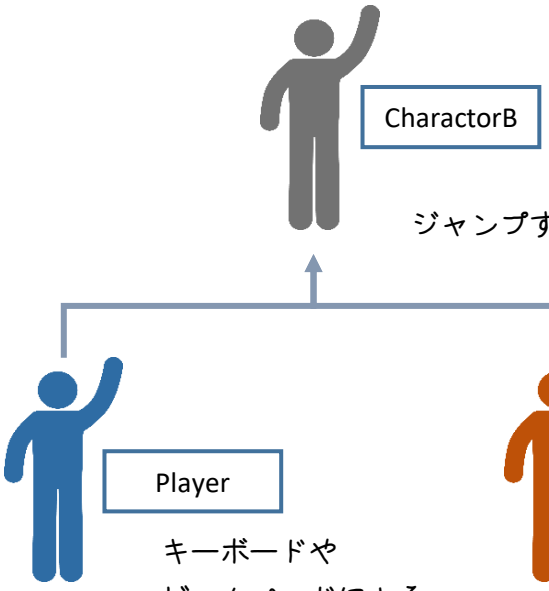
# ジャンプ

現在の設計を考慮しつつ、ジャンプ処理を実装していきます。




void ProcessJump(void);

ジャンプ力は、キャラクターによって異なる可能性がある。

CharactorBase.h

```
protected:

    ～ 省略 ～

    // ジャンプ判定
    bool isJump_;
```

CharactorBase.cpp

```
void CharactorBase::CollisionGravity(void)
{

    ～ 省略 ～

        if (hit.HitFlag > 0)
```

```
        {

            // 衝突地点から、少し上に移動
            transform_.pos = VAdd(
                hit.HitPosition, VScale(AsoUtility::DIR_U, 2.0f));

            // ジャンプリセット
            jumpPow_ = AsoUtility::VECTOR_ZERO;

            // ジャンプ判定
            isJump_ = false;

        }

    ～ 省略 ～

}
```

Player.h

```
public:

    // アニメーション種別
    enum class ANIM_TYPE
    {
        IDLE,
        RUN,
        FAST_RUN,
        JUMP,
    };

private:

    ～ 省略 ～

    // 衝突判定用線分開始(ジャンプ時)
    static constexpr VECTOR COL_LINE_JUMP_START_LOCAL_POS =
    { 0.0f, 130.0f, 0.0f };

    // 衝突判定用線分終了(ジャンプ時)
    static constexpr VECTOR COL_LINE_JUMP_END_LOCAL_POS =
```

```
    { 0.0f, 50.0f, 0.0f };

    // ジャンプ力
    static constexpr float POW_JUMP = ○○○

    // 操作
    void ProcessMove(void);
    void ProcessJump(void);

    // デバッグ描画
    void DrawDebug(void);

};
```

Player.cpp

```
void Player::InitAnimation(void)
{

    ～ 省略 ～

    animationController_->Add(
        static_cast<int>(ANIM_TYPE::JUMP), 60.0f, path + "JumpRising.mvl");

    // 初期アニメーション
    animationController_->Play(static_cast<int>(ANIM_TYPE::IDLE), true);

}

void Player::UpdateProcess(void)
{

    // 移動操作
    ProcessMove();

    // ジャンプ処理
    ProcessJump();

}

void Player::ProcessMove(void)
```

```
{

    ～ 省略 ～

    if (!AsoUtility::EqualsVZero(dir))
    {

        // 移動スピード
        moveSpeed_ = SPEED_MOVE;
        if (isDash)
        {
            moveSpeed_ = SPEED_DASH;
        }

        // ジャンプ中はアニメーションを変えない
        if (!isJump_)
        {

            // アニメーション
            if (isDash)
            {
                animationController_->Play(
                    static_cast<int>(ANIM_TYPE::FAST_RUN), true);
            }
            else
            {
                animationController_->Play(
                    static_cast<int>(ANIM_TYPE::RUN), true);
            }

        }

        // Y軸のみのカメラ角度を取得
        Quaternion cameraRot = scnMng_.GetCamera()->GetQuaRotY();

        // 移動方向をカメラに合わせる
        moveDir_ = Quaternion::PosAxis(cameraRot, dir);

        // 移動量を計算
        movePow_ = VScale(moveDir_, moveSpeed_);
```

