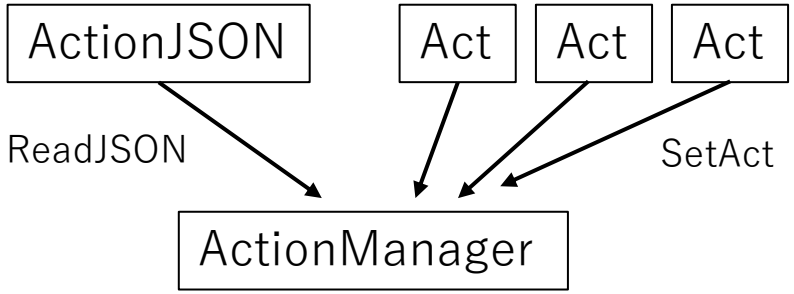
ActionManagerクラス

ActionManagerクラス

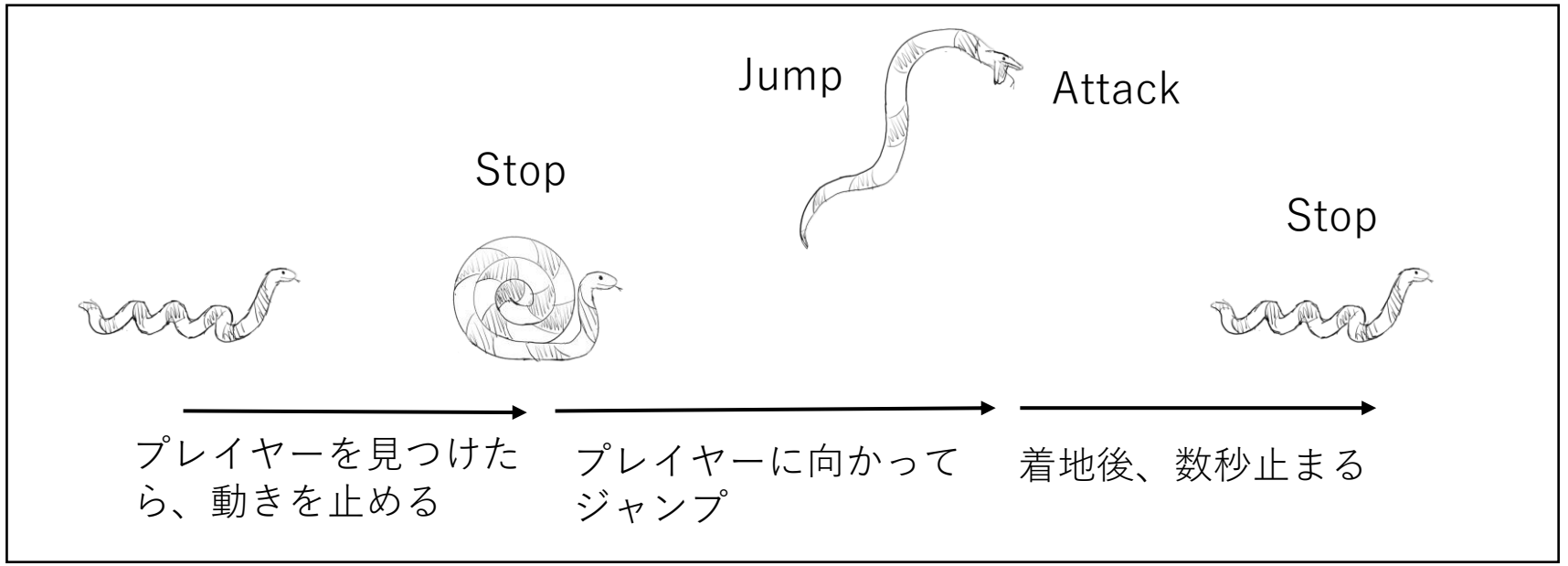
メンバ関数

- **ReadJSON**　ActionJSON(アクションの流れを記述したJSONファイル)を読み込む
- **SetAct**　Actをセット
- **SetAll**　Actのセットが完了したら呼び出す(今後これをしなくてもよくなるかも)
- **update**　セットされたActを実行
- **draw**　Actのdrawを呼び出す(攻撃判定など)
- **drawTexture**　TextureDrawクラスの描画
- **Start**　指定したアクションを開始
- **End**　指定したアクションを強制終了
- **StartAct**　実行中のアクションのActを開始させる
- **EndAct**　実行中のアクションのActを終了させる
- **Active**　指定したアクションが実行中かどうかをboolで返す



