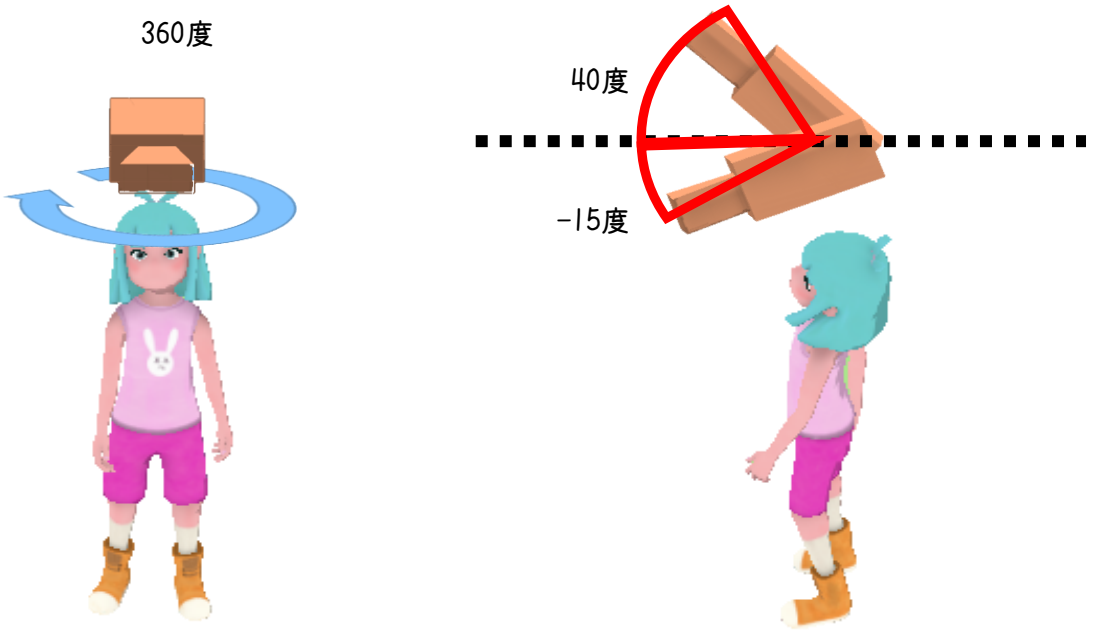
カメラ操作

今回のカメラ要件は、

- Y軸に、360度回転すること
- X軸に、上は40度、下は15度回転すること
- カメラ操作は矢印キーを用いること



カメラをユーザ操作で、どれくらい回転させたか、情報を保持しておきます。

```
// カメラ角度(rad)
VECTOR angles_;                    xとyを使用し、zは使用しない。
```

x軸においては、保持している角度が上限下限を超えないように
条件を作ること。

angles_ に回転量を加算していけば、SyncFollow 関数で、
注視点やカメラ位置を計算していますので、カメラが回るようになっています。

SyncFollow 関数の処理を解説していきます。

```
void Camera::SyncFollow(void)
{

    auto& gIns = GravityManager::GetInstance();

    // 同期先の位置
    VECTOR pos = followTransform_->pos;

    // 重力の方向制御に従う
    Quaternion gRot = gIns.GetTransform().quaRot;

    // 正面から設定されたY軸分、回転させる
    rotOutX_ = gRot.Mult(
        Quaternion::AngleAxis(angles_.y, AsoUtility::AXIS_Y));

    // 正面から設定されたX軸分、回転させる
    rot_ = rotOutX_.Mult(
        Quaternion::AngleAxis(angles_.x, AsoUtility::AXIS_X));

    VECTOR localPos;

    // 注視点(通常重力でいうところのY値を追従対象と同じにする)
    localPos = rotOutX_.PosAxis(LOCAL_F2T_POS);
    targetPos_ = VAdd(pos, localPos);

    // カメラ位置
    localPos = rot_.PosAxis(LOCAL_F2C_POS);
    pos_ = VAdd(pos, localPos);

    // カメラの上方向
    cameraUp_ = gRot.GetUp();

}
```

