Unityはじめるよ

～IKを使って
アニメーションをコントロール～

統合開発環境を内蔵したゲームエンジン
http://japan.unity3d.com/

※いろんな職業の方が見る資料なので説明を簡単にしてある部分があります。正確には本来の意味と違いますが上記理由のためです。ご了承ください。
この資料内の一部の画像、一部の文章はUnity公式サイトから引用しています。

体の一部を別アニメーションに

今回も、アニメーション中のキャラの一部を
別のアニメーションと合成する方法を紹介します。

・スクリプトを使う方法（前々回やったよ）
・Animatorを使う方法（前回やったよ）
・IK（インバースキネマティクス）を使う方法

IKを使う方法

IK(インバースキネマティクス)とは？

子ボーンの位置から親ボーンの位置を決める方法。

IKの逆がFK(フォワードキネマティクス)で、
親ボーンから子ボーンの位置を決めていく方法。

IK
https://docs.unity3d.com/jp/current/Manual/InverseKinematics.html

IKで何ができるかというと

・階段登る時に足の位置がしっかり段差に乗る
・ハシゴを登る時に両手両足ともしっかりハシゴを掴む
・坂道の角度に合わせて自然に足首の向きを変えられる

など、
アニメーションクリップに地形等の影響を与える
ことができます。

制限

IKが使えるアニメーションタイプはHumanoidに限られる。
つまり一般的な人型のボーン構成を持ったキャラとなる。

ちちもはボーン構成が違うので今回はお休みです。

今回は、

壁に近づいたら手すりを掴み
その状態で歩く

ということをやってみます。

手順

1.Animatorのセットアップ
2.壁と手すりの準備
3.スクリプトの準備

と、その前に、
Mecanimのおさらい

※今回の内容はあまりMecanimが重要なわけではないけど

Mecanimなんてちょろいぜって人は
１２ページまで進んでください

詳しく知りたい人は、
↓ ↓を見てね

http://monolizm.com/sab/pdf/第19回_プレゼン資料(Unityはじめるよ～アニメーター・モーションブレンド～).pdf

用語の説明

- **Mecanim(メカニム)とは**
  「歩く」「走る」「ジャンプする」などのアニメーションクリップを
  うまいこと管理してくれる機能の総称

- **Animation(アニメーションクリップ)とは**
  「歩く」「走る」などのアニメーションが入ったファイル

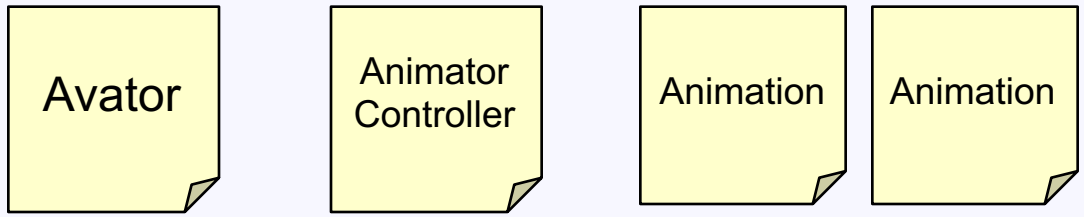
- **AnimatorController(アニメーターコントローラ)とは**
  各ステート（アニメーションクリップに「走る」などの名前をつけたもの
  どのようにつながっているのか、
  ステート間の遷移のさせ方、条件などを管理する機能