Actについては次のページで解説
ActionJSONについては後程解説

Actについて



Actはアクションの構成要素です。
例えば上の図では、Stop,Jump,Attack,StopがActクラスを基底とするクラスになります。

ちなみにAttackというクラスに開始硬直時間と終了硬直時間を指定できる機能、さらにジャンプするという機能もつければ、このアクションは一つのクラスで表現することもできます。

どんなActを使うかによって、アクションの形も変わります。

Actを作る

派生クラスで実装する主なメンバ関数は以下の4つです

・**startAct**	Act開始時に行う処理を記述(例えば物体に初速度を与えたり)
・**endAct**	Act終了時に行う処理を記述(例えば物体を静止させたり)
・**update**	Act実行中に行う処理を記述
・**draw**	何か描きたいものがあれば(Attakcクラスでは攻撃判定の描写を行っている)

派生クラスを作るとき、4つすべてを実装する必要はありません。必要なメンバ関数だけ実装しましょう。

例.Jumpクラス

```
class Jump :public Act
{
public:
    Jump(Character* chara) :chara(chara) {};
    ~Jump() {};
    //ジャンプ時の重力加速度とジャンプの高さ
    void SetJump(double gravity, double height);
    void startAct();
    void update();
    void endAct();
private:
    Character* chara;
    double gravity;
    double speed;//初速度
};
```

←
startActでキャラクターの加速度の変更や初速度の加算、updateでキャラクターの速度に重力加速度を毎回加算、endActでキャラクターの加速度をステージ上の重力加速度に戻す、というようなことをやっています。
Move.h/Move.cppに定義してあるので、気になったら読んでみてください。

