈任意の文字列〉 〈改行文字〉

但し、コメントはキャラ電スタジオに取り込まれませんので、キャラ電スタジオで一度編集を行うと破棄されます。

9.8 トークン

スクリプト中に現れる以下の文字を区切りとしてトークンを切り出します。

文字	意　味	文字	意　味
	空白	{	左中カッコ
¥n	改行	}	右中カッコ
¥r	復帰	:	コロン
¥t	水平タブ	;	セミコロン
!	感嘆符		

9.9 トークンの回避

以下のファイル名を表す場合にのみ、二重引用符(“)で対象文字を囲む事により特殊文字(改行・復帰、:(コロン)、;(セミコロン))を除くトークンを無視させる事ができます。
以下のファイル名以外でのトークンの使用はシンタックスエラーとなります。

(1) 前景・背景ファイル
(2) テクスチャファイル
(3) モデルファイル
(4) アクションファイル
(5) 環境マッピングファイル

9.10 その他

文字列として漢字の使用は可能ですがファイル名において空白文字を含む指定を行うと、上記トークンのルールにより、別の文字列として認識されます。

```
setfg 背景　その1.bmp
```

上記は「setfg」、「背景」、「その1.bmp」と3つの文字列として認識します。ファイル名として「背景　その1.bmp」と認識させる必要がある場合は、setfg ”背景　その1.bmp” の様に二重引用符で囲む必要があります。

10. 構文規則

この章ではビヘイビアとモータについて各セクション毎に説明します。

10.1 ビヘイビア

ビヘイビア(behavior)は以降に説明する各セクション毎の定義によって、イベントの受取り・モータ(motor)の状態の一括変更・目的状態の設定[1]を行ない、キャラ電の振る舞いを決定します。テキストファイルとして作成します(拡張子は.bhvを推奨)。ビヘイビアファイル名に特殊文字を含める事はできません。ビヘイビアファイルの漢字コードはシフトJISで作成します。

以降、ビヘイビアファイル内でのセクション定義について説明します。
ビヘイビアを定義する為の構文は以下の様に定義されます。

```
<behavior-name> {
        [ model { [ <model-file> ] } ]
        [ motor { <motor-file> [ <motor-file>... ] } ]
        [ texture { [ <texture-file>... ] } ]
        [ map { [ <map-file>] } ]
        [ document { [ <document-file>... ] } ]
        [ start { [ <command>... ] } ]
        [ reset { [ <command>... ] } ]
        [ event <event-name> { [ <command>... ] }
        [ event <event-name> { [ <command>... ] }... ] ]
    }
```

✧ <behavior-name>(ビヘイビア名)
ビヘイビア名を定義する文字列です。
✓ ビヘイビアファイルには1つのビヘイビアのみ定義出来可能です。
✓ 特殊文字、トークン共に使用不可です。

✧ model motor texture document start reset event (セクション)
以降に説明する各セクションには定義する順番に指定があります。
以下の順で定義して下さい。
✓ motor セクションは start/reset/event の全てのセクションより前に定義すること。
上記の順番を守ればどの順番で定義しても構いません。

✧ 重複定義
event 以外は重複して定義出来ません。
また、event <event-name> の重複も定義出来ません。

[1] ビヘイビアとモータを使用してキャラ電の状態を遷移させることがスクリプトの役割ですが、スクリプトではキャラ電の状態について「現在状態」と「最終的な状態(目的状態)」を設定する事が可能になっています。「現在状態」を設定するコマンド(set)ではキャラ電の状態をsetコマンドで設定された状態に変更します(指定した動作が即実行されます)。また、「最終的な状態(目的状態)」を設定するコマンド(play)では現在のキャラ電の状態は変更せず、playコマンドによって設定された状態をキャラ電に設定するまで状態を遷移し続けます。(設定された動作までの間の動作を順次実行させる事が可能になります。)

