演習：フィジカル・コンピューティング（後半）

01/11 ３限・４限

補講｜UnityのアニメーションとArduinoの接続

自由課題

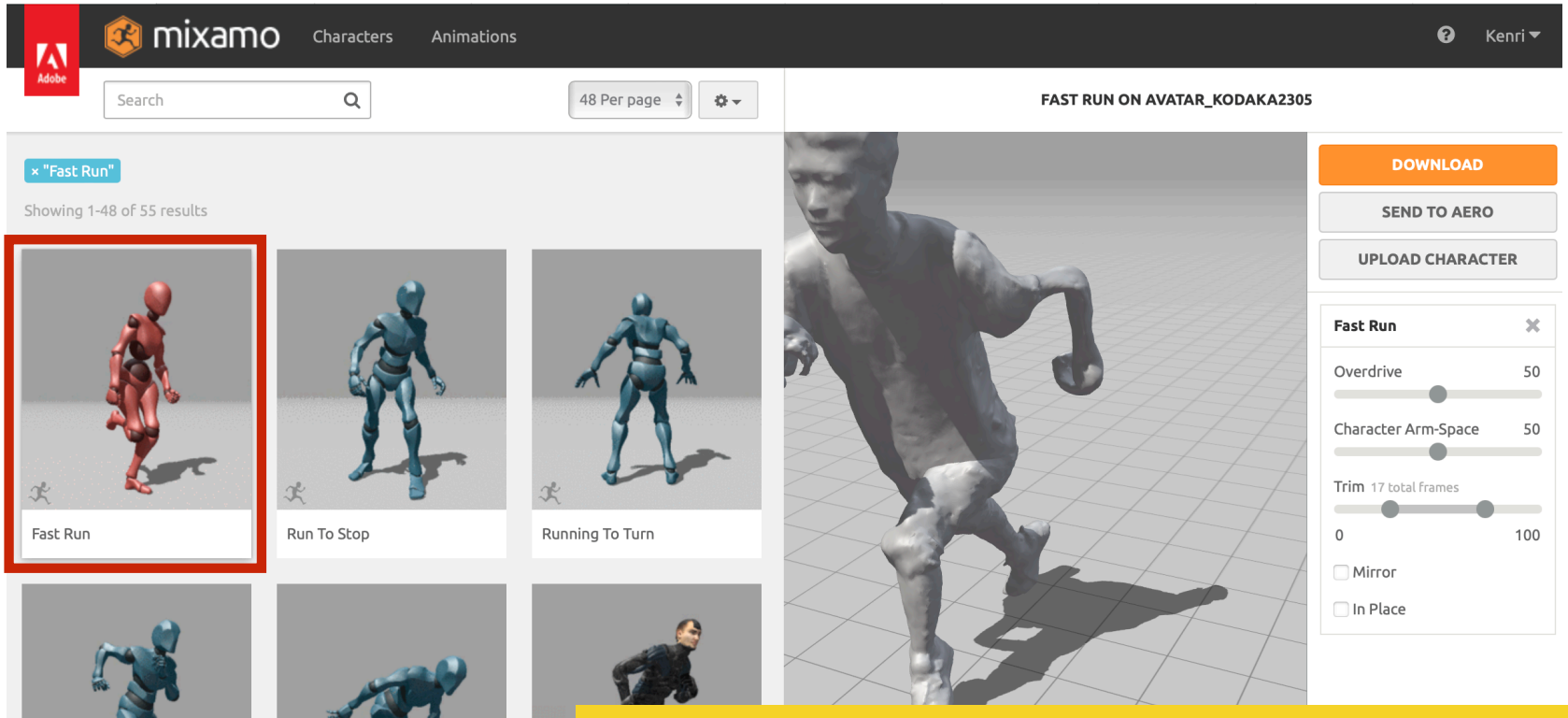
01/18 ３限・４限

自由課題

01/25 ３限・４限

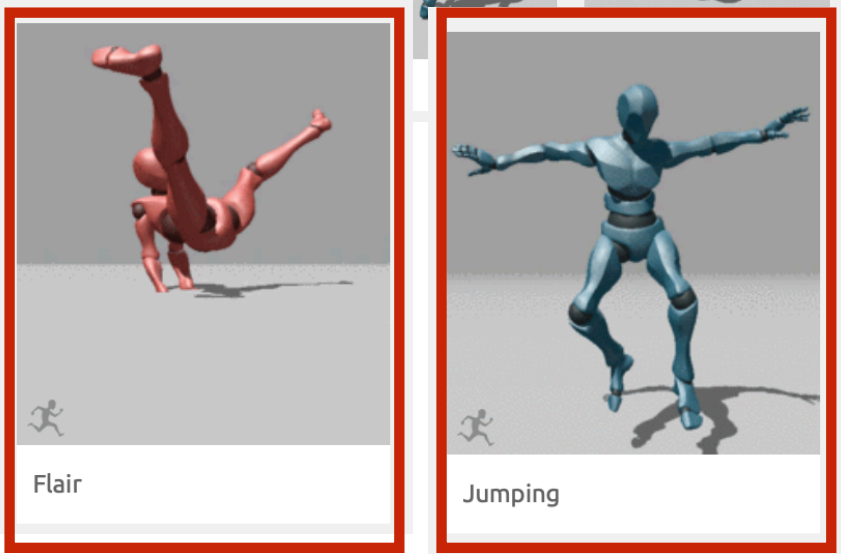
講評会（課題提出）

