3D アクションゲームプログラミング

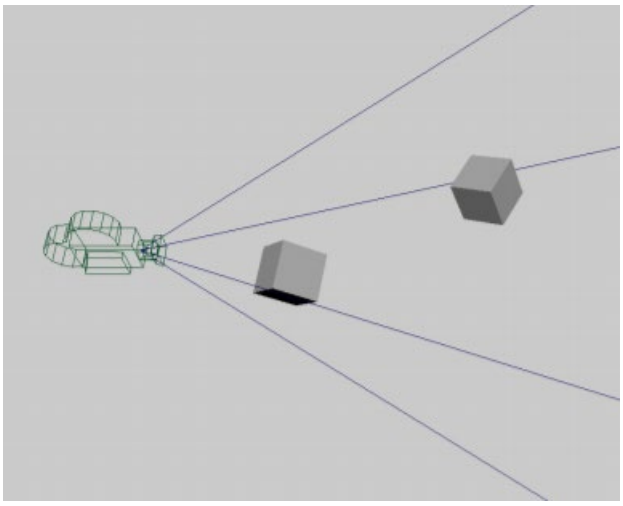
○評価要件

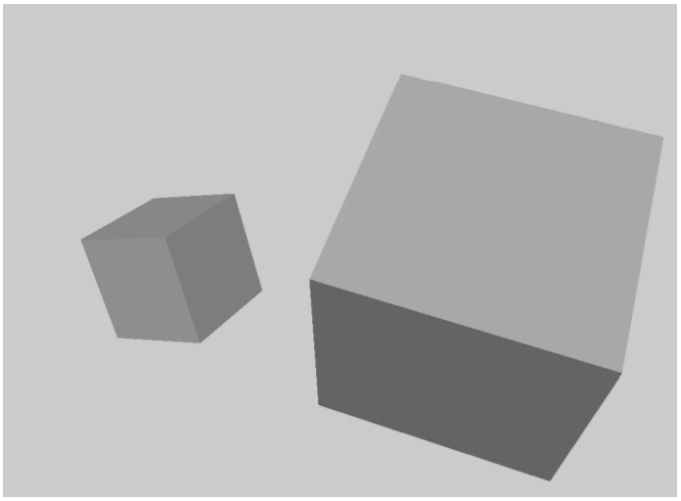
☑カメラクラスの作成
☑三人称視点カメラ操作
☑三人称視点カメラ X 軸回転制限

○概要

３D ゲームでは３D 空間にステージやキャラクターなどが配置されています。
この３D 空間の景色を画面に表示するために「カメラ」という概念について学習しましょう。