□　ビヘイビアの各セクション毎に説明します。

10.2 model セクション

model セクションでは、キャラ電を作成する際に使用するモデルファイル名を指定します。
3D キャラ電を作成する場合にはこのセクションにモデルファイル名の指定を行います。
2D キャラ電を作成する場合には指定しません。

model セクションは以下の様に定義されます。

```
model { <model-file> }
```

✧ model セクション
- ✓ model セクションの複数定義は出来ません。

✧ <model-file>（モデルファイル名）
モデルファイル名（.bac）を指定します。
- ✓ 存在するモデルファイルである必要があります。
- ✓ 複数のモデルファイルは定義出来ません。
- ✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
- ✓ モデルファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
- ✓ model セクションの定義を行った場合はモデルファイルの指定は必須となります。

例）モデルファイルとして「figure.bac」を使用する場合。

```
model { figure.bac }
```

例）モデルファイルとして「figure 01.bac」を使用する場合。

```
model { “figure 01.bac” }
```

ファイル名に空白を含める場合は、二重引用符で囲む必要があります

10.3 motor セクション

motor セクションでは、使用するモータとその順番をモータファイル名を指定する事で定義します。以降に説明する start/reset/event セクションで定義するモータはこのセクションで定義しておく必要があります。

motor セクションは以下の様に定義されます。

```
motor { <motor-file> [ <motor-file>…]  }
```

✧ motor セクション
- ✓ motor セクションの複数定義は出来ません。

✧ <motor-file>（モータファイル名）
モータファイル名（.mot）を指定します。
- ✓ 存在するモータファイル名である必要があります。
- ✓ 複数のモータファイル名を定義する場合は空白で区切ります。
- ✓ 同一モータファイル名の重複定義は出来ません。
- ✓ 定義出来るモータファイル数は最大16個です。
- ✓ モータファイル名に特殊文字、トークンは使用出来ません。
- ✓ モータファイル名の定義の順番により、アクションの優先度[2]が決定されます。
- ✓ start/reset/event セクションよりも前に定義する必要があります。

例）モータファイル名として「motor1.mot」と「motor2.mot」を使用する場合。

```
motor { motor1.mot motor2.mot }
```

[2] アクションの合成時に同一ボーンへのアクションが発生した場合には、当セクションにて指定されたモータファイルの左から順に優先度が高く再生されます。

10.4 texture セクション

texture セクションでは、モデルに適用するテクスチャファイル名（.gif、.bmp）を指定します。
使用するモデルに合わせたテクスチャファイル名を指定します。
2Dキャラ電を作成する場合には指定しません。

texture セクションは以下の様に定義されます。

```
texture { <texture-file> [ <texture-file>…]  }
```

✧ texture セクション
- ✓ texture セクションの複数定義は出来ません。
- ✓ texture セクションを定義した場合は model セクションの定義が必須となります。

✧ <texture-file>（テクスチャファイル名）
- ✓ 存在するテクスチャファイル名である必要があります。
- ✓ 複数のテクスチャファイル名を定義する場合は空白で区切ります。
- ✓ 定義出来るテクスチャファイル数は最大16個です。
- ✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
- ✓ テクスチャファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
- ✓ モデルに対するテクスチャの順番を意識して[3]設定する必要があります。

例）テクスチャファイルとして「texture1.gif」と「texture2.gif」を使用する場合。

```
texure { texture1.gif texture2.gif }
```

例）テクスチャファイルとして「texture 1.gif」と「texture 2.gif」を使用する場合。

```
texture { “texture 1.gif” “texture 2.gif” }
```

ファイル名に空白を含める場合は、二重引用符で囲む必要があります

[3] 作成したモデルに設定しているテクスチャIDの順に定義する必要があります。

10.5 map セクション

map セクションでは、モデルに適用する環境マッピングファイル名（.gif、.bmp）を指定します。
使用するモデルに合わせた環境マッピングファイル名を指定します。
2Dキャラ電を作成する場合には指定しません。

map セクションは以下の様に定義されます。

```
map { <map-file>}
```

✧ map セクション
- ✓ map セクションの複数定義は出来ません。
- ✓ map セクションを定義した場合は model セクションの定義が必須となります。