- **Avatar(アバター)とは**
  Unity側のボーン構造と
  モデルファイルのボーンをマッピング（紐付け）したもの

- **Animator(アニメーター)とは**
  AvatarとAnimatorControllerを紐付けるコンポーネント

関係図

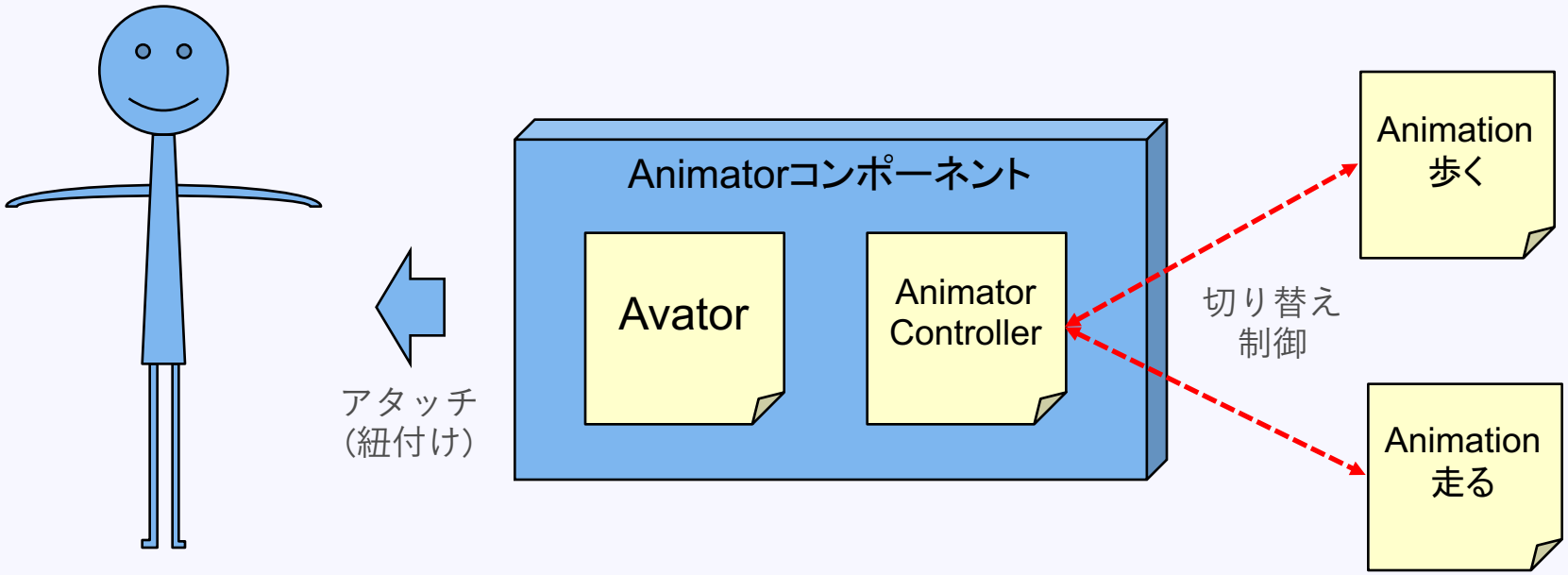
用意しておくファイル



モデルデータを
Unityに取り込む時に
作る

Unityエディタ上で
作る

主に
モデリングソフトで
作る



キャラクターなどの
GameObject

アタッチ
(紐付け)