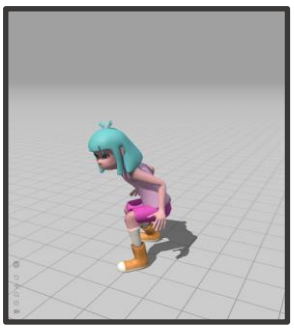
```
    }
    else
    {
        // ジャンプ中はアニメーションを変えない
        if (!isJump_)
        {
            // IDLE状態に戻す
            animationController_->Play(
                static_cast<int>(ANIM_TYPE::IDLE), true);
        }
    }

}

void Player::ProcessJump(void)
{

    auto& ins = InputManager::GetInstance();
    bool isHitKey = ins.IsTrgDown(KEY_INPUT_BACKSLASH)
        || ins.IsPadBtnTrgDown(
            InputManager::JOYPAD_NO::PAD1,
                InputManager::JOYPAD_BTN::DOWN);

    // ジャンプ
    if (isHitKey && !isJump_)
    {

        // ジャンプ量の計算
        float jumpSpeed = POW_JUMP * scnMng_.GetDeltaTime();
        jumpPow_ = VScale(AsoUtility::DIR_U, jumpSpeed);
        isJump_ = true;

        // アニメーション再生
        animationController_->Play(
            static_cast<int>(ANIM_TYPE::JUMP), false);

    }

}
```