３D 空間内にカメラが存在し、カメラから見た世界が画面に映し出されているのです。



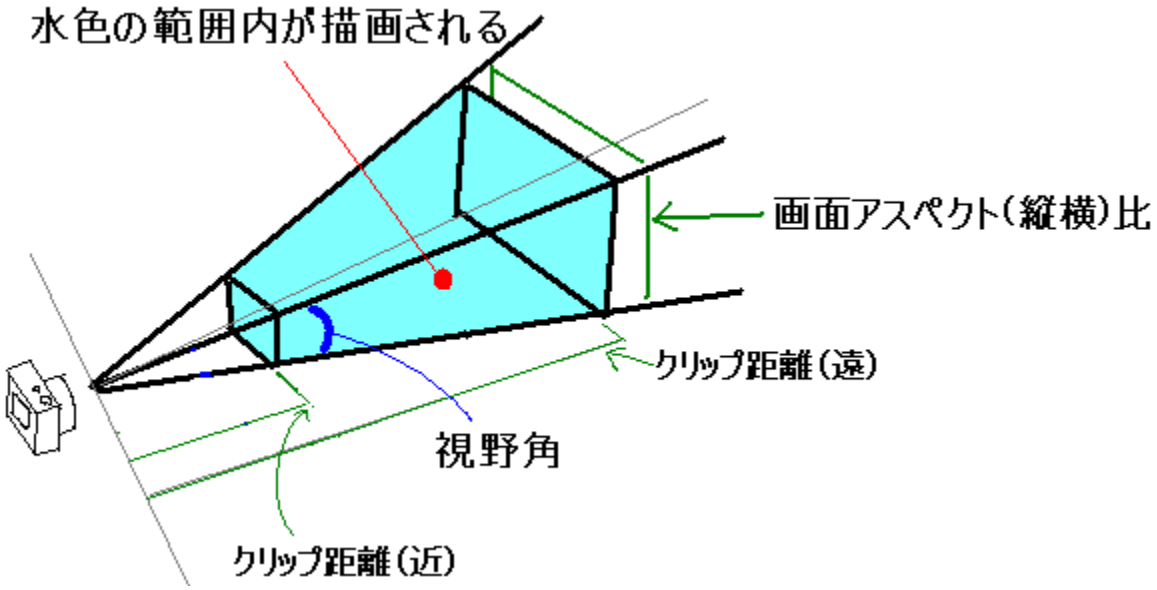


今回は簡単にカメラの情報の設定や情報の取得ができるカメラクラスと
三人称視点でカメラを操作するカメラコントローラークラスを作成します。

○カメラに必要な情報

まず、カメラに必要な情報を把握しましょう。

Eye	カメラの視点。カメラの位置情報。
Focus	カメラの注視点。カメラが注目する位置。
Up	カメラの上方向。
Fov	カメラの視野角。
Aspect	画面の縦横比率。（画面幅÷画面高さ）
NearZ	クリップ距離（近）。この距離よりも近いオブジェクトは表示しない。
FarZ	クリップ距離（遠）。この距離よりも遠いオブジェクトは表示しない。



今現在、ステージやキャラクターが画面に表示されているのは、SceneGame.cpp の Render()関数内であらかじめカメラの設定をしているからです。

SceneGame.cpp を見てみましょう。

```
// 描画処理
void SceneGame::Render()
{
    ---省略---

    // 描画処理
    RenderContext rc;
    rc.lightDirection = { 0.0f, -1.0f, 0.0f, 0.0f };  // ライト方向（下方向）

    // ビュー行列
    {
```

描画するために必要な情報をまとめた構造体

```
        DirectX::XMFLOAT3 eye = { 0, 10, -10 };    // カメラの視点（位置）
        DirectX::XMFLOAT3 focus = { 0, 0, 0 };// カメラの注視点（ターゲット）
        DirectX::XMFLOAT3 up = { 0, 1, 0 };        // カメラの上方向

        DirectX::XMVECTOR Eye = DirectX::XMLoadFloat3(&eye);
        DirectX::XMVECTOR Focus = DirectX::XMLoadFloat3(&focus);
        DirectX::XMVECTOR Up = DirectX::XMLoadFloat3(&up);
        DirectX::XMMATRIX View = DirectX::XMMatrixLookAtLH(Eye, Focus, Up);
        DirectX::XMStoreFloat4x4(&rc.view, View);
    }

    // プロジェクション行列
    {
        // 視野角
        float fovY = DirectX::XMConvertToRadians(45);
        // 画面縦横比率
        float aspectRatio = graphics.GetScreenWidth() / graphics.GetScreenHeight ();
        // カメラが映し出すの最近距離
        float nearZ = 0.1f;
        // カメラが映し出すの最遠距離
        float farZ = 1000.0f;
        // プロジェクション行列作成
        DirectX::XMMATRIX Projection = DirectX::XMMatrixPerspectiveFovLH(fovY, aspectRatio,
                                                                         nearZ, farZ);
        DirectX::XMStoreFloat4x4(&rc.projection, Projection);
    }

    // 3Dモデル描画
    {
        Shader* shader = graphics.GetShader();
        shader->Begin(dc, rc);

        // ステージ描画
        stage->Render(dc, shader);
        // プレイヤー描画
        player->Render(dc, shader);
```

