??????????

Timmyの移動できたとしても、操作が確認しにくいので、プレーヤーを追いかけるカメラ（　　三人称視点　　）を作りましょう

????????

一番簡単なのは「カメラの位置を主人公の位置にしろ！」の仕組みである。これをすぐ試せるので、「CameraController」スクリプトを作成してみましょう:

?????

```
// 主人公を追いかけるカメラ
public class CameraController : MonoBehaviour
{
    // 追いかけるオブジェクトのTransform
    [SerializeField]
    private Transform target;

    private void Update()
    {
        // 自分の位置をターゲットの位置にするだけ
        transform.position = target.position;
    }
}
```

このスクリプトをカメラにアタッチし、確認しましょう。なお、　　Unity でカメラの「　　target」を「　Player　」にしてください。

??????????

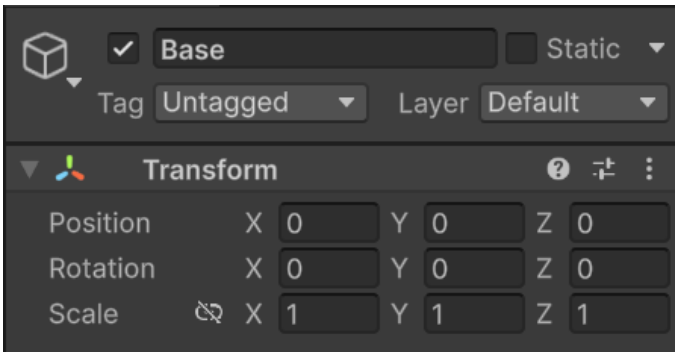
今の処理はとても単純で、カメラはプレーヤーの足元に行ってしまいました。もう少し賢い作戦を考えてみましょう。「三人称」といえば:

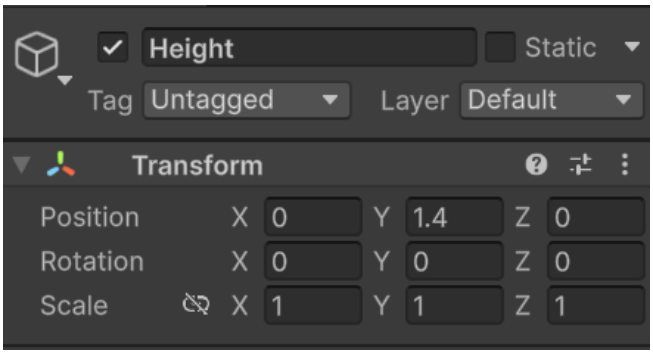
セルフィスティックは良いでしょう! シーンにあるカメラの親子関係を変えて、同じ仕組みになるようにしましょう。具体的に:

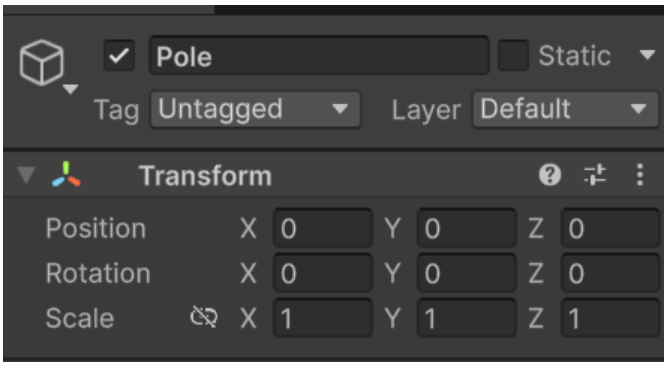
- Base (基礎) : 主人公を追いかける根。セルフィスティックなら、「人間が立っている場所」
- Height (高さ) : 地面からの高さ。だいたい Timmyの頭までにすれば良いでしょう。
- Pole (ポール) : スティックのそのもの。上がるか下がることにより、撮影角度を変えられる
- Camera (カメラ) : スティックの端にあるカメラ。適切な距離にし、 Timmy が全身が見えるようにしましょう。

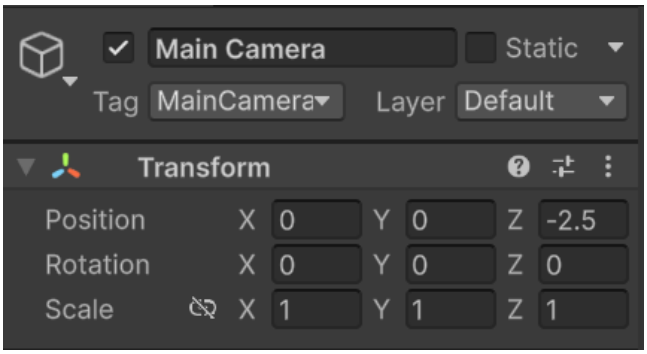
これはヒエラルキー (とその位置) はこうなる:



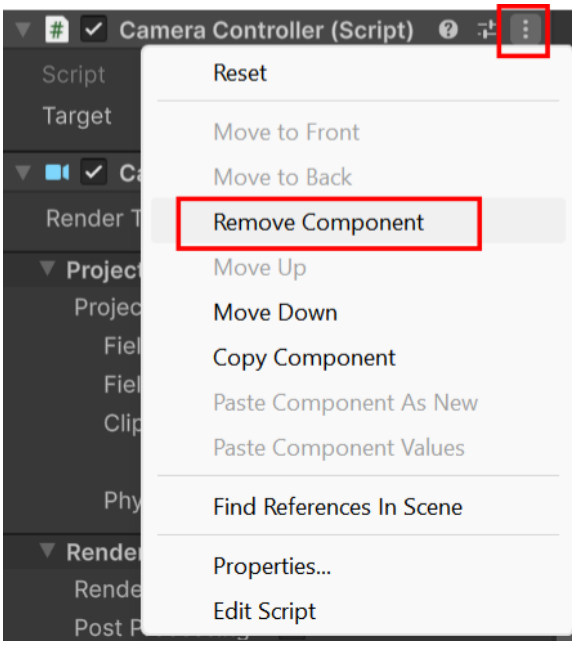








□□□□ CameraController□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Base □□□□□□□□□ Main Camera □□□□□□□□□□□□□□□□□□□□□□□ ⋮□□□□□□ Remove Component□□□□□□□□□□□□□□□□□□□□



□□□□ CameraController□□ Base□□□□□□□□□□□□□□□□□□□□

??????????

カメラは主人公に追いかけても、周りを自由に見たいので、マウスでカメラを回転させたい。今のカメラの作りだと簡単に責任を分けることができる!

- Height オブジェクトを Y軸まわりに回転させると、360度の回転ができる。
- Pole オブジェクトを X軸まわりに回転させると、上下の角度を変えることができる。

IMAGE: Height rotating and Pole pivoting

そして、WASDの入力は `Input.GetAxis("Horizontal");` と `Input.GetAxis("Vertical");` で取得できたが、同じようにカメラの回転 `Input.GetAxis("CamRotY");` とカメラの上下角度 `Input.GetAxis("CamRotX");` を取れれば良いけど、この入力軸が定義されていない。