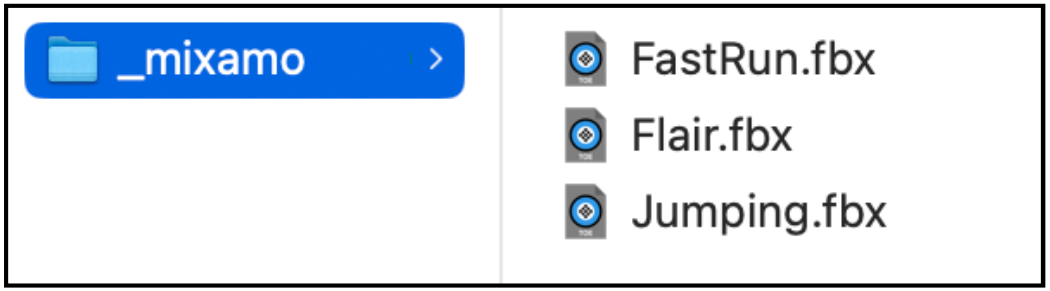
MIXAMよりFBXのダウンロード



https://www.mixamo.com

作成済みの自分自身のアバターのOBJファイルを用いて、3つのパターンのFBXファイルを作成し、ダウンロードしておいてください。





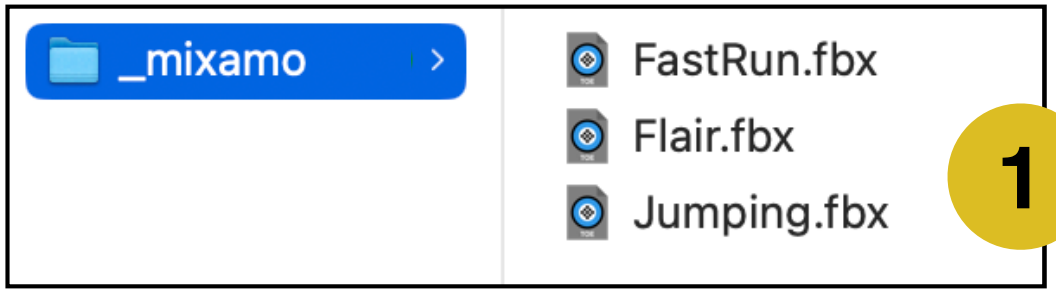
UNITYへの移植

1

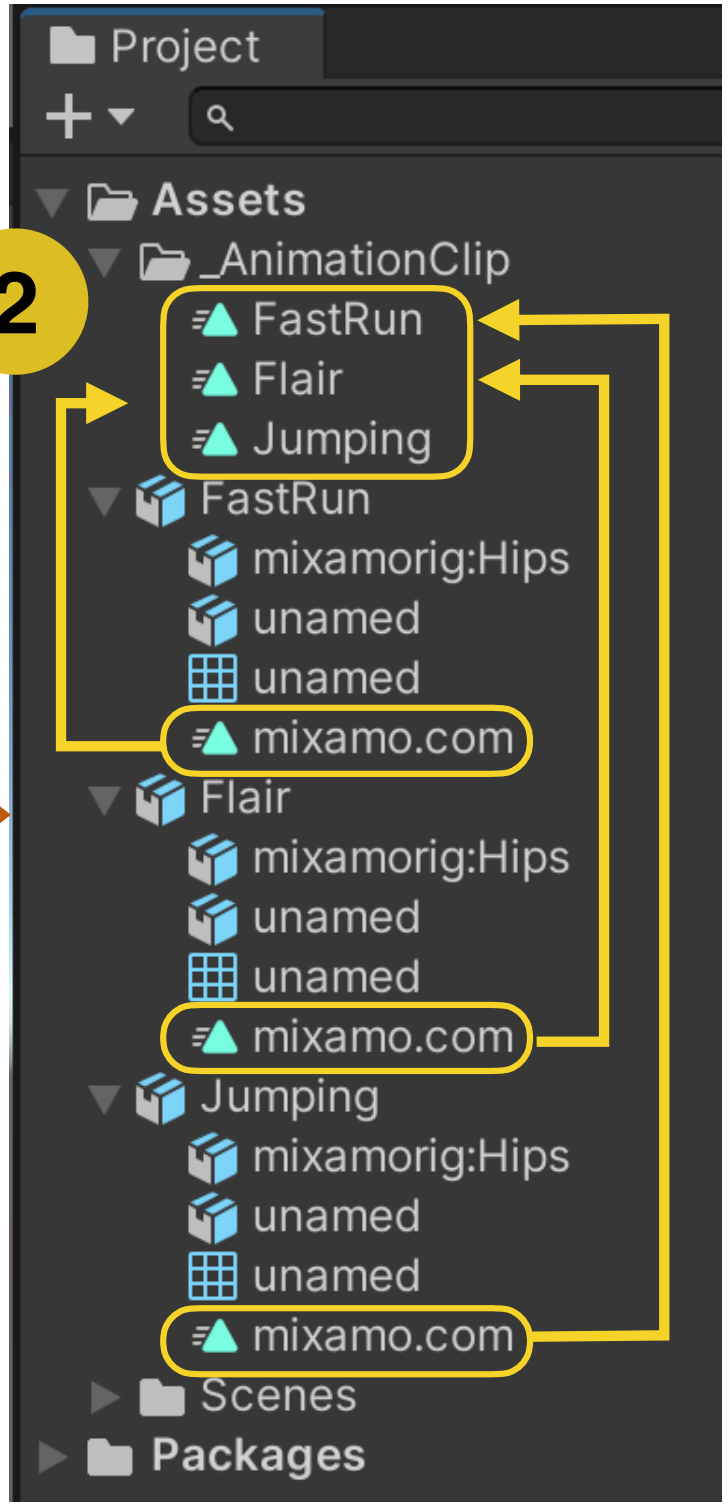