視点、注視点、上方向の情報をもとにビュー行列を作成

作成したビュー行列を rc.view に保存

視野角、画面比率、クリップ距離の情報をもとにプロジェクション行列を作成

作成したプロジェクション行列を rc.projection に保存

シェーダーにカメラの情報を渡す

```
        shader->End(dc);
    }

    ---省略---
}
```

上記のプログラムから見てわかるようにカメラにはビュー行列とプロジェクション行列が必要です。
ビュー行列はカメラ自身の位置や姿勢情報です。
プロジェクション行列はカメラが見た世界をスクリーン画面に映し出すための情報です。

○カメラクラス

先ほどのプログラムではゲームプログラミングをする上で扱いづらいのでクラス化しましょう。
今回はどこでもカメラの情報を取り出せるように扱いやすいシングルトンにします。
シングルトンとはクラスの実態を世界で一つだけにするという考えのことです。
シングルトン化することでカメラクラス内の関数でしかカメラクラスを実体化できなくなります。
Camera.cpp と Camera.h を作成し、下記プログラムコードを記述しましょう。

Camera.h

```
#pragma once

#include <DirectXMath.h>

// カメラ
class Camera
{
private:
    Camera() {}
    ~Camera() {}

public:
    // 唯一のインスタンス取得
    static Camera& Instance()
    {
        static Camera camera;
        return camera;
    }

    // 指定方向を向く
    void SetLookAt(const DirectX::XMFLOAT3& eye, const DirectX::XMFLOAT3& focus,
                                                 const DirectX::XMFLOAT3& up);

    // パースペクティブ設定
    void SetPerspectiveFov(float fovY, float aspect, float nearZ, float farZ);
```

シングルトン化するために
コンストラクタをプライベート化する。

カメラは様々な所で情報を取り出したいので
世界に一つしか存在しないシングルトンにする。

```
    // ビュー行列取得
    const DirectX::XMFLOAT4X4& GetView() const { return view; }

    // プロジェクション行列取得
    const DirectX::XMFLOAT4X4& GetProjection() const { return projection; }

private:
    DirectX::XMFLOAT4X4    view;
    DirectX::XMFLOAT4X4    projection;
};
```

Camera.cpp

```
#include "Camera.h"

// 指定方向を向く
void Camera::SetLookAt(const DirectX::XMFLOAT3& eye, const DirectX::XMFLOAT3& focus, const
DirectX::XMFLOAT3& up)
{
    // 視点、注視点、上方向からビュー行列を作成
    DirectX::XMVECTOR Eye   =
    DirectX::XMVECTOR Focus =
    DirectX::XMVECTOR Up    =
    DirectX::XMMATRIX View  =
    DirectX::XMStoreFloat4x4(&view, View);
}

// パースペクティブ設定
void Camera::SetPerspectiveFov(float fovY, float aspect, float nearZ, float farZ)
{
    // 画角、画面比率、クリップ距離からプロジェクション行列を作成
    DirectX::XMMATRIX Projection =
    DirectX::XMStoreFloat4x4(&projection, Projection);
}
```