切り替え
制御

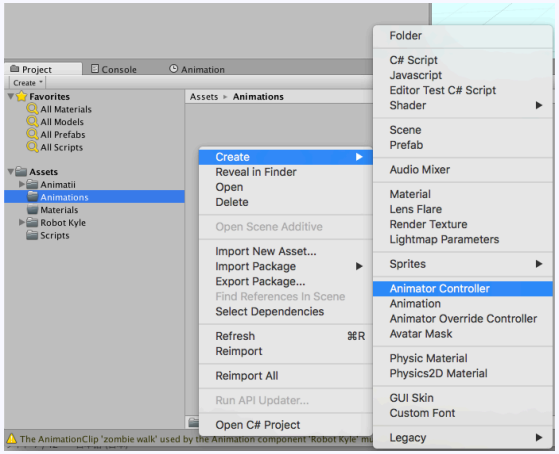
それでは本題

IKを試してみよう！

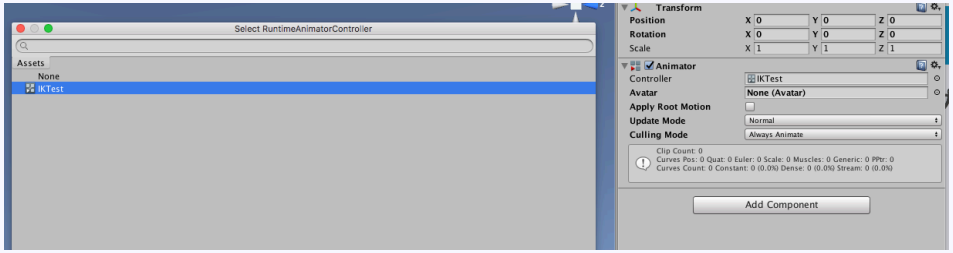
Animatorのセットアップ

まずはロボットが歩く・止まるのアニメーションができるようにします。

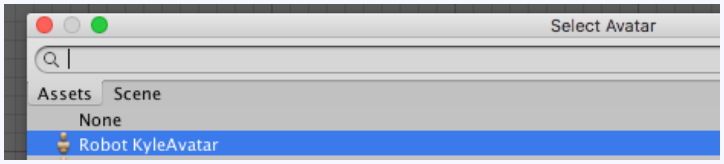
1.アニメーションを管理するコントローラの作成
プロジェクトビューのAnimationsフォルダの中で、
右クリック ＞ Create ＞ Animator Controller
名前は**「IKTest」**



2.ヒエラルキービューで**「Robot Kyle」**を選択し、
インスペクタビューのControllerに上で作った**「IKTest」**をセット

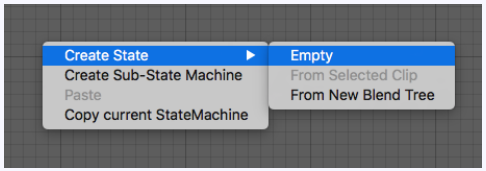


3.その下のAvatorには**「Robot KyleAvatar」**をセット

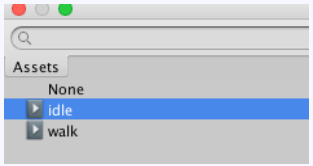


4.次はステートの作成
Animatorウィンドウを開く
上部メニュー ＞ Window ＞ Animator

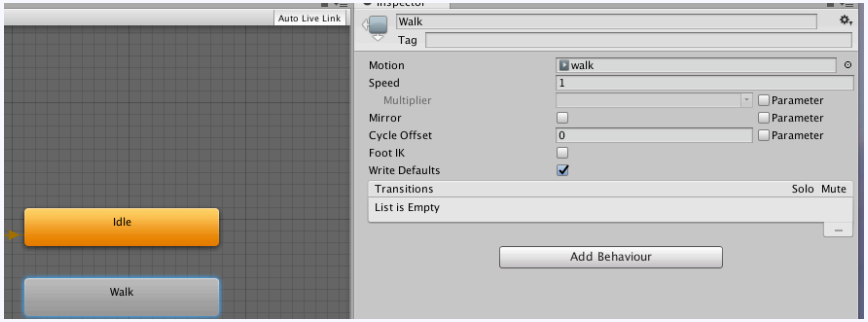
5.走るアニメーションをセットする
Animatorウィンドウ内で
右クリックし ＞ CreateState ＞ Empty
でステートを作りインスペクタビューから名前を**「Idle」**にする。



6.そのままインスペクタビューで**「Motion」**を**「idle」**にする

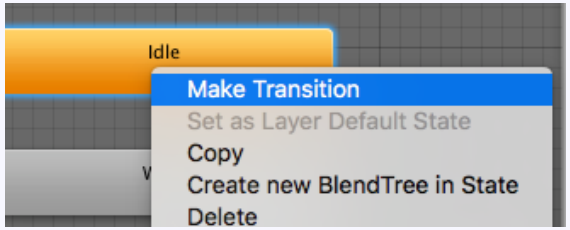


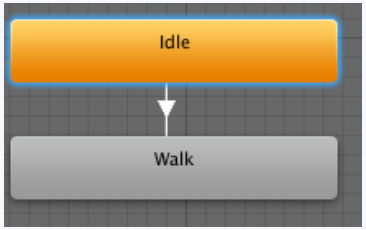
7.同様手順(5,6)で**Walk**を作る



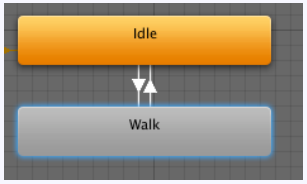
8.ステートを繋ぐトランジションの作成

ステート「Idle」を右クリック ＞ MakeTransition
そのままマウスを動かしてステート「Walk」の上で左クリック





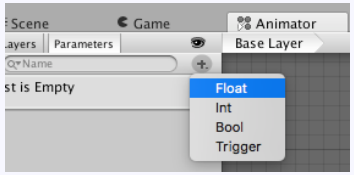
9.同様の手順で「**Walk**」から「**Idle**」のトランジションも作成