**解説**

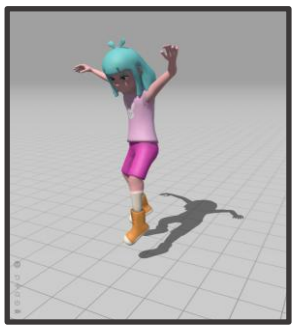
○ ジャンプアニメーションは本来３段階

mixamoだと、ふんばりも追加した４段階あり、
それらが1つのアニメーションとしてセットになっています。

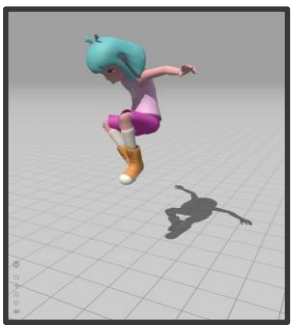
ふんばりアニメーション



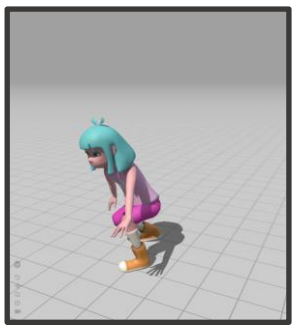
上昇アニメーション



滞空アニメーション



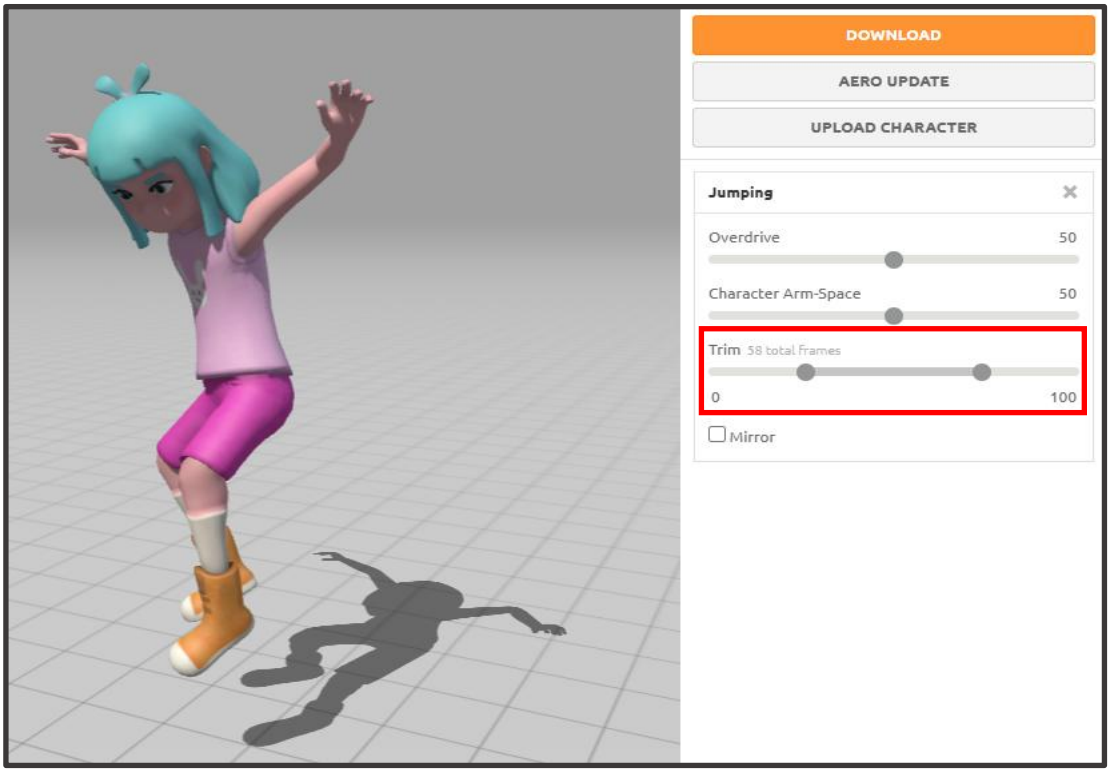
着地アニメーション



それらが1つのアニメーションとしてセットになっています。
今回用意している、"JumpRising.mvl"は、
上昇アニメーションとして、mixamoから切り抜いたアニメーションです。

今回は、上昇アニメーションのみ使用予定です。

切り抜きは、Trim項目を使用しましょう。




【要件】

Playerのジャンプ力を設定しましょう。

- 1フレームあたりの重力は、重力 * 重力調整 * デルタタイム
  981 * 0.7 * 0.0167 = およそ11.5

- ジャンプ力を500とした場合、500 * 0.0167 = 8.35 となり、
  初めから重力より弱い状態になるので、
  移動値による上昇は行われない

- ジャンプ力を1000とした場合、1000 * 0.0167 = 16.57 となり、
  16.57 - 11.5 = 5.07、5.07 - 11.5 = -6.43 と移動量が計算され、
  初期座標が 0 の場合、0 → 5.07 → -1.36 衝突判定により、0に押し戻し
  2フレーム後には、着地する

