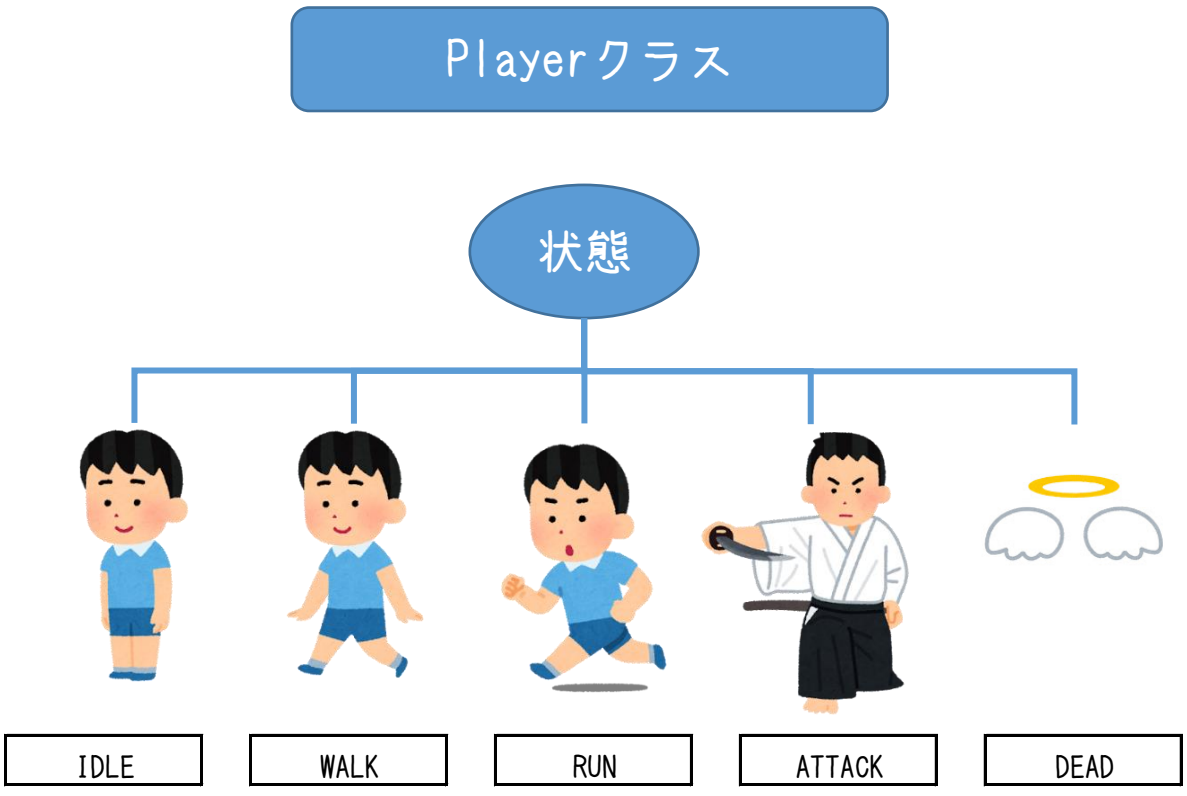
Stateパターン

ゲームをプログラミングして、実行する際に、
今、何がどうなっているか、わからなくなることは無かったでしょうか？

例えば、アクションゲームの敵キャラであれば、
今、プレイヤーを探しているのか、追跡しているのか、攻撃しているのか、
不具合でウロウロしているだけなのか。。。
『状態』がよくわからなくなることがあります。

また、クラスのに『状態』という要素が加わると、
プログラムも煩雑になりやすく、デバッグトレースでも
追いにくくなり易い傾向があります。

そこで、今回は、簡単なStateパターンという、状態がわかりやすくなるような、
プログラムの書き方を実践していきたいと思います。



まずは状態をenumで定義。

```
enum class STATE
{
    NONE,
    IDLE,
    WARK,
    RUN,
    ATTACK,
    DEAD
};
STATE mState;
```

UpdateやDrawなど、フレームごとの処理を状態で分ける。

```
switch (mState)
{
case Player::STATE::NONE:
    // 何もしない
    break;
case Player::STATE::IDLE:
    // 待機状態
    break;
case Player::STATE::WARK:
    // 歩き状態
    break;
case Player::STATE::RUN:
    // 走り状態
    break;
case Player::STATE::ATTACK:
    // 攻撃状態
    break;
case Player::STATE::DEAD:
    // 死亡状態
    break;
default:
    break;
}
```

状態を変更する時は、必ず専用の関数を通す。

```
void Player::ChangeState(STATE state)
{
    // 状態を更新
    mState = state;
    // 状態ごとの初期処理を行う
    switch (mState)
    {
    case Player::STATE::NONE:
        break;
    case Player::STATE::IDLE:
        break;
    case Player::STATE::WARK:
        break;
    case Player::STATE::RUN:
        break;
    case Player::STATE::ATTACK:
        break;
    case Player::STATE::DEAD:
        break;
    default:
        break;
    }
}
```

※ mState = STATE::IDLEとかは、基本記述しない。

このデザインパターーンを使うだけで、かなり状態制御が管理しやすくなります。

- ゲーム実行中でも、ChangeState内に変更Stateをログに出力するだけで、状態変化がわかるようになる

- 他人が見た時も、どこで何をしているのか一目でわかるようになる！
- パターーンでプログラムが書けるようになる