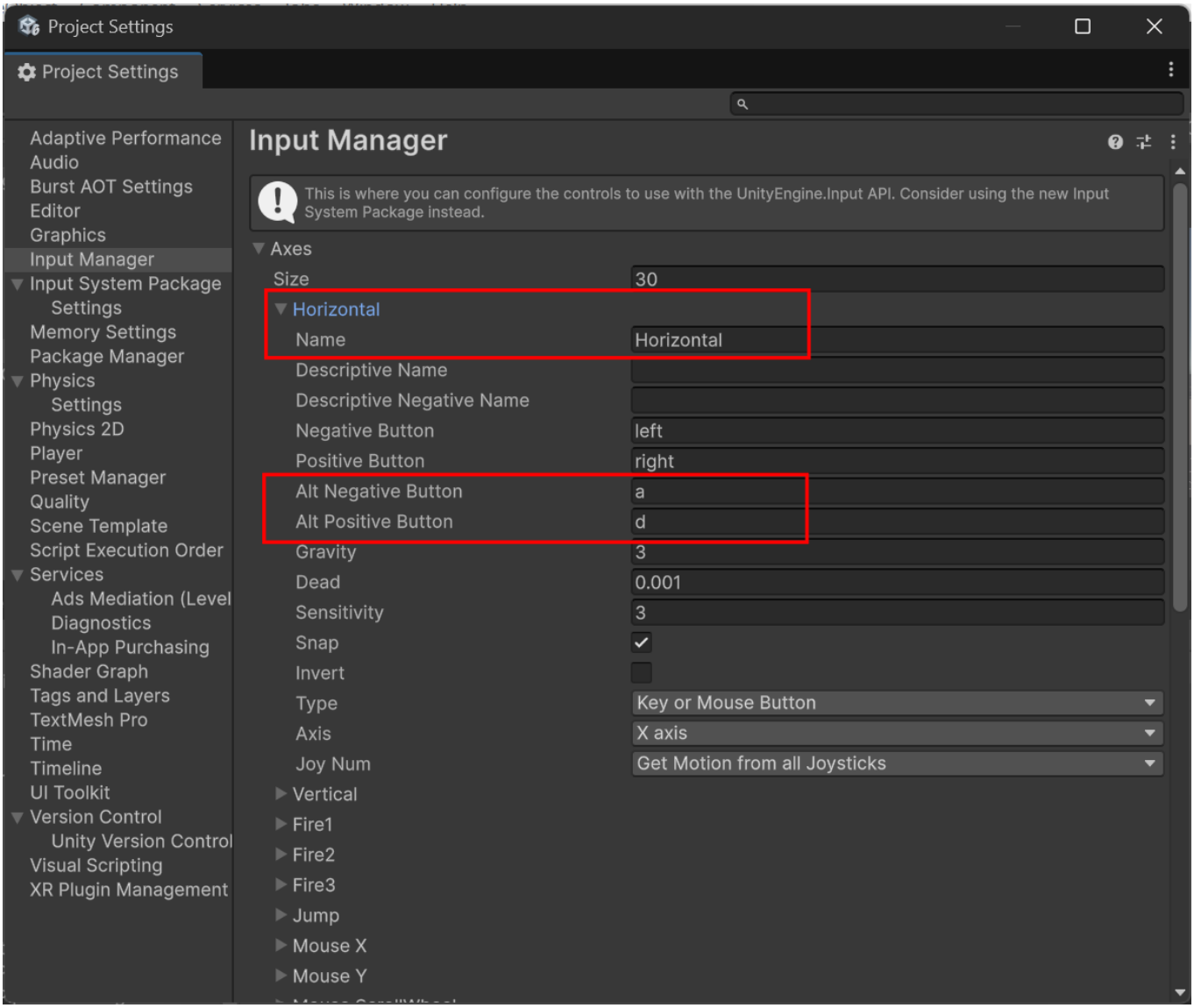
??????????

??????????

※

以下は「入力マネージャー（旧）」に対する話しである。2年生になったら「入力システム（新）」に切り替える予定。

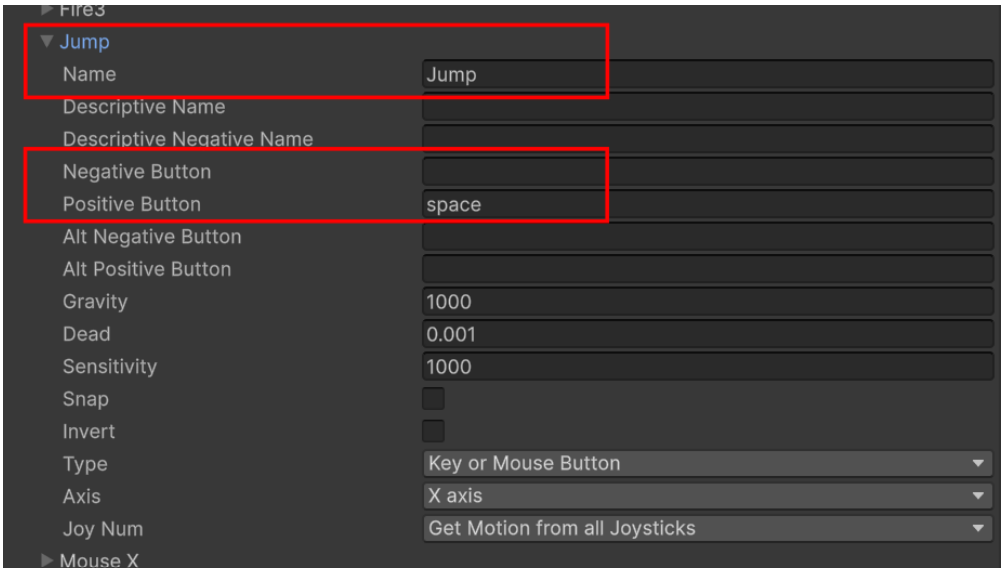
`"Horizontal"`（水平入力）と `"Vertical"`（垂直入力）が魔法の言葉ではなく、プロジェクト設定として定義されている入力である。これをトップメニューの「Edit / Project Settings」（編集 / プロジェクト設定）の中にある「Input Manager」（入力マネージャー）から確認ができる。



