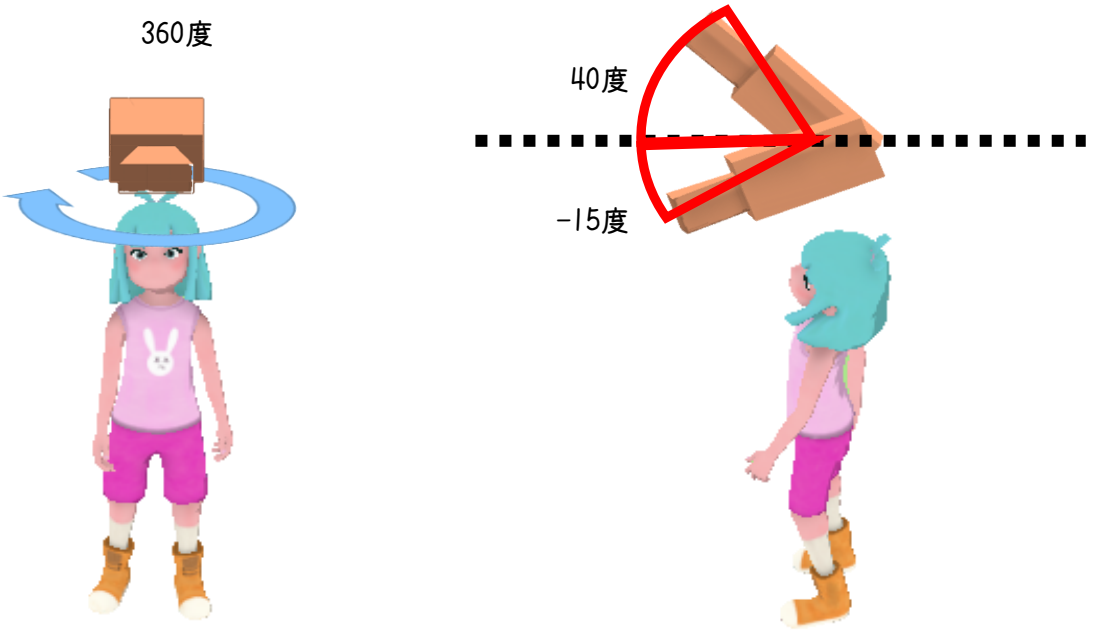
カメラ操作

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 mAngles;                    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。

mAnglesに回転量を加算していけば、SyncTransform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。

SyncTransform関数の処理を解説していきます。

```
void Camera::SyncTransform(void)
{

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

    // 重力の方向制御に従う
    Quaternion gRot = mGravityManager->GetTransform()->quaRot;

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

    // 正面から設定されたX軸分、回転させる
    mQuaRot = mQuaRotOutX.Mult(Quaternion::AngleAxis(mAngles.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 = mQuaRotOutX.PosAxis(RELATIVE_TARGET_POS);
    mTargetPos = VAdd(pos, localPos);

    // カメラ位置
    localPos = mQuaRot.PosAxis(RELATIVE_CAMERA_POS_FOLLOW);
    mPos = VAdd(pos, localPos);

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

}
```

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

同期先の位置(座標)です。今回はプレイヤーになります。

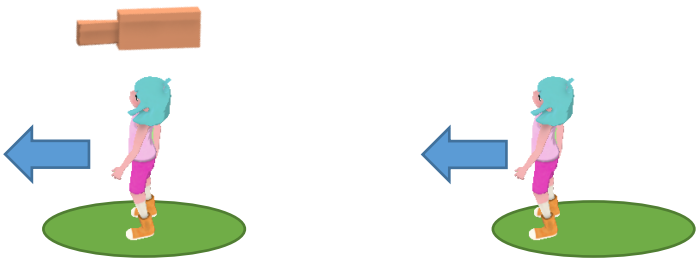
② // 重力の方向制御に従う
```
Quaternion gRot = mGravityManager->GetTransform()->quaRot;
```

重力制御マネージャーから、どこを正面に向いているのか把握するために、
回転情報を取得してきます。
初期時点では、Zの正方向を向いています。
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。

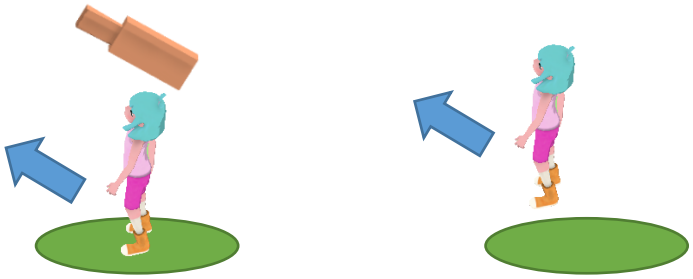
③ // 正面から設定されたY軸分、回転させる
```
mQuaRotOutX = gRot.Mult(Quaternion::AngleAxis(mAngles.y, AsoUtility::AXIS_Y));
```