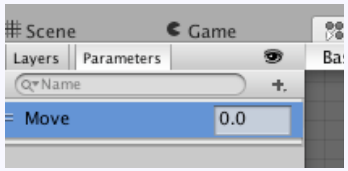


10.ステートを切り替えるパラメータの作成

Parametersタブの下の＋ボタンをクリックしFloatを選択

名前を**「Move」**にする。





11.トランジションとパラメータの紐付け

ステート「Idle」から「Walk」に伸びる矢印をクリックし、

インスペクタビューで**「HasExitTime」**を外す。

（アニメーションが完了しなくてもステートを切り替えられるようになる）

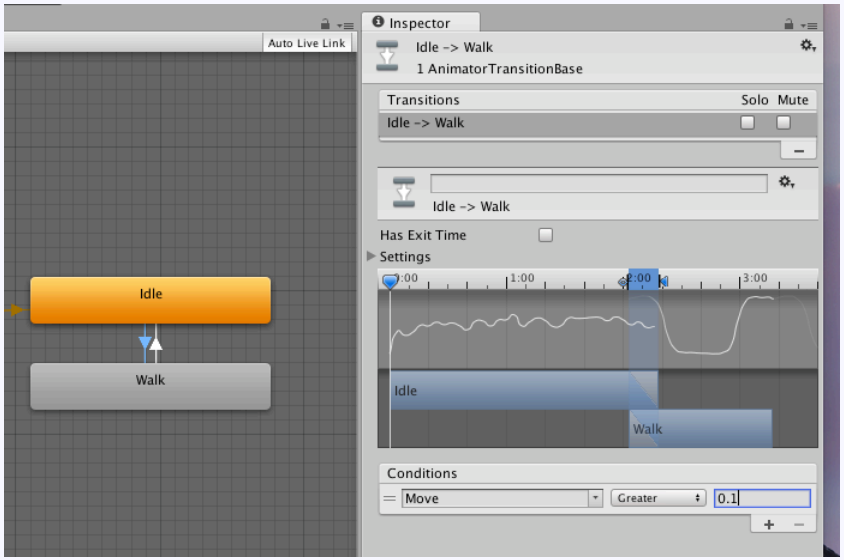
Conditionsの＋ボタンを押し、

「Move」を選んで**「Greater 0.1」**とする。

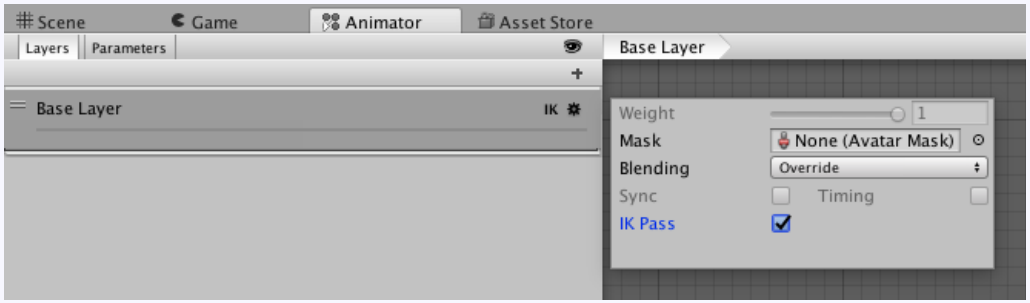
（パラメータ「Move」が0.1以上になったら
ステートを「Walk」に切り替えるということ）

