# ワープの実装

次の惑星に移動するためのワープスターの機能を実装していきましょう。

まず最初に、スターとプレイヤーの衝突判定を行います。

**演習①　　スター(球体)とプレイヤー(点)の衝突判定を実装し、**
**WarpStarのRESERVE状態を実装しましょう**

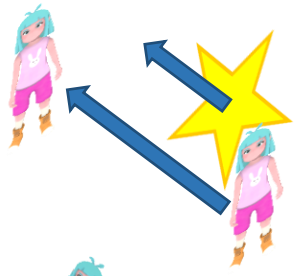
- 球体とカプセルの中心座標が衝突したら、
  RESERVE状態へ遷移する

- RESERVE状態も、Z回転とエフェクトを発生させ続ける

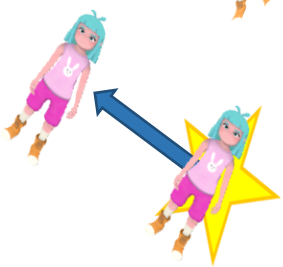
- IDLE時よりも、Z回転のスピードを速くする
  SPEED_ROT_RESERVE

## ワープ前の準備

衝突したら、スターが向いている方向にプレイヤーを移動させたいのですが、
その前に、プレイヤーをスターの中心(移動開始地点)に移動させ、
プレイヤーの向きをスターの向きに回転させる必要があります。



しっかり制御しないと、
変な向きで、
変な位置から移動を開始して
しまいますので、



位置は線形補間、
回転は球面補間を使用して、
徐々に移動準備を行います。

WarpStar側から、Playerへワープ準備の指示を出し、
その際に、プレイヤーに移動して貰いたい位置と、向き(回転)の情報も
一緒に渡していきます。

WarpStar.h

```
private:

    // ワープ(移動する)方向
    Quaternion warpQua_;

    // ワープ開始座標(ワールド座標)
    VECTOR warpStartPos_;
```

WarpStar.cpp

```
void WarpStar::Init(void)
{

    ～ 省略 ～

    // Zが無回転の状態を保持しておく
    VECTOR angles = transform_.quaRot.ToEuler();
    warpQua_ = Quaternion::Euler(angles.x, angles.y, 0.0f);

    // ワープ開始座標(ワールド座標)
    warpStartPos_ =
        VAdd(transform_.pos, warpQua_.PosAxis(WARP_RELATIVE_POS));

    ～ 省略 ～

}

void WarpStar::ChangeStateReserve(void)
{

    stateUpdate_ = std::bind(&WarpStar::UpdateReserve, this);

    // プレイヤーの状態を変更
    player_.StartWarpReserve(TIME_WARP_RESERVE, warpQua_, warpStartPos_);

}
```

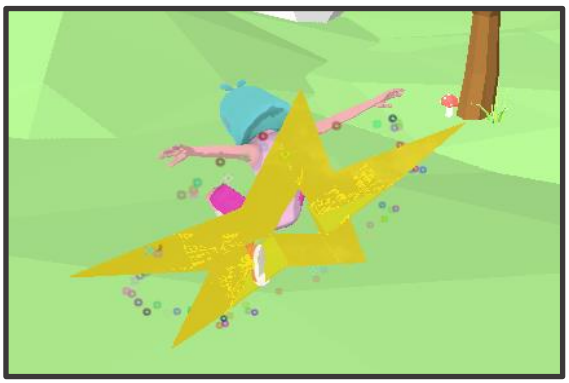
Player.h

```
public:

    ～ 省略 ～

    // ワープ準備開始
    void StartWarpReserve(
        float time, const Quaternion& goalRot, const VECTOR& goalPos);

private:

    ～ 省略 ～

    // ワープ準備時間
    float timeWarp_;

    // ワープ準備経過時間
    float stepWarp_;

    // ワープ準備完了時の回転
    Quaternion warpQua_;

    // ワープ準備完了時の座標
    VECTOR warpReservePos_;

    // ワープ準備開始時のプレイヤー情報
    Quaternion reserveStartQua_;
    VECTOR reserveStartPos_;

    ～ 省略 ～

    void ChangeStateWarpReserve(void);

    ～ 省略 ～

    void UpdateWarpReserve(void);
```