① // 同期先の位置
```
VECTOR pos = followTransform_->pos;
```

同期先の位置(座標)です。今回はプレイヤーになります。

② // 重力の方向制御に従う
```
Quaternion gRot = gIns.GetTransform().quaRot;
```

重力マネージャーから、回転情報を取得してきます。
これは、どこを正面としているか把握するために使用します。
初期時点では、Zの正方向を向いており、
重力方向が下を向き続けている間は、この方向が変わることはありませんが、
惑星(ステージ)が変わり、重力方向が変わると、正面の向きも変わります。

■GravityManagerの役割

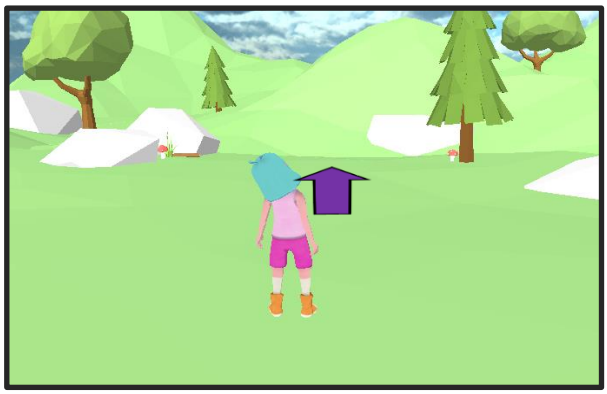
プレイヤーやカメラの回転の元となる親玉の回転制御です。
回転の親子関係でいくと、下記のような関係にあります。

World	※ ワールドの正面 { 0.0f, 0.0f, 1.0f } これは通常変わらない
GravityManager	※ 初期の正面 { 0.0f, 0.0f, 1.0f } 最初は、ワールドと同じ
Player	
Camera	

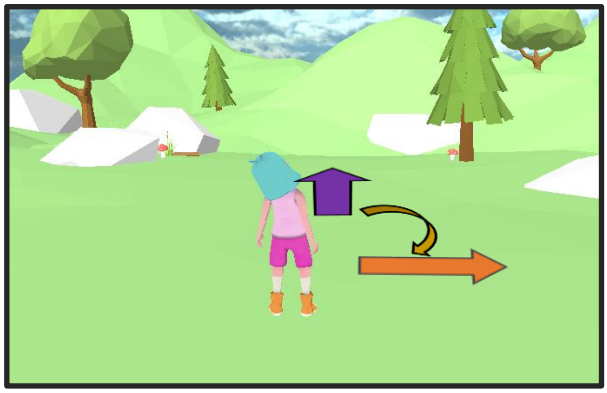
GravityManager が回転すると、
その子供の Player や Camera も回転するという仕組みです。
但し、位置や大きさまで親子関係を持つと、おかしくなりますので、
あくまで回転情報のみ親子関係を持ちます。