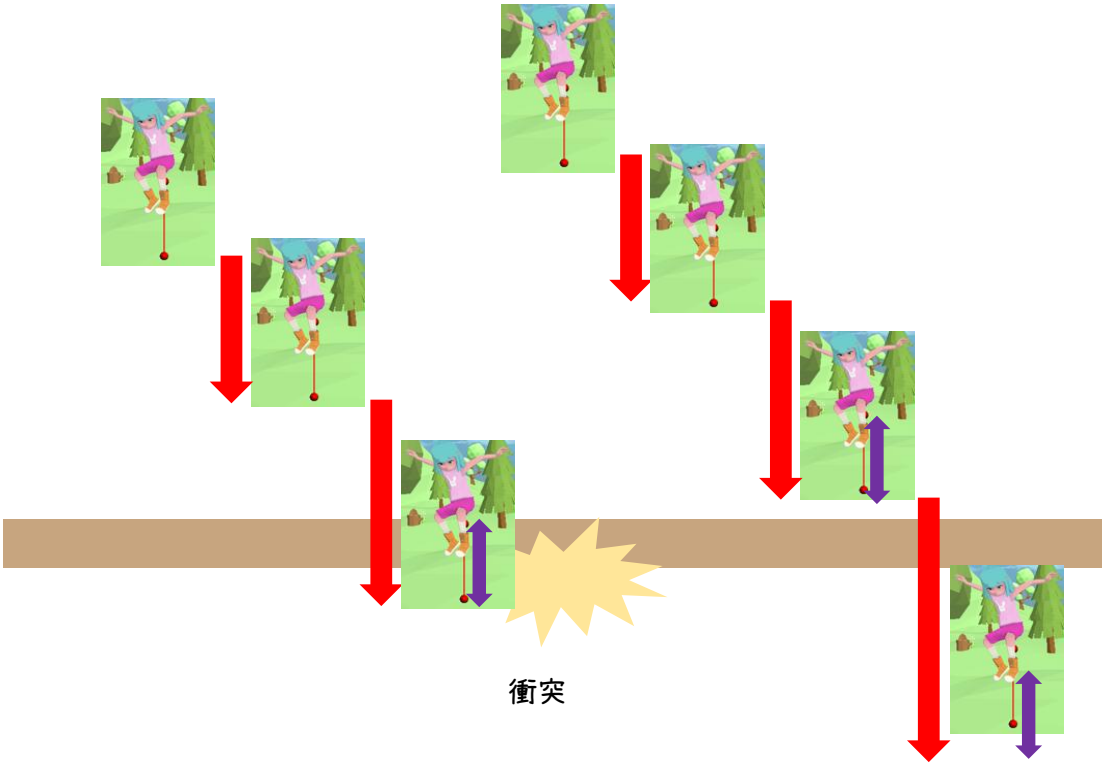
【目標】

おおよそ、木の葉あたりまでジャンプできる。

## 地面突き抜け問題

ジャンプ力を大きく設定した場合、高くジャンプできまずが、
地面を突き抜けてしまう不具合も発生してしまいます。

これは、重力の加速度により、移動量がどんどん増加していくのですが、
その移動量が、衝突判定で作成した線分の長さを超えてしまう場合に発生します。




対応としては、移動前座標と移動後座標にも、
衝突線分を作って、絶対に突き抜けないように制御を行うか、
もしくは、速度が大きくなり過ぎないように、速度制限を行うかです。

速度制限を行う場合、今回のゲームのように、
重力がYのマイナス方向にしか向かないのであれば、
下記のように単純な計算式で制御することができます。

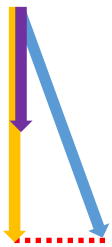
```
if(jumpPow_.y < -30.0f) {
    jumpPow_.y = -30.0f;
}
```

もし、重力方向が斜めだったり、固定出ない場合は、
内積を取って、以下のように実装します。

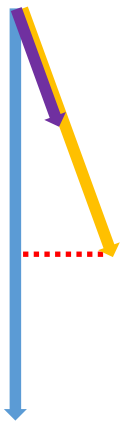
```
// 現在の速度のうち、重力方向の成分を算出
float fallSpeed = VDot(jumpPow_, dirGravity);

// 落下方向に進みすぎているなら制限
if (fallSpeed > MAX_FALL_SPEED)
{
    // 現在の速度から、超過分だけ打ち消す
    VECTOR limit = VScale(dirGravity, MAX_FALL_SPEED);
    VECTOR horiz = VSub(jumpPow_, VScale(dirGravity, fallSpeed));
    jumpPow_ = VAdd(horiz, limit);
}
```

内積を使うと、対象方向におけるベクトルの長さを抽出することができます。

ジャンプ力
重力方向
ジャンプ力における重力方向の強さ(長さ)



| | x | y | z |
|---|---|---|---|
| 重力 | 0.0 | -1.0 | 0.0 |
| ジャンプ力 | 15.0 | -25.0 | 0.0 |
| 内積式 | 0 * 30 | -1 * -25 | 0 * 0 |
| 内積結果 | 25.0 | | |



| | x | y | z |
|---|---|---|---|
| 重力 | 0.7 | -0.7 | 0.0 |
| ジャンプ力 | 0.0 | -40.0 | 0.0 |
| 内積式 | 0.7 * 0 | -0.7 * -40 | 0 * 0 |
| 内積結果 | 28.0 | | |

重力方向におけるジャンプ力の制限を30.0とした場合、
どちらも、制限には引っかからない。

| | x | y | z |
|---|---|---|---|
| 重力 | 0.0 | -1.0 | 0.0 |
| ジャンプ力 | 0.0 | -32.0 | 0.0 |
| 内積式 | 0 * 0 | -1 * -32 | 0 * 0 |
| 内積結果 | 32.0 | | |