12.同様の手順で「Walk」から「Idle」も。

こちらは**「Less 0.1」**とする。

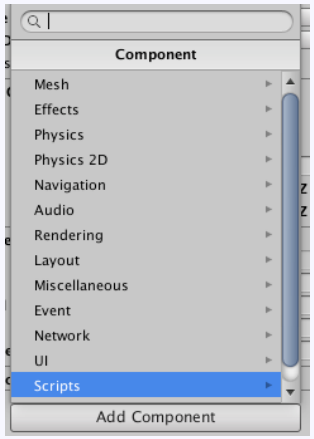


13.やっとIKのセットアップにたどり着きました。
Layersタブを選択し「**Base Layer**」の歯車をクリックして設定を開き
「**IK Pass**」にチェックを入れる



以上。
IKの部分はこれだけです。

14.ちゃんとアニメーションが動くか確認する。
パラメータを切り替えるスクリプトは用意してあるので、
RobotKyleにアタッチしよう
ヒエラルキービューで**「RobotKyle」**を選択し、
一番下の「Add Component」ボタン ＞ Scripts > PlayerController



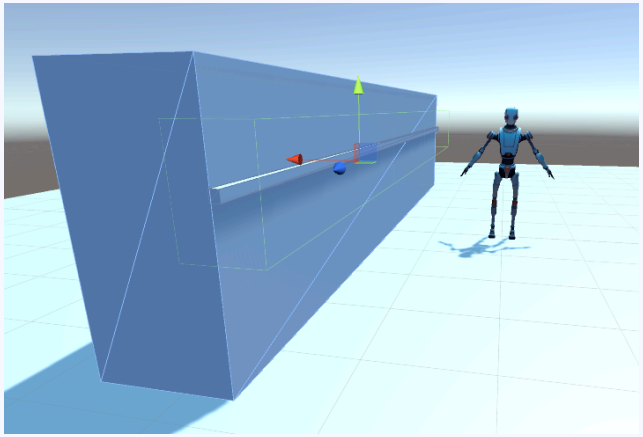
実行して矢印キーでアニメーションが変わるかチェーック！

一応スクリプト解説。パラメータのスクリプトを変える処理は、
AnimatorのSetFloatメソッドを呼ぶだけです。

Animatorが入った変数.SetFloat (パラメータ名, 値);