GravityManager が無回転の時のイメージ

通常のワールドと何も変わらない。



正面方向から。。。

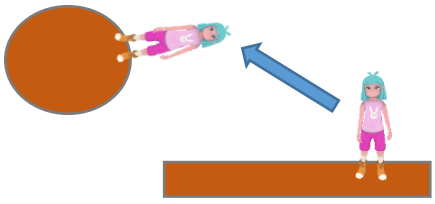


カメラの angles_ 分、
回転させた視点となる。

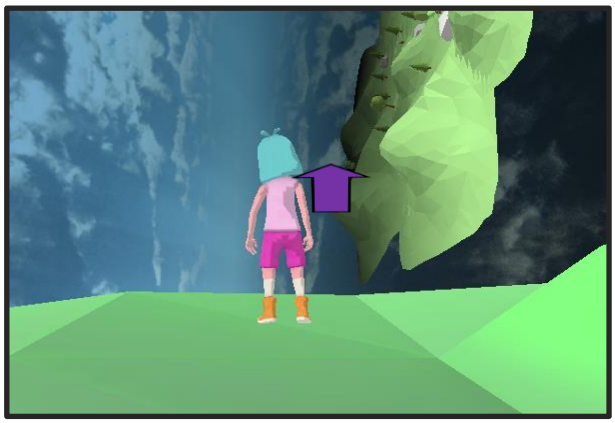
仮にY軸にプラス90度
だと、オレンジ矢印の
方向にカメラが向く。

GravityManager が回転している時のイメージ

球体上の惑星重力に降り立った時。

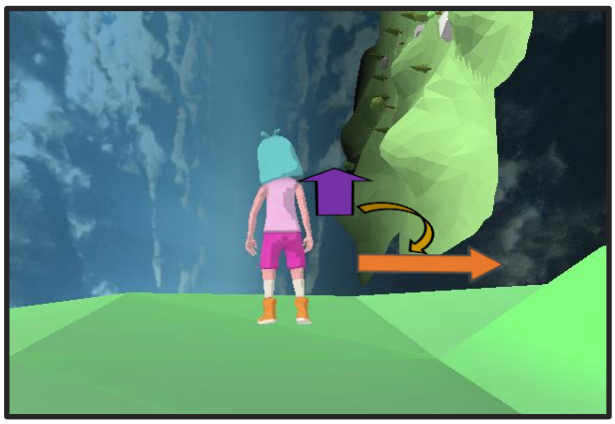


← 通常のステージ



見て通り、
元のステージから
すると、かなり回転した
位置にいます。

この時のGravityManager
の正面は紫矢印になっている



同じく、
仮にY軸にプラス90度
だと、オレンジ矢印の
方向にカメラが向く。

③ // 正面から設定されたY軸分、回転させる
```
rotOutX_ = gRot.Mult(
    Quaternion::AngleAxis(angles_.y, AsoUtility::AXIS_Y));
```

Y軸へのカメラ操作を angles_.y（正面方向からの角度）に保存しており、
それを使用して、Quaternion に変換しています。

X軸へのカメラ操作を省いているのは、
キャラクターの移動方向をこのY軸のみ回転で決めるからです。
(カメラが上を向いているからといって、キャラは空を飛んだりしない)

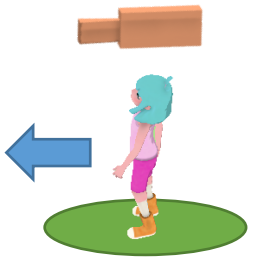
キャラクターの移動方向は、カメラ基準ですが、
地面を歩いて移動するタイプのゲームでは、カメラが持つX軸Y軸の回転を
分けて制御すると、制作しやすいでしょう。

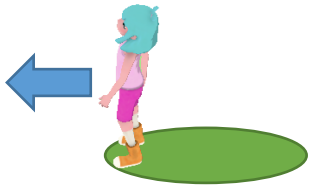
④ // 正面から設定されたX軸分、回転させる
```
rot_ = rotOutX_.Mult(
    Quaternion::AngleAxis(angles_.x, AsoUtility::AXIS_X));
```