3つのサンプルの中だと、ジャンプ力単体のベクトルの
大きさは、最も小さいが、重力方向と一致しているため
このパターンのみが、ジャンプ力に制限がかかる。

少し難しいですが、方向(角度)比較同様に、
内積は計算コストも軽く、様々なゲーム判定に応用できますで、
理解できた方は、諸々試してみましょう。

教材としては、シンプルな式で判断していきます。

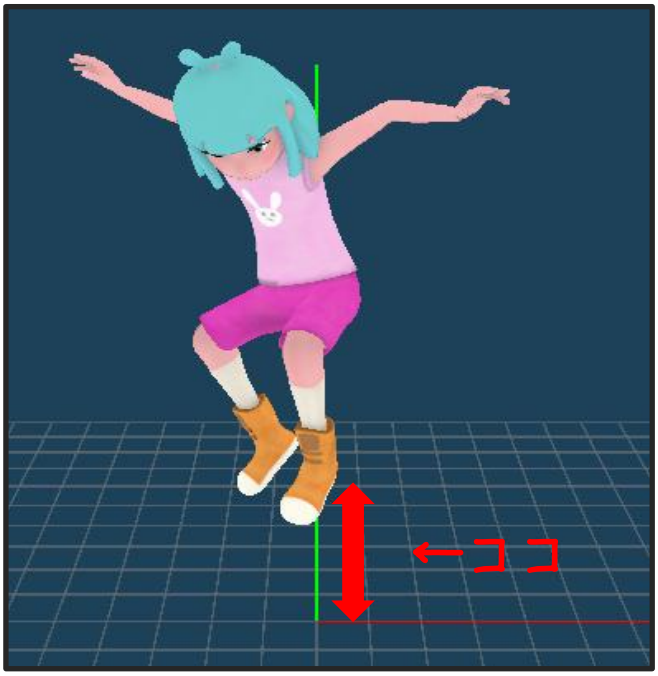
CharactorBase.h

```
protected:

    // 最大落下速度
    static constexpr float MAX_FALL_SPEED = -30.0f;
```

CharactorBase.cpp

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

    ～ 省略 ～

    // 重力速度の制限
    if (jumpPow_.y < MAX_FALL_SPEED)
    {
        jumpPow_.y = MAX_FALL_SPEED;
    }

}
```

## アニメーションによる座標移動の問題




キャラクター座標は
変わっていませんが、
見た目上では、
明らかにYの値が
増えている

アニメーションによっては、ルートフレームの座標移動が
含まれている場合があり、キャラクターの座標自体は変わっていないのですが、
アニメーション上で、移動しているように見えてしまいます。

そうなると、見た目とプログラム上での衝突判定に差異が出て、
また、足が接地していないように見えるのに、
地面と線分が衝突して、着地判定となってしまいます。

その場合は、
アニメーションによる移動値を無くしたり、
そのまま移動量として算出して、座標に加えるなどやり方があり、
DxLibのホームページでも紹介されています。

■サンプルプログラム　アニメーションによる座標移動
https://dxlib.xsrv.jp/program/dxprogram_AnimationMove.html

しっかり付いてこれている方は、ぜひチャレンジしてみてください。

授業では、もっと簡単なやり方で、
ジャンプ中は、線分の相対座標を変えて対応していきたいと思います。

Player.cpp

```
void Player::UpdateProcess(void)
{

    // 移動操作
    ProcessMove();

    // ジャンプ処理
    ProcessJump();

    // アニメーションごとの線分調整
    if (animationController_->GetPlayType() == static_cast<int>(ANIM_TYPE::JUMP))
    {
        // ジャンプ中は線分を伸ばす
        if (ownColliders_.count(static_cast<int>(COLLIDER_TYPE::LINE)) != 0)
        {
            ColliderLine* colLine = dynamic_cast<ColliderLine*>(
                ownColliders_.at(static_cast<int>(COLLIDER_TYPE::LINE)));
            colLine->SetLocalPosStart(COL_LINE_JUMP_START_LOCAL_POS);
            colLine->SetLocalPosEnd(COL_LINE_JUMP_END_LOCAL_POS);
        }
    }
    else
    {
        // 通常時の線分に戻す
        if (ownColliders_.count(static_cast<int>(COLLIDER_TYPE::LINE)) != 0)
        {
            ColliderLine* colLine = dynamic_cast<ColliderLine*>(
                ownColliders_.at(static_cast<int>(COLLIDER_TYPE::LINE)));
            colLine->SetLocalPosStart(COL_LINE_START_LOCAL_POS);
            colLine->SetLocalPosEnd(COL_LINE_END_LOCAL_POS);
        }
    }

}
```