ここで「 Horizontal」という入力名は「 Positive」（正の数）と「 Negative」（負の数）があるので、 ‐ １～１の間の「軸」であることを確認できる。つまり、 Negative Buttonの「 a」を押すと、 Input.GetAxis("Horizontal") が -1 を返す、 Positive Button の「 d」を押すと Input.GetAxis("Horizontal") が１を返す。この行動がすでに使用し、 Timmyを動かしました。

その他には、 Fire1～ Fire3 や Jump
の入力名がある。これはデフォルトとして定義されている入力であるが、変えたり、消したり、増やしたり
することが可能である。

「Jump」を確認してみましょう：



この入力は「 Positive」のみで、「 Negative
」がない。つまり、この入力は「軸」ではなく「ボタン」である。ボタンは「 Input.GetButtonXXX("入力名") 」で取得でき、「 true」または「 false」を返す。 GetMouseButtonXXX と同様に３パターンがある：

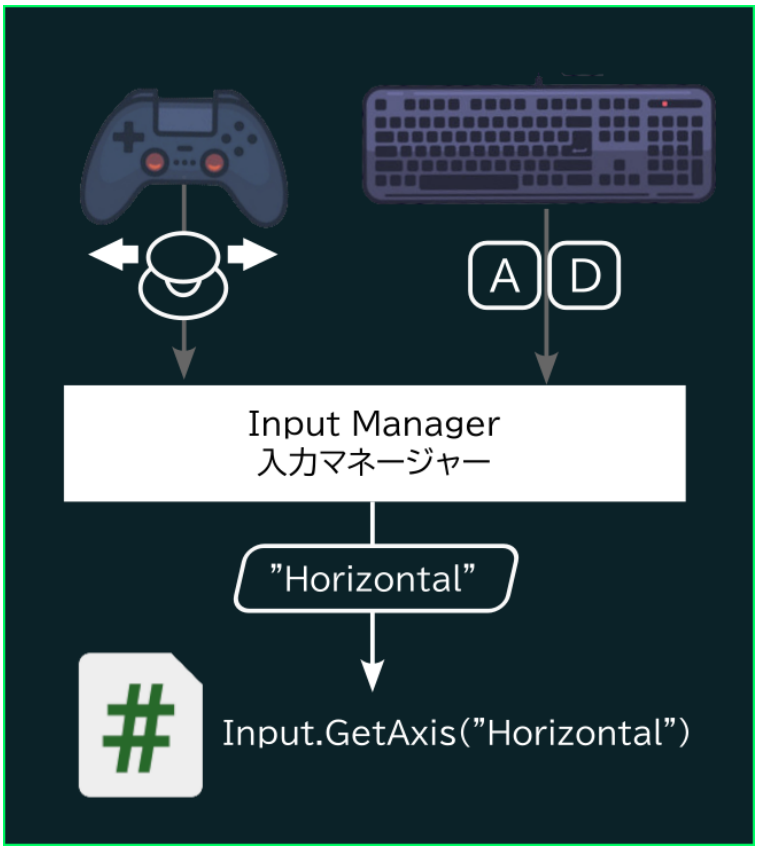
GetButtonDown("入力名") ： 押された瞬間に「 true」を返す。
GetButton("入力名") ：押し続けるときに「 true」を返す。
GetButtonUp("入力名") ：放した瞬間に「 true」を返す。

デフォルト設定としては「 Jump」という入力は「 space」に設置されているので、スペースキーを押したら
GetButtonDown("Jump") は true になる！

もちろん、これらあくまで初期設定なので、キーを変えて良いし、名前も変えても大丈夫（例えば、「 Kick 」「 Punch」「 Attack」など）

もうちょっと見ると、多くの入力は２回が現れる！もう一つの "Horizontal" を確認すると：

あれ？「 Positive」も「 Negative」もない？なにこれ？



これは、 "Horizontal" に対するもう一つの入力処理である。 Typeを詳しくみると、「 Joystick 」というキーワードが書かれている。

そう！これはコントローラーに対応する入力。 A/Dを使えば、 "Horizontal" が発動するし、コントローラーの「 X Axis」を使うと、同じ "Horizontal" が発動する。

つまり、入力マネージャーが入力ハードウェアを抽象化してくれる機能である！

キーワード＆マウス、コントローラーを関係なく、 C#で「 Horizontal」を処理すれば十分。 Unity が裏側で入力を通訳してくれる。

???????????