Player.cpp

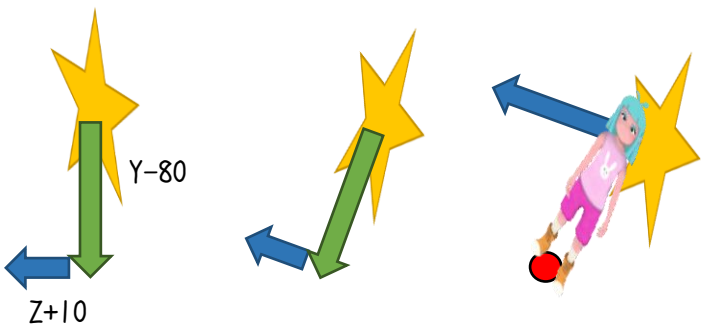
```
Player::Player(void)
{

    ～ 省略 ～

    reserveStartPos_ = AsoUtility::VECTOR_ZERO;
    stepWarp_ = 0.0f;
    timeWarp_ = 0.0f;
    warpReservePos_ = AsoUtility::VECTOR_ZERO;

    stateChanges_.emplace(
        STATE::WARP_RESERVE, std::bind(&Player::ChangeStateWarpReserve, this));

}

void Player::StartWarpReserve(
    float time, const Quaternion& goalRot, const VECTOR& goalPos)
{

    // ワープ準備時間
    timeWarp_ = time;

    // ワープ準備経過時間
    stepWarp_ = time;

    // ワープ準備完了時の回転
    warpQua_ = goalRot;

    // ワープ準備完了時の座標
    warpReservePos_ = goalPos;

    ChangeState(STATE::WARP_RESERVE);

}
```

```
void Player::ChangeStateWarpReserve(void)
{

    stateUpdate_ = std::bind(&Player::UpdateWarpReserve, this);

    jumpPow_ = AsoUtility::VECTOR_ZERO;

    // ワープ準備開始時のプレイヤー情報
    reserveStartQua_ = transform_.quaRot;
    reserveStartPos_ = transform_.pos;
    animationController_->Play((int)Player::ANIM_TYPE::WARP_PAUSE);

}

void Player::UpdateWarpReserve(void)
{
    // 演習②　位置は線形補間、回転は球面補間を使用して、
    //         現在のプレイヤー情報から、指定の位置、回転に
    //         徐々に遷移するように実装してください
}
```



プレイヤーがワープ方向を向いて、星の手前の位置に移動できたらＯＫです。

※　ワープ開始座標
　　の補足



**演習③　　ワープ準備が整ったら、プレイヤー状態をWARP_MOVEに**
**遷移させて、ワープ方向に移動するよう実装してください**


星に向かって
異動ができたらＯＫ

アニメーションは
FLY を使用

上記が実装できたらで良いですが、
WarpStarの後始末のため、Playerの状態把握用の、
　IsPlay
　IsWarpMove
上記関数も実装してください。

WarpStar.cpp

```
void WarpStar::UpdateReserve(void)
{

    ～ 省略 ～

    if (player_.IsWarpMove())
    {
        ChangeState(STATE::MOVE);
    }

}

void WarpStar::UpdateMove(void)
{
    if (player_.IsPlay())
    {
        ChangeState(STATE::IDLE);
    }
}
```