✧ < map -file>（環境マッピングファイル名）
- ✓ 存在する環境マッピングファイル名である必要があります。
- ✓ 定義出来る環境マッピングファイル数は最大1個です。
- ✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
- ✓ 環境マッピングファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。

例）環境マッピングファイルとして「map 1.gif」と「map 2.gif」を使用する場合。

```
map { map 1.gif map 2.gif }
```

例）環境マッピングファイルとして「map 1.gif」と「map 2.gif」を使用する場合。

```
map { “map 1.gif” “map 2.gif” }
```

ファイル名に空白を含める場合は、二重引用符で囲む必要があります

10.6 document セクション

document セクションでは、キーアサインヘルプ情報を設定します。
感情アクション、パーツアクション毎のキーアサインヘルプファイル名を指定します。

document セクションは以下の様に定義されます。

```
document { <help-file> [ <help-file>…]  }
```

✧ document セクション
- ✓ document セクションの複数定義は出来ません。

✧ <help-file>（キーアサインヘルプファイル名）
キーアサインヘルプファイル名(.txt)を指定します。
- ✓ 存在するキーアサインヘルプファイル名である必要があります。
- ✓ 複数のキーアサインヘルプファイル名を定義する場合は空白で区切ります。
- ✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
- ✓ キーアサインヘルプファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
- ✓ キーアサインヘルプファイル名の順番は以下の様に定義します。
  document { help-file0 help-file1 }
  help-file0 ・・・「全体アクション」のキーアサインヘルプ情報
  help-file1 ・・・「パーツアクション」のキーアサインヘルプ情報
- ✓ 3つ目以降のキーアサインヘルプファイルについては処理対象外となります。

例）キーアサインヘルプファイルとして「help0.txt」と「help1.txt」を使用する場合。

```
document { help0.txt help1.txt }
```

例）キーアサインヘルプファイルとして「help 0.txt」と「help 1.txt」を使用する場合。

```
document { “help 0.txt” “help 1.txt” }
```

ファイル名に空白を含める場合は、二重引用符で囲む必要があります

✧ キーアサインヘルプファイルのフォーマット

キーアサインヘルプファイルの内容は以下の様に設定します。
一行目はキーアサインヘルプのレコード数を設定します。
キーアサインヘルプファイルの漢字コードはシフトJISで作成します。
“アサインキー（最大 7 バイト）”, 0（表示不可）／1（可）, “アクション名（最大30文字）” [改行]

例） キーアサインヘルプファイル（help0.txt）の内容は、以下の様に定義します。

```
3 [改行]          ←行頭はレコード数を設定
“1”, 1, “笑う“[改行]
“2”, 1, “泣く” [改行]
“3”, 1, “ジャンプ” [改行]
```

- アサインキー
  - ✓ 1～7 バイト以内で設定する必要があります。
  - ✓ アサインキーを”（二重引用符）で囲む必要があります。
  - ✓ 重複定義可能です。
  - ✓ 必須入力です。
  - ✓ アサインキーに”（二重引用符）を定義する場合は””（二重引用符）を二つ記入します。

- 表示可／不可
  - ✓ 0（表示可）または1（表示不可）のいずれかを設定する必要があります。

- アクション名
  - ✓ 1～30文字以内で設定する必要があります。
  - ✓ アクション名を”（二重引用符）で囲む必要があります。
  - ✓ 必須入力です。

10.7 start セクション

start セクションでは、背景・前景および、状態を設定する各コマンドを定義することにより、キャラ電の初期動作[4]または状態を設定します。

start セクションは以下の様に定義されます。

```
start {  [
            ( event ( on | off )                                    |
            setfg <foreground-file> [attribute]  |
            setbg <background-file> [attribute] |
            modeloffset <X> <Y>                                     |
            modelscale <scale>                                      |
            light <light-id> <X> <Y> <Z> <light-intensity> <ambient-intensity> |
            play  { <motor-file>:<state> [ <motor-file>:<state>…]  }  |
            set   { <motor-file>:<state> [ <motor-file>:<state>…]  } );…
         ]
}
```