すでに「 MouseX」と「 MouseY」の入力を定義されているので、これを再利用しましょう。 MouseX はマウスの横の動きなので、これを「 CamRotY」にしましょう。同様に MouseY を「 CamRotX 」に書き換えましょう。

これで、後ほどコントローラーの対応を追加したいなら、同じ名前の新しい入力を追加し、適切なボタンやスティックを割り当てることが可能。

?????

360???

CameraController を更新しましょう。まず、 Y軸まわり（ 360度回転）を実装するには先ほど定義した「CamRotY」の入力を活用し、 Height を回転する。

```
// 主人公を追いかけるカメラ
public class CameraController : MonoBehaviour
{
    // 追いかけるオブジェクトのTransform
    [SerializeField]
    private Transform target;

    // 360度回転に使うTransform
    [SerializeField]
    private Transform height;

    private void Update()
    {
        // 自分の位置をターゲットの位置
        transform.position = target.position;

        // 回転処理
        Rotate360();
    }
```

```
    // 360度の回転する
    private void Rotate360()
    {
        // トランスフォームの回転を取得
        // Quaternion（クオーターニオン・四元数）は回転を表すクラスである
        // Quaternionについては、「ゲーム数学」でまた勉強する
        Quaternion rot = height.localRotation;

        // このまま使えないので、XYZの回転角度（オイラー角度）にする
        Vector3 angles = rot.eulerAngles;

        // Y軸だけを更新
        angles.y += Input.GetAxis("CamRotY");

        // 回転を更新（オイラー角度 → クオーターニオンに戻す）
        height.localRotation = Quaternion.Euler(angles);
    }
}
```

一応、回転するが、速度が多少遅く感じるので、回転速度を「度 / 秒」単位で調整できるようにしましょう：

```
using UnityEngine;

// 主人公を追いかけるカメラ
public class CameraController : MonoBehaviour
{
    // 回転速度（度／秒）
    [SerializeField]
    private float rotSpeed = 10;

    //  （省略）

    // 360度の回転する
    private void Rotate360()
    {
        // トランスフォームの回転を取得
        // Quaternion（クオーターニオン・四元数）は回転を表すクラスである
        // Quaternionについては、「ゲーム数学」でまた勉強する
        Quaternion rot = height.localRotation;
```

```
        // このまま使えないので、XYZの回転角度（オイラー角度）にする
        Vector3 angles = rot.eulerAngles;

        // Y軸だけを更新
        angles.y += Input.GetAxis("CamRotY") * rotSpeed * Time.deltaTime;

        // 回転を更新（オイラー角度 → クオーターニオンに戻す）
        height.localRotation = Quaternion.Euler(angles);
    }
}
```

????

これで主人公まわり 360度の回転が実現できたが、今度、カメラの上下角度を変えたい。このため、「 Pole 」を X 軸まわりに回転すれば良いでしょう。 `Input.GetAxis("CamRotX");` を使用し、実装しましょう:

```
using UnityEngine;

// 主人公を追いかけるカメラ
public class CameraController : MonoBehaviour
{
    // 上下角度を変えるために使うTransform
    [SerializeField]
    private Transform pole;

    // （省略）

    private void Update()
    {
        // 自分の位置をターゲットの位置
        transform.position = target.position;

        // 回転処理
        Rotate360();
        RotateUpDown();
    }

    // 360度の回転する
    private void Rotate360()// （省略）
```

```
    // 上下の回転
    private void RotateUpDown()
    {
        // ポールの回転を取得し、X軸まわりだけ回転する
        Vector3 angles = pole.localRotation.eulerAngles;
        angles.x -= Input.GetAxis("CamRotX") * rotSpeed * Time.deltaTime;
        pole.localRotation = Quaternion.Euler(angles);
    }
}
```