SceneGame.cpp を開き、下記プログラムコードを追記&修正しましょう。

SceneGame.cpp

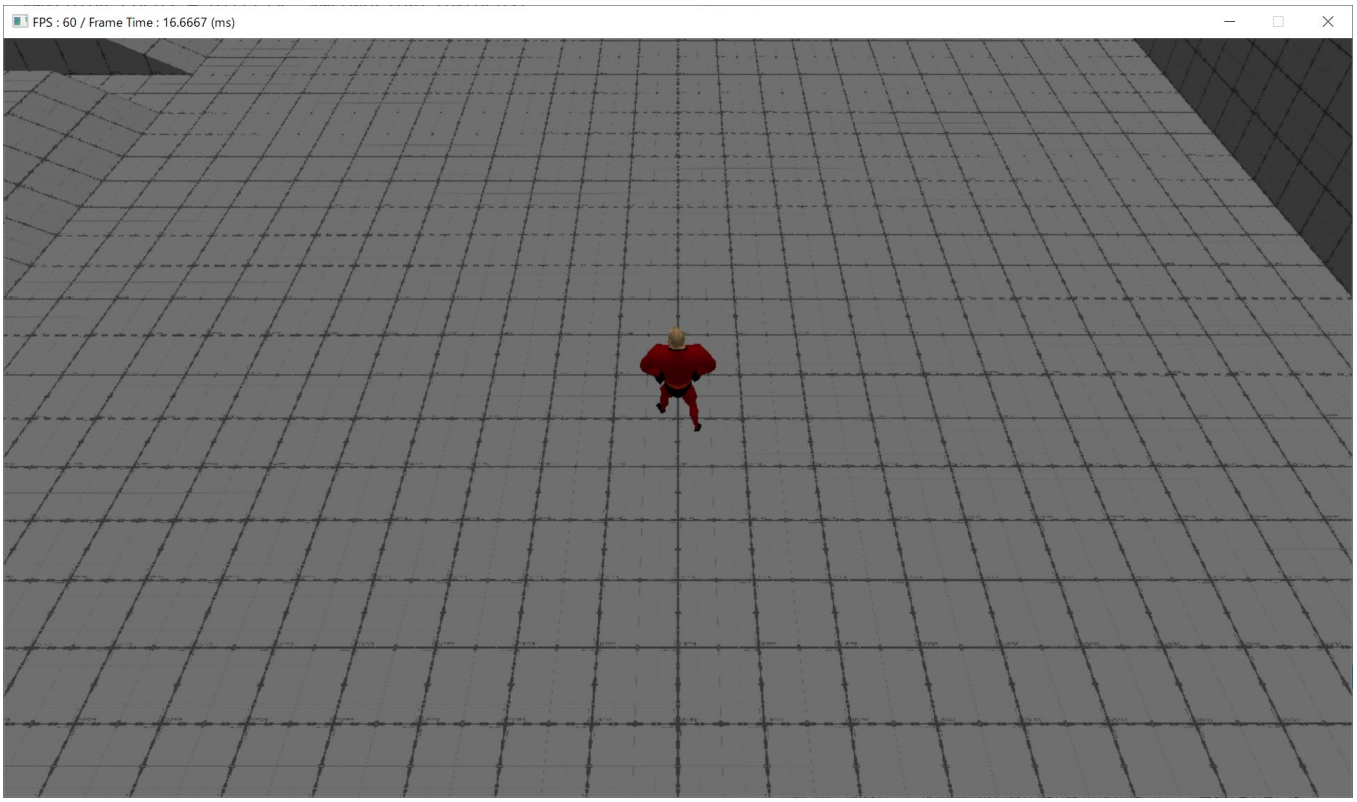
```
---省略---
#include "Camera.h"

// 初期化
void SceneGame::Initialize()
{
    ---省略---

    // カメラ初期設定
    Graphics& graphics = Graphics::Instance();
    Camera& camera = Camera::Instance();
    camera.SetLookAt(
        DirectX::XMFLOAT3(0, 10, -10),
        DirectX::XMFLOAT3(0, 0, 0),
```

```
        DirectX::XMFLOAT3(0, 1, 0)
    );
    camera.SetPerspectiveFov(
        DirectX::XMConvertToRadians(45),
        graphics.GetScreenWidth() / graphics.GetScreenHeight(),
        0.1f,
        1000.0f
    );
}
---省略---

// 描画処理
void SceneGame::Render()
{
    ---省略---
    // 描画処理
    RenderContext rc;
    rc.lightDirection = { 0.0f, -1.0f, 0.0f, 0.0f };  // ライト方向（下方向）

    // カメラパラメータ設定
    Camera& camera = Camera::Instance();
    rc.view = camera.GetView();
    rc.projection = camera.GetProjection();

    // ビュー行列
    {
        DirectX::XMFLOAT3 eye = { 0, 10, -10 };    // カメラの視点（位置）
        DirectX::XMFLOAT3 focus = { 0, 0, 0 };// カメラの注視点（ターゲット）
        DirectX::XMFLOAT3 up = { 0, 1, 0 };        // カメラの上方向

        DirectX::XMVECTOR Eye = DirectX::XMLoadFloat3(&eye);
        DirectX::XMVECTOR Focus = DirectX::XMLoadFloat3(&focus);
        DirectX::XMVECTOR Up = DirectX::XMLoadFloat3(&up);
        DirectX::XMMATRIX View = DirectX::XMMatrixLookAtLH(Eye, Focus, Up);
        DirectX::XMStoreFloat4x4(&rc.view, View);
    }
    // プロジェクション行列
    {
        float fovY = DirectX::XMConvertToRadians(45); // 視野角
        float aspectRatio = graphics.GetScreenWidth() / graphics.GetScreenHeight();
        float nearZ = 0.1f;// カメラが映し出すの最近距離
        float farZ = 1000.0f;  // カメラが映し出すの最遠距離
        DirectX::XMMATRIX Projection = DirectX::XMMatrixPerspectiveFovLH(fovY, aspectRatio,
                                                                         nearZ, farZ);
        DirectX::XMStoreFloat4x4(&rc.projection, Projection);
    }

    ---省略---
}
```