- ✧ start セクション
  - ✓ start セクションの複数定義は出来ません。
  - ✓ motor セクションより後に定義する必要があります。
  - ✓ 以降に説明する各コマンド（event/setfg/setbg/modeloffset/modelscale/light/play/set）の間は;（セミコロン）で区切ります。
  - ✓ 各コマンド（event/setfg/setbg/modeloffset/modelscale/light/play/set）は定義された順番に動作します。

- ✧ コマンドに関しては、コマンド詳細（ start / reset / event ）を参照下さい。

4 ロードされたキャラ電がイベントを受付ける状態になる前に行なう初期の動作または状態

10.8 reset セクション

reset セクションでは、背景・前景および、状態を設定する各コマンドを定義することにより、キャラ電がリセット[5]された時の動作や状態を設定します。

reset セクションは以下の様に定義されます。

```
reset {  [
            ( event ( on | off )                                    |
            setfg <foreground-file> [attribute]  |
            setbg <background-file> [attribute] |
            modeloffset <X> <Y>                                     |
            modelscale <scale>                                      |
            light <light-id> <X> <Y> <Z> <light-intensity> <ambient-intensity> |
            play  { <motor-file>:<state> [ <motor-file>:<state>…]  }  |
            set   { <motor-file>:<state> [ <motor-file>:<state>…]  } );…
         ]
}
```

✧ reset セクション
- ✓ reset セクションの複数定義は出来ません。
- ✓ motor セクションより後に定義する必要があります。
- ✓ 以降に説明する各コマンド（event/setfg/setbg/modeloffset/modelscale/light/play/set）の間は;（セミコロン）で区切ります。
- ✓ 各コマンド（event/setfg/setbg/modeloffset/modelscale/light/play/set）は定義された順番に動作します。

✧ コマンドに関しては、コマンド詳細（ start / reset / event ）を参照下さい。

5 キャラ電がリセットイベントを受付けた場合の動作または状態

10.9 event セクション

event セクションでは、キャラ電が受け取るイベントを設定し、イベント毎の動作や状態の設定を行います。

event セクションは以下の様に定義されます。

```
event <event-name> {
        [
            ( event ( on | off ) |
            setfg <foreground-file> [attribute]  |
            setbg <background-file> [attribute] |
            modeloffset <X> <Y>                          |
            modelscale <scale>                           |
            light <light-id> <X> <Y> <Z> <light-intensity> <ambient-intensity> |
            play  { <motor-file>:<state> [ <motor-file>:<state>…]  }  |
            set   { <motor-file>:<state> [ <motor-file>:<state>…]  } );…
        ]
}
```

✧ event セクション
- ✓ <event-name>は受付けるイベントを設定します。(必須)
- ✓ <event-name>で設定可能な文字列の長さは最大7バイト(#12345#)です。
- ✓ event セクションは同一イベントでの複数定義は出来ません。
  例)
  event 1 { command. . . }
  event 1 { command. . . } ・・・ ×
  同一イベント(1)に対する複数の定義は不可
- ✓ motor セクションより後に定義する必要があります。
- ✓ 以降に説明する各コマンド
  (event/setfg/setbg/modeloffset/modelscale/light/play/set)の間は;(セミコロン)で区切ります。
- ✓ 各コマンド(event/setfg/setbg/modeloffset/modelscale/light/play/set)は定義された順番に動作します。