## ステージ(惑星)の切り替え

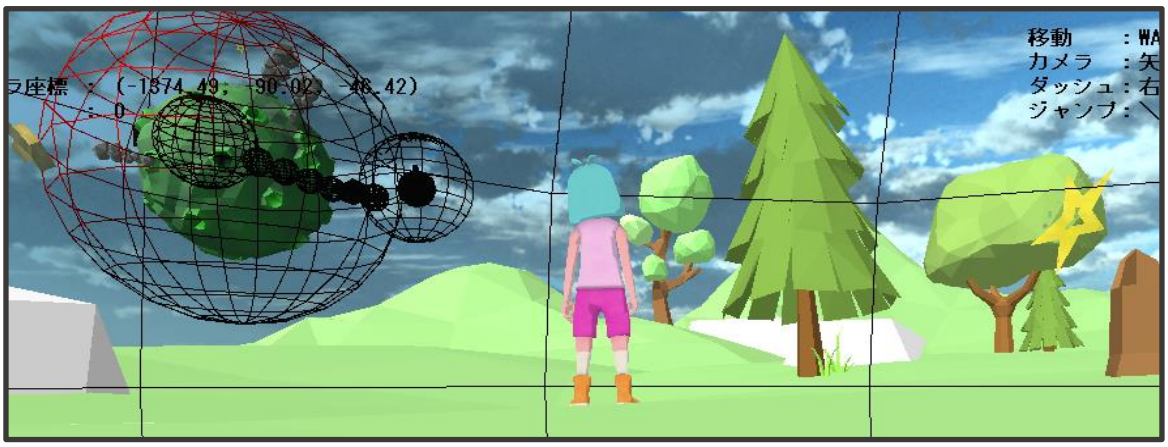
ワープ後の、次のステージへ切り替える処理を実装していきます。
惑星１つ１つを管理しているのは、Planetクラス、
全ての惑星を管理しているのが、Stageクラスです。

今いる惑星とは、別の惑星の重力範囲にプレイヤーが入ったら、
その惑星に切り替えるように実装したいと思います。
重力範囲は、惑星座標を中心とした球体にしていきます。

まずは、確認用に重力範囲を可視化していきます。

Planet.cpp

```
void Planet::Draw(void)
{

    MVIDrawModel(transform_.modelId);

    // 重力範囲
    DrawSphere3D(
        transform_.pos, gravityRadius_, 10, 0xff0000, 0xff0000, false);

}
```




こんな常時になっていれば、ＯＫです。

次に、惑星が切り替わったことが確認できるように、
以下を実装してきます。

Planet.cpp

```
void Player::DrawDebug(void)
{

    ～ 省略 ～

    // アクティブな惑星
    DrawFormatString(20, 80, black, "惑星        : %d",
        (int)grvMng_.GetActivePlanet().lock()->GetName()
    );

}
```

それでは演習に入ります。

**演習㊹　　プレイヤーが、今いる惑星以外の惑星の重力範囲に**
**入ったら、アクティブな惑星を切り替えるようにしてください**

惑星が切り替わる条件１

一定時間(isPossibleChange)経過し、現在の惑星と異なる場合

惑星が切り替わる条件２

プレイヤーの座標が惑星の重力圏内に入った場合
※PlanetクラスのInRangeGravity関数を実装する必要がある

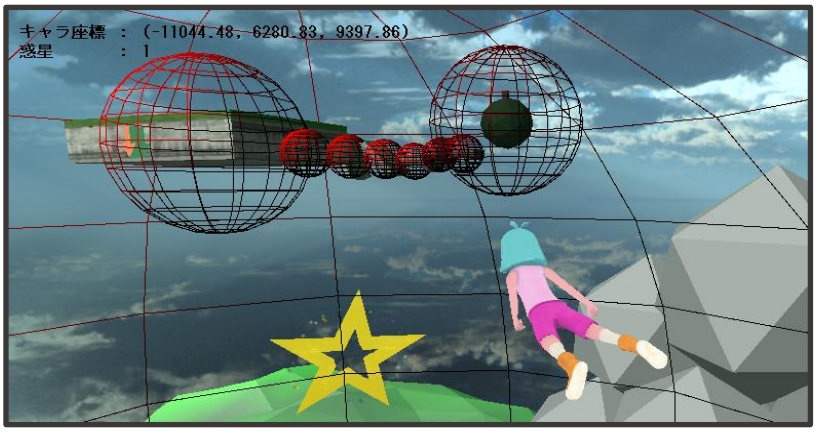
以上の条件を満たした場合、アクティブな惑星を切り替える

```
// 次のステージへ遷移
ChangeStage(s.second->GetName());

// 以降のループでステージが変わらないようにする
isPossibleChange = false;
```