代わりに
こっちを使う

ここのプログラムは
もう必要ないので
削除しましょう

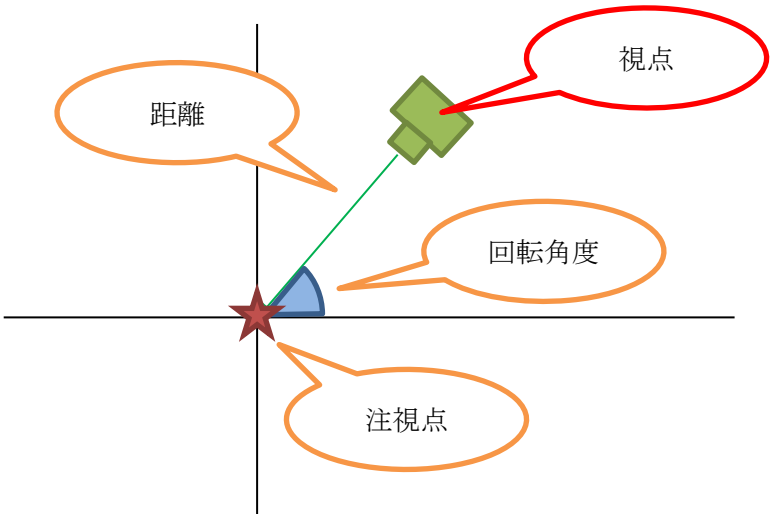
ここまで実装出来たら実行確認をしてみましょう。
実装前と見た目が変わってなければ OK です。



