Aide technique DOF – Heddoko

## Extract angles

Sur un transform (ou gameobject.transform), je récupère la rotation locale et la projette sur les 3 axes pour connaitre la rotation indépendante sur chaque axe. L’idée est que les angles d’euler s’enchainent, et n’ont pas une unique représentation d’une même rotation.

public static void ExtractAngles(Quaternion a\_QuatLocal, ref *Vector3* a\_Angles)

    {

*Vector3* localForward = a\_QuatLocal \* *Vector3*.*forward*;

*Vector3* localRight = a\_QuatLocal \* *Vector3*.*right*;

*Vector3* localUp = a\_QuatLocal \* *Vector3*.*up*;

*Vector3* upProj = *Vector3*.*ProjectOnPlane*(localUp, *Vector3*.*right*);         *Vector3* forwardProj = *Vector3*.*ProjectOnPlane*(localForward, *Vector3*.*up*);

*Vector3* rightProj = *Vector3*.*ProjectOnPlane*(localRight, *Vector3*.*forward*);

        upProj.*Normalize*();

        forwardProj.*Normalize*();

        rightProj.*Normalize*();

        a\_Angles.*x* = *Vector3*.*Angle*(*Vector3*.*up*, upProj);

        a\_Angles.*y* = *Vector3*.*Angle*(*Vector3*.*forward*, forwardProj);

        a\_Angles.*z* = *Vector3*.*Angle*(*Vector3*.*right*, rightProj);

    }

## Clamp angles

Après avoir récupérer l’angle indépendant sur les trois axes, on peut calculer l’offset à appliquer pour corriger une rotation

private *Vector3* ClampAngle(*Vector3* a\_ProjAngles)

    {

*Vector3* offset = *Vector3*.*zero*;

        if      (a\_ProjAngles.x > XminMax.*y*)

                       offset.*x* = XminMax.*y* - a\_ProjAngles.x;

        else if (a\_ProjAngles.x < XminMax.*x*)

                       offset.*x* = XminMax.*x* - a\_ProjAngles.x;

        else           offset.*x* = 0;

        if      (a\_ProjAngles.y > YminMax.*y*)

                       offset.*y* = YminMax.*y* - a\_ProjAngles.y;

        else if (a\_ProjAngles.y < YminMax.*x*)

                       offset.*y* = YminMax.*x* - a\_ProjAngles.y;

        else           offset.*y* = 0;

        if      (a\_ProjAngles.z > ZminMax.*y*)

                       offset.*z* = ZminMax.*y* - a\_ProjAngles.z;

        else if (a\_ProjAngles.z < ZminMax.*x*)

                       offset.*z* = ZminMax.*x* - a\_ProjAngles.z;

        else           offset.*z* = 0;

        return offset;

    }

## Reconstruct

Après avoir calculé un offset sur chacun des axes, il faudrait pouvoir recalculer le quaternion représentatif, mais je suis un peu bloqué. Je n’ai peut-être simplement pas testé correctement les solutions possibles :

### Euler

Calcul faux mathématiquement :

Quaternion tRet = Quaternion.Euler(a\_ProjAngles);

### Vectors

Ici je perds la rotation autour de l’axe Z, soit le roll. Quelle suite de cross product effectuer pour orienter mon repère dans le bon sens ?

public static Quaternion ComputeQuaternion(*Vector3* a\_WantedAngles)

{

*Vector3* localForward = *Vector3*.*forward*;

*Vector3* localRight = *Vector3*.*right*;

*Vector3* localUp = *Vector3*.*up*;

        localUp.y = *Mathf*.*Cos*(a\_WantedAngles.*x* \* *Mathf*.*Deg2Rad*);

        localUp.z = *Mathf*.*Sin*(a\_WantedAngles.*x* \* *Mathf*.*Deg2Rad*);

        localForward.x = *Mathf*.*Sin*(a\_WantedAngles.*y* \* *Mathf*.*Deg2Rad*);

        localForward.z = *Mathf*.*Cos*(a\_WantedAngles.*y* \* *Mathf*.*Deg2Rad*);

        localRight.x = *Mathf*.*Cos*(a\_WantedAngles.*z* \* *Mathf*.*Deg2Rad*);

        localRight.y = *Mathf*.*Sin*(a\_WantedAngles.*z* \* *Mathf*.*Deg2Rad*);

*Vector3* tt = *Vector3*.*Cross*(localUp, localForward);

// ici je perds la rotation autour de Z = forward axis = roll

        Quaternion tret = Quaternion.LookRotation(localForward, localUp);

return tret;

}

### Autre solution ?

Si une autre solution est possible, je suis preneur ☺  
pour ce matin, en essayant d’expliquer mon problème, je vais me (re)pencher sur la solution des vecteurs et du cross product