実装箇所

Planet.cpp

```
void Stage::Update(void)
{

    ～ 省略 ～

    // 重力範囲が重なっていた場合、惑星がコロコロ切り替わらないように
    // 一定時間ステージが変わらないようにする
    bool isPossibleChange = true;
    if (step_ > 0.0f)
    {
        step_ -= SceneManager::GetInstance().GetDeltaTime();
        isPossibleChange = false;
    }

    // 惑星
    for (const auto& s : planets_)
    {

        s.second->Update();

        // 演習の実装箇所

    }

}
```




次の惑星の
重力圏内に入ったら


惑星の表示が
1 に切り替わる

## プレイヤーを落下状態へ遷移させる

ワープ前の惑星情報を確保しておいて、
惑星が切り替わったら、WARP_MOVEから通常のPLAY状態へ遷移させます。

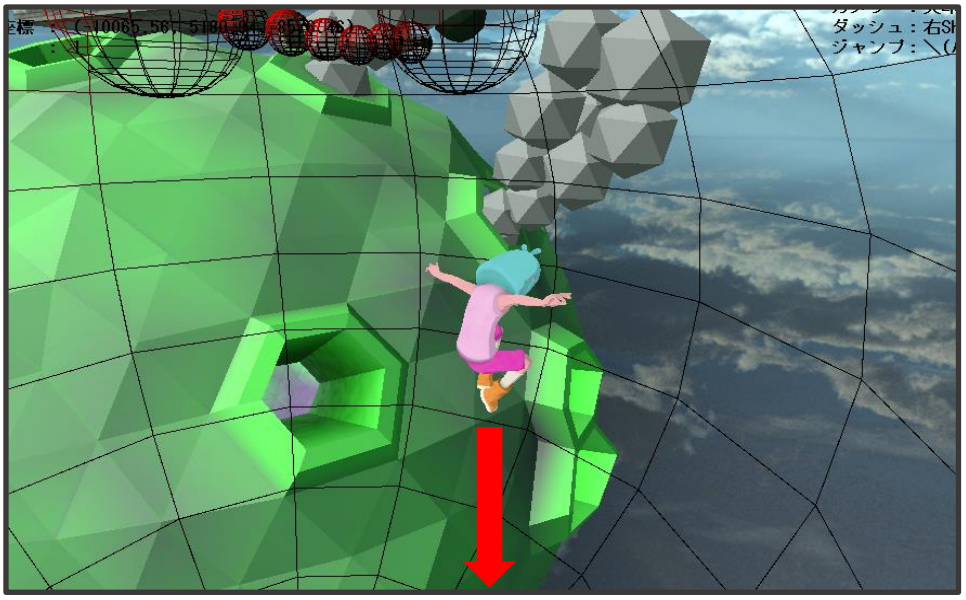
Player.h

```
#include "Stage.h"

class Player : public ActorBase
{

private:

    ～ 省略 ～

    // ワープ前の惑星名
    Stage::NAME preWarpName_;
```

Player.cpp

```
Player::Player(void)
{

    ～ 省略 ～

    preWarpName_ = Stage::NAME::MAIN_PLANET;

}

void Player::StartWarpReserve(
    float time, const Quaternion& goalRot, const VECTOR& goalPos)
{

    ～ 省略 ～

    // ワープ前の惑星情報を保持
    preWarpName_ = grvMng_.GetActivePlanet().lock()->GetName();

    ChangeState(STATE::WARP_RESERVE);
```

```
}


void Player::UpdateWarpMove(void)
{

    ～ 省略 ～

    // 次の惑星に切り替わったらワープ状態からプレイ状態へ切り替える
    Stage::NAME name = grvMng_.GetActivePlanet().lock()->GetName();
    if (name != preWarpName_)
    {

        // 落下アニメーション
        animationController_->Play((int)ANIM_TYPE::JUMP, true, 13.0f, 25.0f);
        animationController_->SetEndLoop(23.0f, 25.0f, 5.0f);
        ChangeState(Player::STATE::PLAY);
        return;

    }

}
```



重力圏内に入ったら、ワープ移動止めて、
Yのマイナス方向に落下が始まったらＯＫです。