壁と手すりの準備

ロボットが手を置く手すりを作ります。

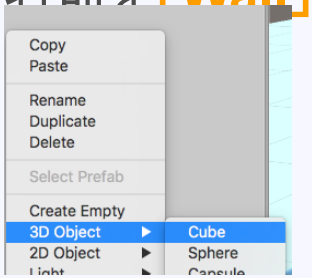


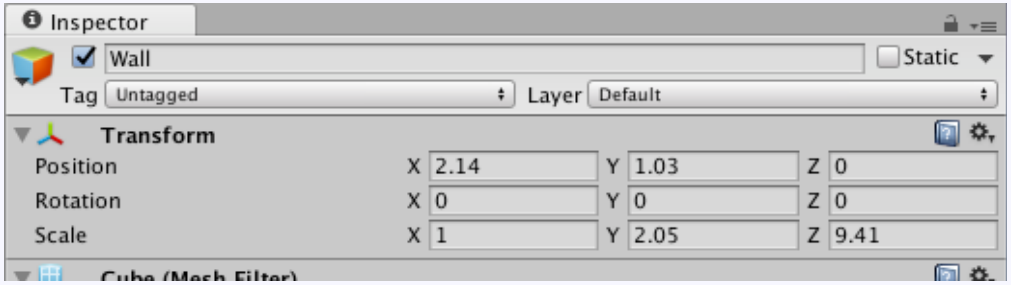
1.壁の作成

Cubeを作り位置と大きさを調整し壁を作ります。

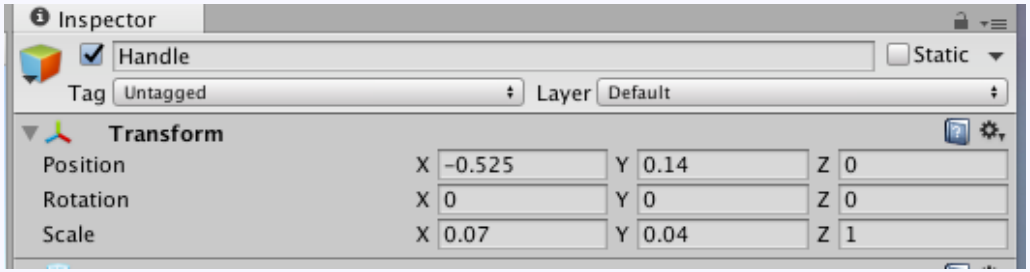
ヒエラルキービューで右クリック ＞ 3DObject ＞ Cube

名前を「Wall」とし、Transformの値はこんな感じにしました。

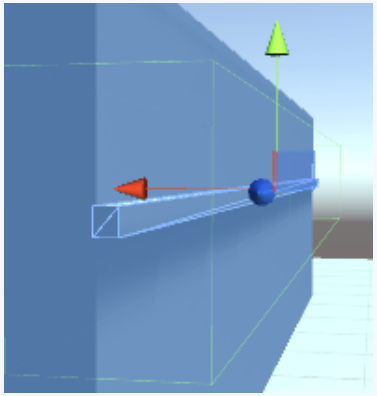




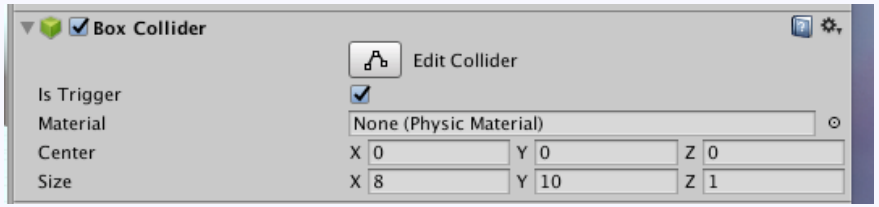
2.手すりを作ります。※壁の子供とする
ヒエラルキービューのWallの上で右クリック ＞ 3DObject ＞ Cube
名前を「**Handle**」とし、Transformの値はこんな感じにしました。



3.手すりに近づいたら自然に手をかけるようにしたいので、
近づいたことを知るためにColliderの値を調整します。
※isTriggerのチェックをつけるのを忘れないように



緑の線の中に
入ったら
手をかける



スクリプトの準備

用意してあるので解説をします。Unity公式マニュアルに載っていたソースを改造しました。

基本的な考え方は、
重み（適用度）と**目標（位置や回転）**
をAnimatorに用意されたメソッドでセットするだけです。

重みをセットするメソッド
```
SetIKPositionWeight(動かしたいボーン, 重み[0.0f～1.0f]); // 位置
SetIKRotationWeight(動かしたいボーン, 重み[0.0f～1.0f]); // 回転
```
動かしたいボーンの指定は、AvatarIKGoalという列挙体で指定できます。

目標をセットするメソッド
```
SetIKPosition(動かしたいボーン, 目標位置[Vector3]); // 位置
SetIKRotation(動かしたいボーン, 目標回転[Quaternion]); // 回転
```

これだけです。

以前、HeadLookControllerでやった「顔を目標物に向ける」
ということもIKでできました！

重みをセットするメソッド
SetLookAtWeight(重み[0.0f～1.0f]);

目標をセットするメソッド
SetLookAtPosition(目標位置[Vector3]);

このプロジェクトのスクリプトの解説

手すりに近づくと、右手を手すりに捕まるようにしています。
この時、顔も手すりに向くようになっています。

OnTriggerEnterとOnTrggerExitを使って、
・手すりに近づいた　手すりから離れた
を判定して、IKを使う/使わないを切り替えています。
また、切り替える際に、時間経過で滑らかに変化するように
重みの値を変数にしてあります。

SetIKPositionで指定できるのは、あくまでも点としての位置。
何も処理を加えないと、手すりの中心部分だけを持つようになってしまうので、
歩いた位置に合わせて手すりを持つ位置を変えるようにしてあります。
ただし、簡易的な対応なので、ゲームなどで使う場合は細かい制御が必要。