✧ コマンドに関しては、コマンド詳細(start / reset / event)を参照下さい。

10.10 コマンド詳細(start / reset / event)

startセクション、resetセクション、eventセクションにおいて以下に説明する各コマンドを指定する事でキャラ電の動作を制御します。

✧ event（イベント受付コマンド）
on または off を設定する事により、実行中に他のイベントを受付けるか否かを設定します。
✓ on または off のいずれかを指定します。(デフォルトは on)

✧ setfg（前景設定コマンド）／setbg（背景設定コマンド）
setfg ＜foreground-file＞ [attribute]により前景画像を指定する場合に使用します。
setbg ＜background-file＞ [attribute]により背景画像を指定する場合に使用します。
✓ 存在するファイルである必要があります。
✓ 指定出来る画像ファイルは GIF89a または Bitmap 形式である必要があります。
✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
✓ 前景／背景ファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
✓ setfg コマンドを定義した場合は前景画像の設定は必須です。
✓ [attribute]には属性として以下の内容が設定可能です。
ー(ハイフン) クリア
tile タイリング
zoom ストレッチ
center センタリング
何も指定しない場合は「属性なし」となります。
✓ 設定されている画像について同じ属性が指定されていなければなりません。

✧ modeloffset（モデル位置設定コマンド）
modeloffset ＜X＞ ＜Y＞ によりモデルの位置を指定する場合に使用します。
ピクセル単位で指定します。
✓ ＜X＞ ＜Y＞共に-200～200の間で指定する必要があります。

✧ modelscale（モデルスケール設定コマンド）
modeloffset ＜scale＞ によりモデルのスケール(%)を指定する場合に使用します。
✓ ＜scale＞は0～1000の間で指定する必要があります。

✧ light（光源情報設定コマンド）
light ＜light-id＞ ＜X＞ ＜Y＞ ＜Z＞ ＜light-intensity＞ ＜ambient-intensity＞ により光源情報を設定する場合に使用します。
ピクセル単位で指定します。
✓ ＜light-id＞は無効または有効のいずれかを設定します。
✓ ＜X＞ ＜Y＞ ＜Z＞は－32768～32767の間で指定する必要があります。
✓ ＜light-intensity＞ ＜ambient-intensity＞は0～4096の間で指定する必要があります。
✓ light コマンドは、各イベントの最後にコマンドを記述する必要があります。
✓ ＜light-id＞が無効の場合、＜X＞ ＜Y＞ ＜Z＞ ＜light-intensity＞ ＜ambient-intensity＞を省略します。

✧ play（目的状態設定コマンド）
play ＜motor-file:state＞ により目的状態の設定を行う場合に使用します。
✓ 指定する motor-file は事前に motor セクションにて定義されていなければなりません。
✓ 指定する state は motor-file にて定義されている state でなければなりません。
✓ motor-file と state の間は:(コロン)で区切ります。

- ✓ 複数指定を行う場合は各状態を空白で区切ります。
  例） play { motorA:stateA motorB:stateB motorC:stateC }
- ✓ 同一モータに対する複数指定はできません。
  例） play { motorA:stateA motorA:stateB } ･･･ ×
  同一モータ(motorA)に対する複数指定なので不可
- ✓ motor-fileおよびstateは特殊文字及びトークンは使用出来ません。

✧ set（現在状態設定コマンド）

set ＜motor-file:state＞により現在の状態の設定を行う場合に使用します。

- ✓ 指定する motor-file は事前に motor セクションにて定義されていなければなりません。
- ✓ 指定する state は motor-file にて定義されている state でなければなりません。
- ✓ motor-file と state の間は:(コロン)で区切ります。
- ✓ 複数指定を行う場合は各状態を空白で区切ります。
  例） set { motorA:stateA motorB:stateB motorC:stateC }
- ✓ 同一モータに対する複数指定はできません。
  例） set { **motorA**:stateA **motorA**:stateB } ･･･ ×
  同一モータ(motorA)に対する複数指定なので不可
- ✓ motor-fileおよびstateは特殊文字及びトークンは使用出来ません。