新しく作成したUNITY新規ファイルのプロジェクトビューに、3つのFBXファイルを放り込みます。

2

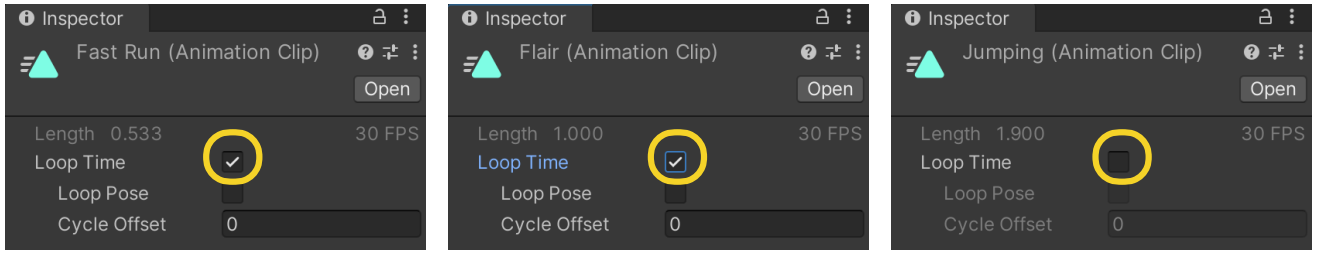
それぞれのFBXフォルダの中にあるアニメーションクリップ「mixamo.com」を⌘Dで複製し、対応するFBXの名前に改名したのちに、新しく作ったフォルダ（_AnimationClip）にまとめましょう。