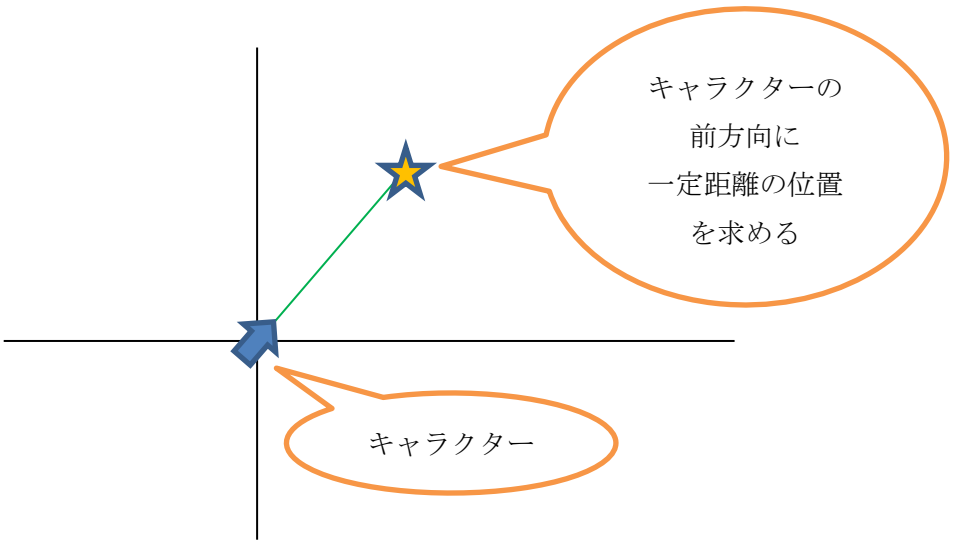
○三人称視点カメラについて

続いて、アクションゲームなどでよくある三人称視点カメラ操作を実装しましょう。
右スティックを操作するとカメラが回転するアレです。

三人称視点カメラは注視点、回転角度、距離の３つです。
この３つの情報からカメラの視点（位置）を計算します。



カメラの視点を算出する方法ですが、キャラクターの行列計算の応用で求めることができます。
キャラクターで例えると、Position が注視点、Angle が回転角度、視点はキャラクターの位置から前方向に一定距離進んだ位置になります。

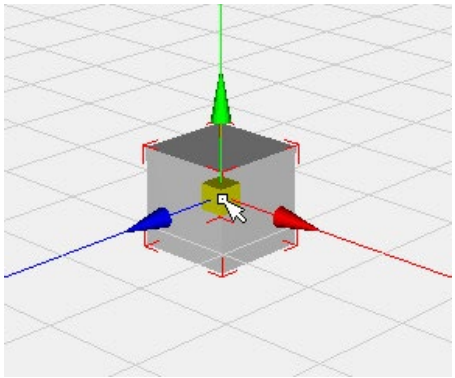


前回、行列はスケール、回転、位置が組み合わさったものとなんとなく学習しました。
今回はもう少し理解を深めましょう。

行列は 4x4 の 16 個の要素で姿勢を表現していますが、細かい内訳は以下のようになっています。
ベクトルとは簡単に言うと矢印です。
ベクトルには方向と大きさが存在します。
姿勢は右方向、上方向、前方向のベクトルと位置があれば表現できます。

$$\begin{bmatrix} Xx & Xy & Xz & 0 \\ Yx & Yy & Yz & 0 \\ Zx & Zy & Zz & 0 \\ Px & Py & Pz & 1 \end{bmatrix}$$

X 軸ベクトル
Y 軸ベクトル
Z 軸ベクトル
位置



この性質を利用すれば視点を求めることができます。

○カメラコントローラークラス

三人称視点カメラを制御するカメラコントローラークラスを作成しましょう。
CameraController.cpp と CameraController.h を作成し、下記プログラムコードを記述しましょう。

CameraController.h

```
#pragma once

#include <DirectXMath.h>

// カメラコントローラー
class CameraController
{
public:
    CameraController() {}
    ~CameraController() {}

    // 更新処理
    void Update(float elapsedTime);

    // ターゲット位置設定
    void SetTarget(const DirectX::XMFLOAT3& target) { this->target = target; }

private:
    DirectX::XMFLOAT3      target = { 0, 0, 0 };
    DirectX::XMFLOAT3      angle = { 0, 0, 0 };
    float                  rollSpeed = DirectX::XMConvertToRadians(90);
    float                  range = 10.0f;
};
```

CameraController.cpp

```
#include "CameraController.h"
#include "Camera.h"
#include "Input/Input.h"

// 更新処理
void CameraController::Update(float elapsedTime)
{
    GamePad& gamePad = Input::Instance().GetGamePad();
    float ax = gamePad.GetAxisRX();
    float ay = gamePad.GetAxisRY();
    // カメラの回転速度
    float speed = rollSpeed * elapsedTime;

    // スティックの入力値に合わせてX軸とY軸を回転
    

    // カメラ回転値を回転行列に変換
    DirectX::XMMATRIX Transform = 

    // 回転行列から前方向ベクトルを取り出す
    DirectX::XMVECTOR Front = Transform.r[2];
    DirectX::XMFLOAT3 front;
    DirectX::XMStoreFloat3(&front, Front);

    // 注視点から後ろベクトル方向に一定距離離れたカメラ視点を求める
    DirectX::XMFLOAT3 eye;
```