X軸Y軸を合成した rot_ をカメラ自身の回転情報として、
カメラ位置と注視点の位置をいつも通りの相対座標を回転させるやり方で
計算していきます。

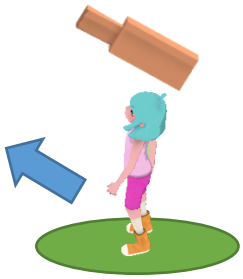
■カメラのX軸、Y軸を分けないと困る例

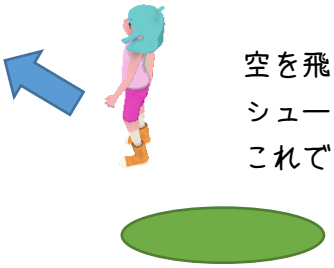
X軸なし





X軸あり

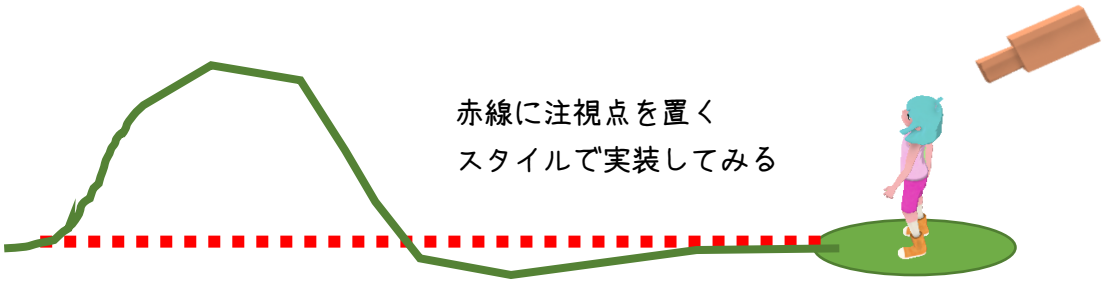




空を飛んでしまう
シューティングは
これで良かった

⑤ // 注視点(通常重力でいうところのY値を追従対象と同じにする)
```
localPos = rotOutX_.PosAxis(LOCAL_F2T_POS);
targetPos_ = VAdd(pos, localPos);
```

注視点を置きたい場所にもよりますが、
注視点の高さをキャラクターの位置の高さと同じにしたければ、
X軸回転が抜かれている rotOutX_ を使用します。

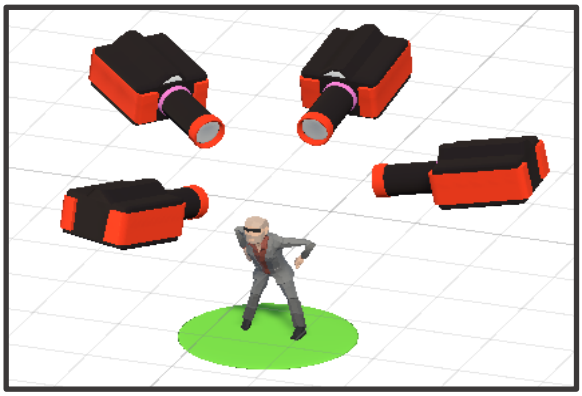


⑥ // カメラ位置

```
localPos = rot_.PosAxis(LOCAL_F2C_POS);
pos_ = VAdd(pos, localPos);
```

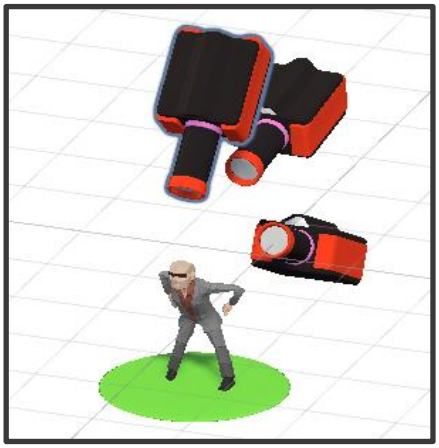
カメラ位置も、キャラクターの位置を基準に決めていきます。
X軸もY軸も使用していくため、rot_ を使用します。

Y軸回転した時の
カメラ位置の変化



X軸回転した時の
カメラ位置の変化

X軸、Y軸ともに、
円軌道で位置変化する



⑦ // カメラの上方向

```
cameraUp_ = gRot.GetUp();
```

重力制御の上方向(正面の上方向)をカメラの上方向とします。

その他、大切なカメラ制御

現段階でのカメラ機能だけだと、カメラ位置が地面にめり込んだりして、
見栄えが悪くなってしまいます。

ユーザのカメラ操作によるめり込みであれば、
注視点から、カメラ位置へ衝突判定を行って、地面などに衝突したら、
カメラ角度を変更しないようにしたり、

衝突地点から、少し注視点側にカメラ位置を強制移動させたり、

衝突したメッシュを半透明にしたり、

色々な制御方法があります。

これらの単発的な実装だけであれば、割と簡単にできるのですが、
単発だけだと、なかなか全ては上手くいかず、
複合的な実装になると思いましたので、見送りとさせて頂きました。

演習①　　カメラの要件に沿って、カメラ操作を実装してください

- Y軸に、360度回転すること
- X軸に、上は40度、下は15度回転すること
- カメラ操作は矢印キーを用いること