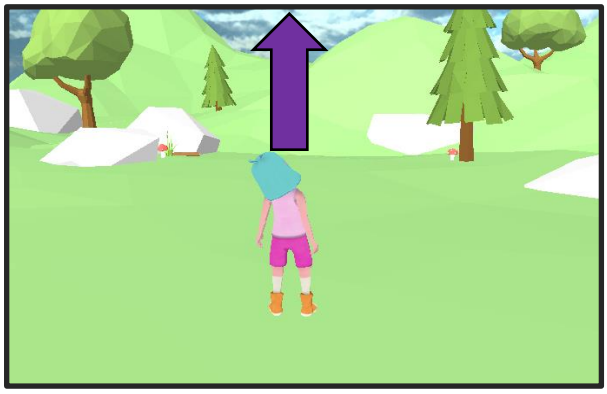
ユーザのY軸へのカメラ操作を正面方向から加算して、回転を求めます。
mQuaRotOutXというメンバ変数に代入し、保持しているのは、
キャラクターの移動方向をこのY軸のみ回転で決めるからです。
仮にX軸の回転が入った回転情報を使って、移動方向を決めてしまうと、
カメラの向きによっては、空中を歩き出したり、地中に進み出したりするので、
X軸回転を抜いた回転情報を保持しておきます。

X軸なし

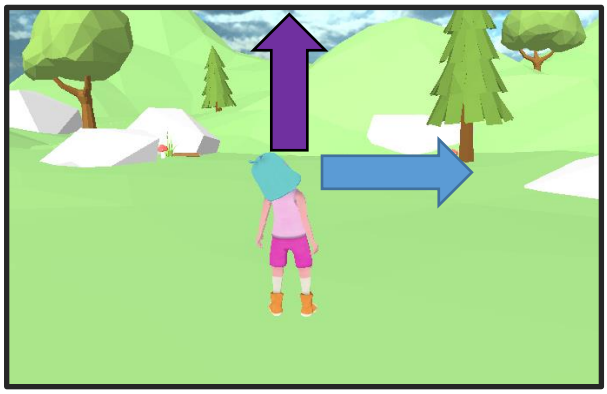


X軸あり



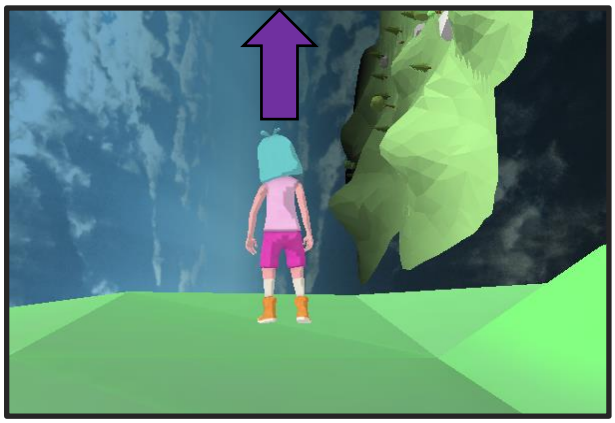


正面方向から。。。

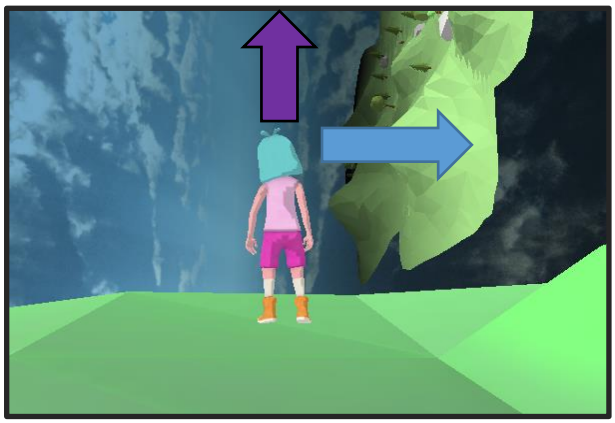


mAngles分、回転させる。

左図は、Y軸にプラス90度。



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
いますが、仮にGravityManagerの
正面方向が紫矢印だとすると、



同じく、
mAngles分、回転させる。

左図は、Y軸にプラス90度。

この紫の線、GravityManagerの回転情報(mTransform.quaRot)の
作り方については、別資料で解説します。

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

ユーザのX軸へのカメラ操作を更に回転に合成させると、
カメラの向きが決定します。

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 = mQuaRotOutX.PosAxis(RELATIVE_TARGET_POS);
mTargetPos = 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mQuaRotOutXを使用します。

⑥ // カメラ位置
```
localPos = mQuaRot.PosAxis(RELATIVE_CAMERA_POS_FOLLOW);
mPos = 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軸回転分、カメラ位置を動かす必要があるため、
mQuaRotを使用します。

⑦ // カメラの上方向
```
mCameraUp = 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。

カメラが背景や３Ｄモデルにめり込んでしまった！

対応方法については、省略させて頂きます。。。
(時間が足らず、すみません)

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、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。