10.11 ビヘイビア使用例

ビヘイビアの使用例を以下に示します。

```
! Sample avator behavior
behavior {
    model     { sample.bac }                                    … ①
    texture   { texture1.bmp texture2.bmp }                     … ②
    map       { map1.bmp }                                      … ③
    motor     { feeling.mot head.mot body1.mot }                … ④
    document  { help0.txt help1.txt  }                          … ⑤
    start {                                                     … ⑥
            setbg standard.bmp;
            set { feeling.mot:wait head.mot:wait body1.mot:wait }
    }
    reset {                                                     … ⑦
            setbg reset.bmp;
            set { feeling.mot:wait head.mot:wait body1.mot:wait };
            light 0
    }
    event 1 {                                                   … ⑧
            setbg glad.bmp;
            set { feeling.mot:wait head.mot:1_head body1.mot:1_body }
            light 1 100 0 0 4096 2048
    }
    event 11 {                                                  … ⑨
            setbg cry.bmp;
            play { head.mot:2_head }
    }
}
```

① モデルファイルとして sample.bac を使用します。
② テクスチャファイルとして texture1.bmp、texture2.bmp を使用します。
③ 環境マッピングファイルとして map1.bmp を使用します。
④ モータファイルとして feeling.mot、head.mot、body1.mot を使用します。
⑤ キーアサインヘルプファイルとして help0.txt(全体)、help1.txt(パーツ)を使用します。
⑥ 初期動作として以下の様な動作を設定します。
1. 背景画像に standard.bmp を設定する。
2. 各パーツの状態を以下の様に設定する。
   - feeling.mot モータの wait ステート
   - head.mot モータの wait ステート
   - body1.mot モータの wait ステート

⑦ リセット動作として以下の様な動作を設定します。
1. 背景画像に reset.bmp を設定する。
2. 各パーツの状態を以下の様に設定する。
   - feeling.mot モータの wait ステート
   - head.mot モータの wait ステート
   - body 1.mot モータの wait ステート

3. 光源情報を 0(無効)に設定する

⑧ キー 1 が押下された場合の動作として以下の様な動作を設定します。
1. 背景画像に glad.bmp を設定する。
2. 各パーツの状態を以下の様に設定する。
   - feeling.mot モータの wait ステート
   - head.mot モータの 1_head ステート

- body 1.mot モータの 1_body ステート

3. 光源情報を以下の様に設定する。
   - <light-id> 1
   - <X> <Y> <Z> 100 0 0(左からの光源)
   - <light-intensity> 4096
   - <ambient-intensity> 2048

⑨ キー 11 が押下された場合の動作として以下の様な動作を設定します。

1. 背景画像に cry.bmp を設定する。
2. 各パーツの状態を以下の様に設定する。
   - head.mot モータの 2_head ステート

10.12 モータ

モータ(motor)は以下に説明するステートの定義によって、キャラ電の状態の遷移を行います。
テキストファイル(拡張子は.mot を推奨)として作成し、モータ毎に別のファイルとして作成します。
モータファイル名は特殊文字及びトークンを含まない名称を使用します。
モータファイルの漢字コードはシフトJISで作成します。

以降、モータファイル内での定義について説明します。
モータを定義する為の構文は以下の様に定義されます。

```
<state-name> : ( - | <action-file> ) {
        [ [ start { <command> [ <command>… ] } ]                    |
          [ end { <command> [ <command>… ] } ]                       |
          [ time <msec> { <command> [ <command>… ] } ] |
          [ on <handler> { <command> [ <command>… ] } ]; ]
}
```

✧ <state-name>(ステート名)
ステート名を定義する文字列です。
- ✓ 特殊文字、トークンを含まない文字列で16バイト以内で指定します。
- ✓ 複数のステート名を定義する場合は先頭に定義したステート名がデフォルトのステート[6]となります。
- ✓ 定義可能なステート数の最大は128です。