また、ジャンプ中の重力加速度とジャンプの高さをSetJumpというこのクラス固有の関数で最初に設定します。

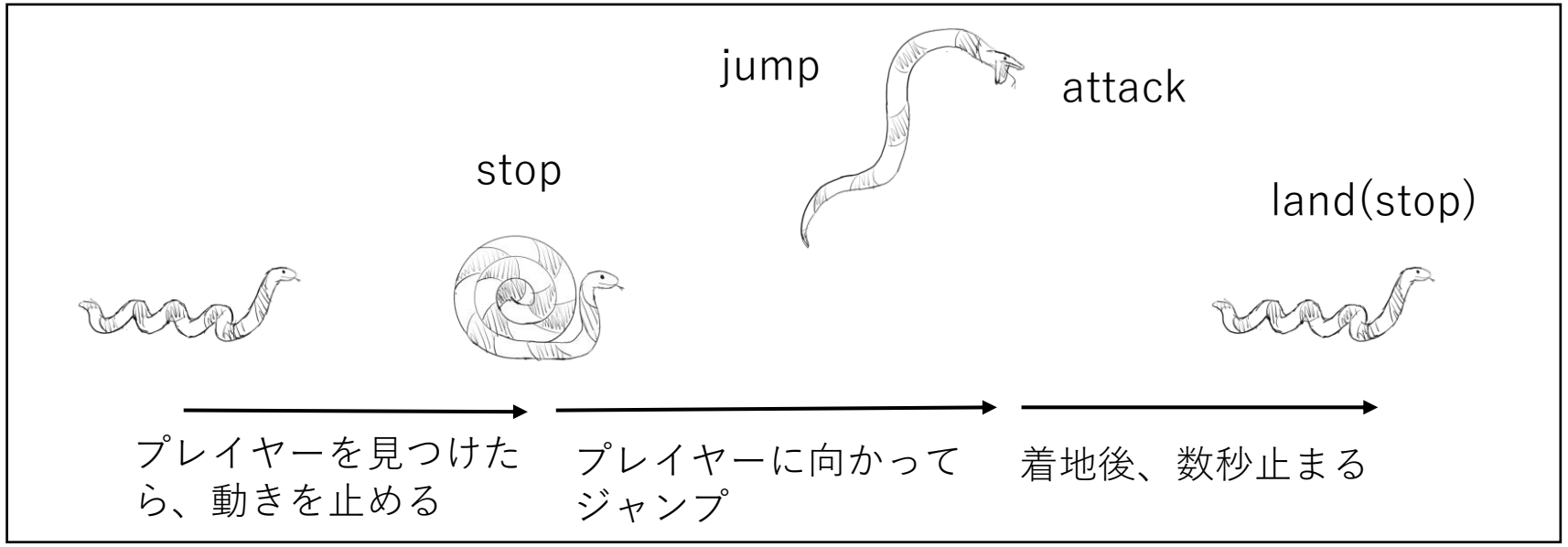
現在用意されてるActが基底のクラス

- **Move** 等速直線/等加速度運動、いろいろな動きを作れる。ごちゃごちゃしてて使いづらい。
- **UniformLinearMotion** 等速直線運動。壁にぶつかると自動で終了する。
- **Jump** 最初に重力加速度(px/s^2)とジャンプの高さ(px)を設定。y軸方向の速度が0になるとき(落下の始まりや、天井との衝突)自動で終了する。
- **Fall** 最初に重力加速度と落下の初速度(px/s)を設定。床に着くと勝手に終了。
- **Slip** キャラクターの向き(direction)とは逆向きに加速度が加わる。その方向の速度が0になると勝手に終了。
- **Stop** Act開始時に速度を0にする。
- **Attack** 最初に当たり判定などを渡し、実行中はその当たり判定を出現させる。
- **TextureDraw** アクション中のTextureを描く。開発時にアクションをわかりやすくするため。

おまけ

- **PrintName** 実行中ならセットした文字列を表示する
- **Switch** 何もしない。ActionManagerがActの開始と終了でこのActクラスを使ってる。

Actの開始条件、続行条件



上の図のアクションはstop,jump,attack,landという四つのActで構成され、それぞれには以下のような開始条件と続行条件があります。

- stop

開始条件: アクション開始から0秒後　　続行条件: 1.5秒間

- jump

開始条件: Stop1終了から0秒後　　続行条件: 着地するまで

- attack

開始条件: Stop1終了から0秒後　　続行条件: Moveが終了するまで

- land

開始条件: Moveが終了から0秒後　　続行条件: 1秒間

Actの開始条件、続行条件について

開始条件:
①アクション開始からt秒後に開始
②他Act開始からt秒後に開始
③他Act終了からt秒後に開始

続行条件:
①t秒間続ける
②他Act開始からt秒後まで続ける
③他Act終了からt秒後まで続ける

上記の条件はJSONファイルで設定できます。

ただし、前ページに書いたjumpの続行条件「着地するまで」 は①から③のどれにも当てはまらないので、JSONファイルで設定することはできません。
①から③に当てはまらない条件はプログラム上で書きます。↓

```
if (box.touch(down_line)) { actions->EndAct(U"Attack", U"jump"); }
```

「もし地面に触れたら”Attack”というアクションの”jump”を終了させる」という指示。このようにして続行条件を作ることができます。また、StartActもあるので開始条件も同様にプログラム上で条件を作ることができます。

このような特殊条件で構成されていてもActionJSON(アクションの流れを記述したJSONファイル)は必須。次のページではActionJSONについて説明する。

ActionJSON

ActionJSONの中身は大体こんな感じになっている

```
{
  "Actions": [
    {
      "Acts": [
        {
          "Continue": {
            "Conection": ,
            "length":
          },
          "Start": {
            "Conection": ,
            "start_time":
          },
          "name":
        }
      ],
      "name":
    }
  ]
  "Pos":[]
}
```

“Actions”は複数のActs(Action)を入れるための配列

“Acts(Action)”は複数のActを入れるための配列

“Continue”は続行条件を入れるオブジェクト

“Start”は開始条件を入れるオブジェクト

“name”にはActの名前を入れる。名前はActs(Action)内で被りがなければなんでもOK

“name”にはActs(Action)の名前を入れる。名前はActions内で被りがなければなんでもOK

“Pos”にはActionGUIでこのファイルを開くときに必要な情報が入ってます。いじっちゃだめ！

このJSONファイルは手動で作成するわけではなく、ActionGUIというツールが作成してくれます。
ActionGUIに関しては別資料「ActionGUIの使い方」で説明します。

ActionJSONの編集の仕方

ActionJSONにはActの開始条件と続行条件が書いてあります。

開始条件は“Start”:{}の中に、続行条件は”Continue”:{}の中です。どっちのオブジェクトも中の構造はほとんど変わりません。

```
“Start”:{
  “start_time”: a //アクション開始からa秒後に開始する。-1ならこの条件は使わない。
  “Conection”: [
    {
      “name”: “fall”
      “from”: “start”もしくは”end”
      “time”: b
      //anotherActが”start”もしくは”end”してからb秒後に開始。
    }
  ]
}
```

アクションの調整で主にいじるのはstart_timeやConectionのtimeだと思います。

慣れないうちはJSONファイルを直接いじるのではなく、ActionGUIで開いて入力しなおすのが無難かもしれません(使い勝手は悪いですけど)。