スクリプトの準備
解説

```
using UnityEngine;
using System;
using System.Collections;

// Unity公式マニュアルに載ってたスクリプトを改造しました。

[RequireComponent(typeof(Animator))]

public class IKControl : MonoBehaviour {

    protected Animator animator;          // Animator

    public bool ikActive = false;         // IKのON/OFF
    public Transform rightHandObj = null;   // 右手の目標位置
    public Transform lookObj = null;      // 注視点の位置

    private float    ikWeight = 0f;       // IKをON/OFFする時に滑らかにつながるように

    void Start ()
    {
        // Animatorを取得
        animator = GetComponent<Animator>();
    }

    // IKPassにチェックを入れたことで呼ばれるようになる関数
    void OnAnimatorIK()
    {
        // Animatorが取得できたなら
        if(animator) {

            // IKが有効なら
            if(ikActive) {

                // 滑らかにつながるようにカウントアップ
                if (ikWeight < 1.0f)
                {
                    ikWeight += Time.deltaTime;
                    if (ikWeight > 1.0f)
                    {
                        ikWeight = 1.0f;
                    }
                }
            }

            // IKが無効なら
            else {
                // 滑らかにつながるようにカウントダウン
                if (ikWeight > 0.0f)
                {
                    ikWeight -= Time.deltaTime;
                    if (ikWeight < 0.0f)
                    {
                        ikWeight = 0.0f;
                    }
                }

            }

            // 注視点
            if(lookObj != null) {
                // IKの適用度
                animator.SetLookAtWeight(ikWeight);

                // 適用度を反映(位置)
                animator.SetLookAtPosition(lookObj.position);
            }

            // 右手
            if(rightHandObj != null) {
                // 位置のIK適用度
                animator.SetIKPositionWeight(AvatarIKGoal.RightHand,ikWeight);
                // 回転のIK適用度
                animator.SetIKRotationWeight(AvatarIKGoal.RightHand,ikWeight);

                // 適用度を反映(位置) ※手すりにつかまって移動(ただし上方向限定)
                Vector3 pos = rightHandObj.position;
                pos.z = transform.position.z;
                animator.SetIKPosition(AvatarIKGoal.RightHand,pos);

                // 適用度を反映(回転)
                animator.SetIKRotation(AvatarIKGoal.RightHand,rightHandObj.rotation);
            }
        }
    }


    void OnTriggerEnter(Collider collider)
    {
        if (rightHandObj)
        {
            // 手すりに触れたなら
            if (collider.name == rightHandObj.name)
            {
                ikActive = true;    // IKを有効
            }
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if (rightHandObj)
        {
            // 手すりから離れたなら
            if (collider.name == rightHandObj.name)
            {
                ikActive = false;    // IKを無効s
            }
        }
    }
}
```

ちっちゃくて見えないけど一応ソースも。
プロジェクト内のソースを確認ください。

1.スクリプトを「RobotKyle」にアタッチする
　ヒエラルキービューで**「RobotKyle」**を選択し、
　　一番下の「Add Component」ボタン　>　Scripts > IKControl

ここまでできたら実行ボタンで動作確認！
壁に近づいたら、右手が手すりの方に行くはず。
※簡易対応なので、キャラの向きによっては手が変な方向に・・・

地面の形状に合わせて足を動かす場合は、
足から下に向かってレイを飛ばして、当たり判定のあった位置を、
目標位置にするようです。
ただ、地形によっては、足が変な方に曲がっちゃうので、
制限をつけたりなどの工夫は必要。

まとめ

- AnimatorウィンドウのLayersから
  IKPassの設定を行う。

- Animatorコンポーネントから、
  重みと目標をセットする。

今回もページ数が多いから難しく感じたかもしれませんが、
ページの大半は、IK以前の設定でした。
なので、IKだけを見れば非常に簡単に使えます。
ただし、自然な動作に見せるためには、それなりの工夫が必要。

■プロジェクト一式はここ

IK
http://monolizm.com/sab/src/ik.zip

■参考サイト

Unity公式マニュアル　IK
https://docs.unity3d.com/jp/current/Manual/InverseKinematics.html

ご清聴ありがとうございました