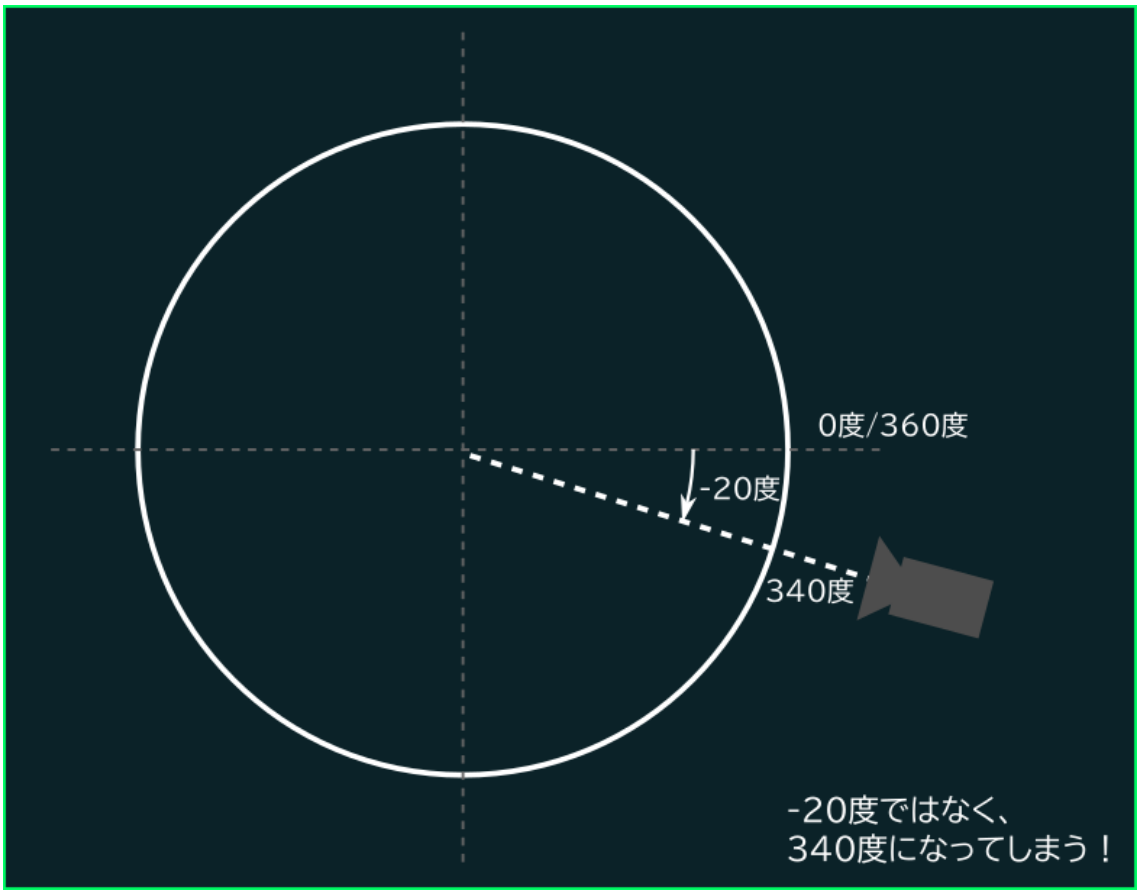
これで上下の回転できたが、カメラが逆さまになるのが望ましくないので、「最低角度」と「最大角度」を設定しましょう。これは自由に変わらないので、定数（`const`）として宣言しても良い。

```
// 上下の回転
private void RotateUpDown()
{
    //□最低と最大角度
    const float MinAngle = -30f;
    const float MaxAngle = 60f;

    // ポールの回転を取得し、X軸まわりだけ回転する
    Vector3 angles = pole.localRotation.eulerAngles;
    angles.x -= Input.GetAxis("CamRotX") * rotSpeed * Time.deltaTime;

    // 制限
    angles.x = Mathf.Clamp(angles.x, MinAngle, MaxAngle);
    pole.localRotation = Quaternion.Euler(angles);
}
```

????

あれ? 上はちゃんと制限されているが、下の方に行くと、また上に飛んでしまう?

一旦制限を取り消し、`Debug.Log` で角度を確認してみると「 - 角度」ではなく、回転一周まわって、また360度から始まってしまう! これが、`Quaternion` の内部的な処理であり、角度が0以下になると、360度に変わり、360は `MaxAngle` を超えるので、`Mathf.Clamp` で最大角度に飛んでしまう!



解決には、 `localRotation` の角度を使用せず、内部の変数を使えば良いでしょう:

```
// 上下回転の角度
private float angleX = 0;

// (省略)

// 上下の回転
private void RotateUpDown()
{
    //□最低と最大角度
    const float MinAngle = -30f;
    const float MaxAngle = 60f;

    // 角度を更新し、制限
    angleX -= Input.GetAxis("CamRotX") * rotSpeed * Time.deltaTime;
    angleX = Mathf.Clamp(angleX, MinAngle, MaxAngle);

    // ポールの回転を取得し、X軸まわりだけ回転する
    pole.localRotation = Quaternion.Euler(angleX, 0, 0);
}
```

Revision #8
Created 2026-05-04 08:08:39 UTC by Menendez Francisco
Updated 2026-05-25 05:09:37 UTC by Menendez Francisco