✧ <action-file> (アクションファイル名)
ステート名に続き、:(コロン)の次に実行するアクションファイルを指定します。
- ✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
- ✓ アクションファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
- ✓ 複数のアクションファイルは定義出来ません。
- ✓ アクションファイルを指定しない場合は-(ハイフン)を記入します。
- ✓ アクションファイル名はファイル名か-(ハイフン)が必須です。
- ✓ アクションファイルとして-(ハイフン)という名称のファイルは指定できません。
- ✓ アクションファイル名は1ステートに1つ定義可能です。

✧ start / end / time / on (イベント)

ステート内で処理するイベントを設定します。以下のイベントが定義可能です。
尚、イベントは1ステート内で重複して定義する事はできません。

➢ **start**
このイベントを定義すると、定義されているステートにのアクションの前にこのイベント内のコマンドが実行されます(アクションが設定されていなくても実行されます)。

➢ **end**
このイベントを定義すると、定義されているステートのアクションの後に
このイベント内のコマンドが実行されます。尚、ステートにアクションが

6 ビヘイビアのstartセクションにて初期のステートが設定されていない場合

設定されていない場合にはこのイベントを定義してあっても実行されません。

- **time** 〈msec〉
  このイベントを定義すると、定義されたステートに遷移してから、指定された秒数（msec）後にこのイベント内のコマンドが実行されます。

- **on** 〈state〉 または 〈*〉
  このイベントを定義すると、定義されたステートに遷移してから指定されたステート（目的状態の設定）があればこのイベント内のコマンドが実行されます。
  ステートを定義する場合はモータ内に存在するステートでなければなりません。
  ＊（アスタリスク）を指定した場合はどのステートが目的状態として設定されてもイベント内のコマンドが実行されます。

コマンドの詳細については次ページを参照して下さい。

〈command〉（コマンド）

各イベントに対して実行するコマンドを記入します。
実行するコマンドには以下のコマンドが指定可能です。

✓ 複数のコマンドを指定する場合は行末に;（セミコロン）を設定する必要があります。

➢ setfg（前景設定コマンド）／setbg（背景設定コマンド）

setfg < foreground-file> [attribute]により前景画像を指定する場合に使用します。
setbg <background-file> [attribute]により背景画像を指定する場合に使用します。

✓ 存在するファイルである必要があります。
✓ 指定出来る画像ファイルは GIF89a または Bitmap 形式である必要があります。
✓ 漢字ファイル名は使用可能ですが、ファイル名に特殊文字は使用出来ません。
✓ 前景／背景ファイル名に特殊文字を除くトークンを含める場合は二重引用符で囲む必要があります。
✓ setfg コマンドに対し、前景画像の設定は必須です。
✓ [attribute]には属性として以下の内容が設定可能です。

－（ハイフン）	クリア
tile	タイリング
zoom	ストレッチ
center	センタリング

何も指定しない場合は「なし」となります。

✓ 設定されている画像について同じ属性が指定されていなければなりません。

➢ next（遷移先設定コマンド）

next 〈state〉により遷移する状態を事前に設定しておく事が可能です。
このコマンドで遷移先を指定した場合、内部の現在状態は変更されません。
当コマンドにより指定されたステートへの遷移は gotonext コマンドにより行われます。
gotonext コマンドが設定されていない状態でコマンドが終了した場合には暗黙で指定されたステートに遷移します。