行列の 3 行目のデータを
取り出している

```
    eye.x =
    eye.y =
    eye.z =

    // カメラの視点と注視点を設定
    Camera::Instance().SetLookAt(eye, target, DirectX::XMFLOAT3(0, 1, 0));
}
```

ターゲットを中心にカメラを回転させる処理ができました。
このカメラコントローラークラスの SetTarget()関数にプレイヤーの座標を設定することでプレイヤーの位置を中心にカメラを回すことができます。

しかし、現時点ではプレイヤーの位置を取得することができないのでキャラクタークラスから位置を取得できるようにしましょう。
ついでにスケールや回転などのセッター、ゲッターも用意してしまいましょう。

Character.h

```
---省略---

// キャラクター
class Character
{
public:
    ---省略---

    // 位置取得
    const DirectX::XMFLOAT3& GetPosition() const { return position; }

    // 位置設定
    void SetPosition(const DirectX::XMFLOAT3& position) { this->position = position; }

    // 回転取得
    const DirectX::XMFLOAT3& GetAngle() const { return angle; }

    // 回転設定
    void SetAngle(const DirectX::XMFLOAT3& angle) { this->angle = angle; }

    // スケール取得
    const DirectX::XMFLOAT3& GetScale() const { return scale; }

    // スケール取得
    void SetScale(const DirectX::XMFLOAT3& scale) { this->scale = scale; }

    ---省略---
};
```

準備ができたので SceneGame.h と SceneGame.cpp に実装しましょう。

SceneGame.h

```
---省略---
#include "CameraController.h"

// ゲームシーン
class SceneGame
{
public:
    ---省略---

private:
    ---省略---
    CameraController*  cameraController = nullptr;
};
```

SceneGame.cpp

```
---省略---

// 初期化
void SceneGame::Initialize()
{
    ---省略---
    // カメラコントローラー初期化
    
}

// 終了化
void SceneGame::Finalize()
{
    // カメラコントローラー終了化
    
    
    ---省略---
}

// 更新処理
void SceneGame::Update(float elapsedTime)
{
    // カメラコントローラー更新処理
    DirectX::XMFLOAT3 target = player->GetPosition();
    target.y += 0.5f;
    cameraController->SetTarget(target);
    cameraController->Update(elapsedTime);

    ---省略---
}

---省略---
```

プレイヤーの腰あたりを
ターゲットに設定

実行確認をしてみましょう。
右スティックを操作し、カメラが自由に回転出来ていれば OK です。
ゲームパッドの右スティックはキーボードの「I」「J」「K」「L」キーに割り当てられています。

○回転角度制限

現状では問題点が一つあります。
カメラの視点がターゲットの真上になったときにガタついた挙動になります。
これはジンバルロックというオイラー回転制御の弱点だからです。
オイラーで回転制御をする場合はこのジンバルロックを回避する必要があります。
今回は X 軸の最大回転値と最小回転値に制限をつける方法で回避してみましょう。

CameraController.h

```
---省略---

// カメラコントローラー
class CameraController
{
public:
    ---省略---

private:
    ---省略---
    float                   maxAngleX = DirectX::XMConvertToRadians(45);
    float                   minAngleX = DirectX::XMConvertToRadians(-45);
};
```

CameraController.cpp

```
---省略---

// 更新処理
void CameraController::Update(float elapsedTime)
{
    ---省略---

    // スティックの入力値に合わせてX軸とY軸を回転
    ---省略---

    // X軸のカメラ回転を制限
    

    // Y軸の回転値を-3.14～3.14に収まるようにする
```

```
    if (angle.y < -DirectX::XM_PI)
    {
        angle.y += DirectX::XM_2PI;
    }
    if (angle.y > DirectX::XM_PI)
    {
        angle.y -= DirectX::XM_2PI;
    }

    ---省略---
}
```

実装出来たら実行確認をしてみましょう。
右スティック操作で縦方向のカメラ回転が制限できていれば OK です。

お疲れさまでした。