重力

重力値やジャンプ力の値設定に関して、少し掘り下げなら実装していきます。

現実世界の重力加速度 9.8 m/s　を、そのまま使うと、
重力が弱すぎてフワフワしてしまいます。

これには、理由があり、
DxLibの座標 1 は、1センチメートル相当だと言われています。

※ Unityの座標 1 は、1メートルと定義されており、
　Unityの座標系をそのままDxLibに適用しても、座標を100倍するか、
　スケールを100分の1にしないと、位置関係が合わないため

仮に1センチメートルだとすると、9.8 cm/s になりますので、
1秒に9.8cmしか移動しないということになり、とても弱い重力になります。

といった理由から、一般的な数学式を使用したいのであれば、

```
// 重力
static constexpr float GRAVITY = 9.81f * 100.0f;
```

重力は、上記のように定義すると計算が合いやすくなると思います。

それでも、ゲームっぽい挙動を行うためには、多少の都合の良い調整値が
必要になりますので、下記のように実装すると調整しやすいです。

Application.h

```
public:

    ～ 省略 ～

    // 重力
    static constexpr float GRAVITY = 9.81f * 100.0f;
    static constexpr float GRAVITY_SCALE = 0.7f;

    ～ 省略 ～

    // 重力の取得
    float GetGravityPow(void) const { return GRAVITY * GRAVITY_SCALE; }
```

重力処理は、各キャラクターに共通してかかりますので、
Playerではなく、CharactorBaseに実装していきます。

CharactorBase.h

```
class CharactorBase : public ActorBase
{

protected:

    ～ 省略 ～

    // ジャンプ量
    VECTOR jumpPow_;

    ～ 省略 ～

    // 重力計算
    void CalcGravityPow(void);
```

CharactorBase.cpp

```
void CharactorBase::Update(void)
{

    // 各キャラクターごとの更新処理
    UpdateProcess();

    // 移動方向に応じた遅延回転
    DelayRotate();

    // 重力による移動量
    CalcGravityPow();

    // モデル制御更新
    transform_.Update();

    ～ 省略 ～

}
```

```
void CharactorBase::CalcGravityPow(void)
{

    // 重力方向
    VECTOR dirGravity = AsoUtility::DIR_D;

    // 重力の強さ
    float gravityPow = ○○○ * scnMng_.GetDeltaTime();

    // 重力
    VECTOR gravity = VScale(dirGravity, gravityPow);
    jumpPow_ = VAdd(jumpPow_, gravity);

    // ジャンプ量を加算
    transform_.pos = VAdd(transform_.pos, jumpPow_);

}
```

解説

○ Update関数の処理フロー

UpdateProcess関数内で、
派生クラス(各キャラクター)独自のジャンプや飛行処理を
行う想定のため、重力の処理は、その後に実行するようにしています。

○ 重力の計算

現実世界の重力加速度 9.8 m/s は、
1秒ごとに9.8メートルずつ速度が増えるという定数になりますが、
60FPSゲームの場合、1フレームあたりの秒数は0.0167秒になりますので、
値が大きくなり過ぎないように、デルタタイムを乗算する必要があります。