✓ 同一モータ内に存在するステートを指定する必要があります。

➢ goto（遷移コマンド）

goto 〈state〉により状態を遷移する先のステートを指定します。内部の動作としては現在状態を指定された遷移先に変更します。

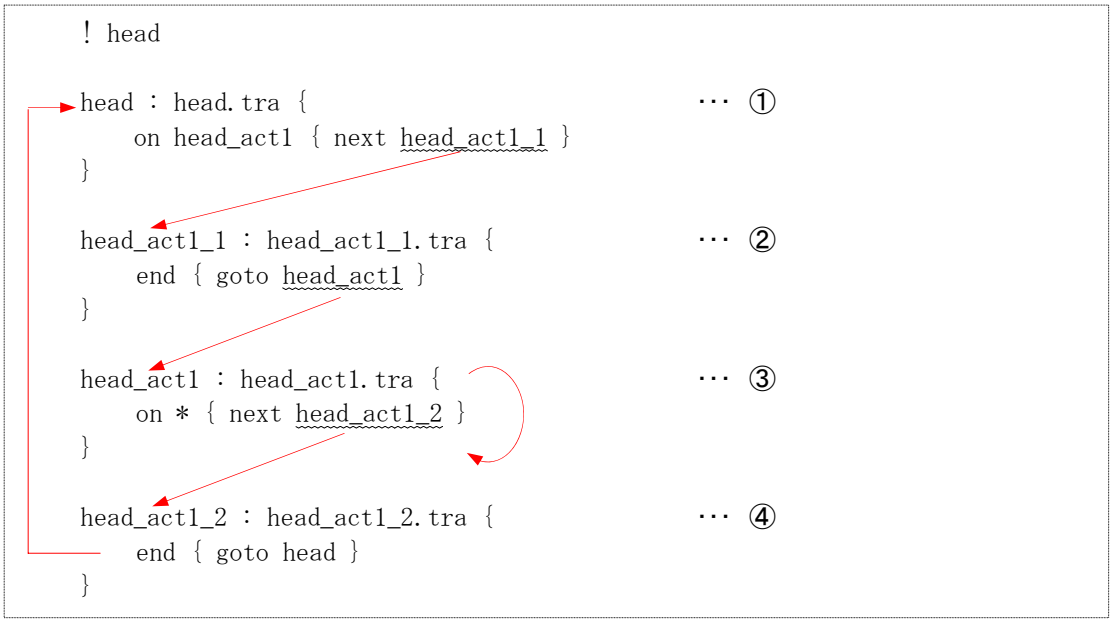
✓ 同一モータ内に存在するステートを指定する必要があります。

➢ gotonext（遷移コマンド）

gotonext により next コマンドで指定されたステートへの遷移が行なわれます。

10.13 モータ使用例

モータの使用例を以下に示します。



① head ステート
head モータ内ではビヘイビアから特に指定されていない限り、デフォルトのステートとなります。当ステートに遷移後、head.tra のアクションが実行されます。目的状態として head_act1 の設定が行われていない場合には上記アクションが繰り返されます。目的状態が on head_act1 に設定されると、 head_act1 イベント指定によりコマンド next head_act1_1 が実行され、次に実行すべきステートが設定されます。コマンドが終了するので自動的に end が発行され現在状態のステートは head_act1_1 に遷移します。

② head_act1_1 ステート
当ステートに遷移後、head_act1_1.tra のアクションが実行されます。実行後、goto head_act1 により、head_act1 ステートに遷移します。

③ head_act1 ステート
当ステートに遷移直後に①のイベント指定で指定されていた目的状態と現在の状態が一致したので目的状態への遷移は終了します。その後、head_act1.tra が実行される。ステート内に on * イベント指定があるので、目的状態の変更があると、次のコマンドが実行されます。next head_act2_1 により次に実行すべきステートが設定されます。ここでのポイントは next コマンドにより現在状態を変更することなく状態を遷移させる事です。その後コマンドが終了して自動的に end が発行され、現在状態のステートは head_act1_2 に遷移します。

④ head_act1_2 ステート
当ステートに遷移後、head_act1_2.tra が実行されます。実行後、end イベントにより goto head コマンドが実行され、head ステートへ遷移します。head ステートへ遷移後は特に目的状態が設定されいないので head.tra のアクションが繰り返されます。

上記①～④の流れの様に、モータ内ではステートの目標状態を設定する事により目標状態が現在のステートと同じになるまで状態を遷移して行きます。(アクションを連続して再生する事が出来ます。)目標状態の設定はビヘイビアの play コマンドにて制御する事になります。