1



2



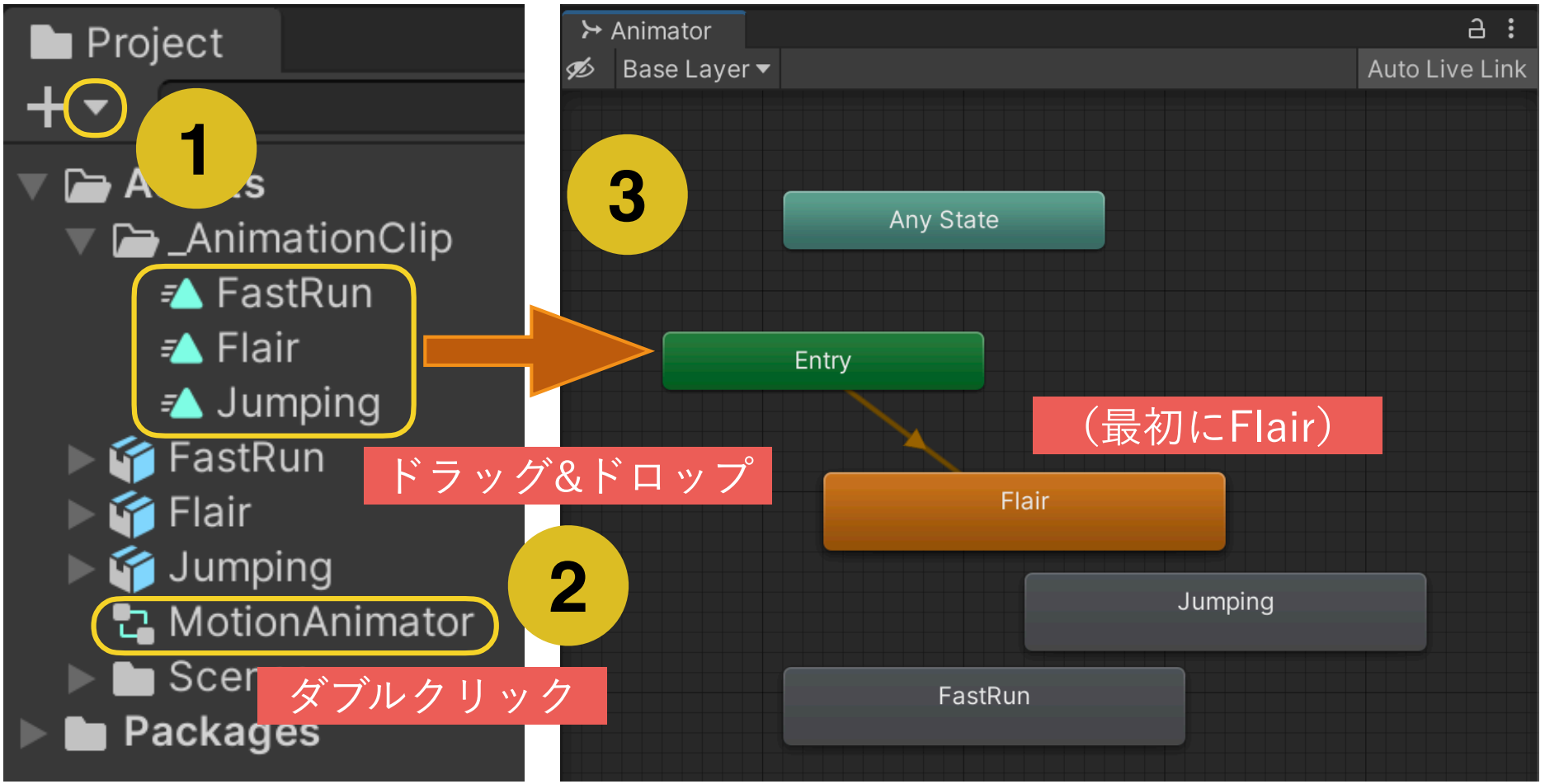
3

それぞれのアニメーションクリップのインスペクタから、ループの有無を設定します（Jumpingのみループなし）。

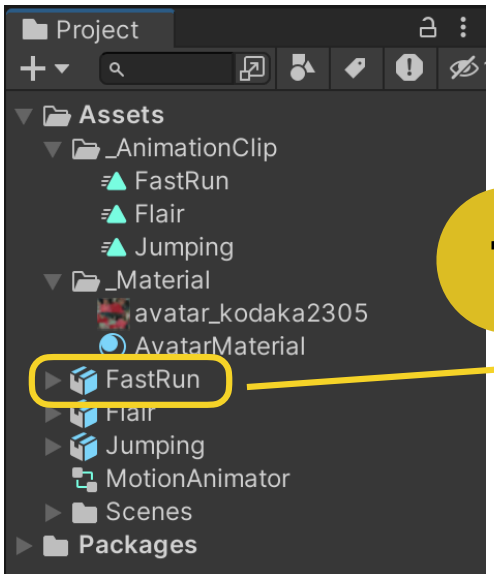
Animation Controllerの作成

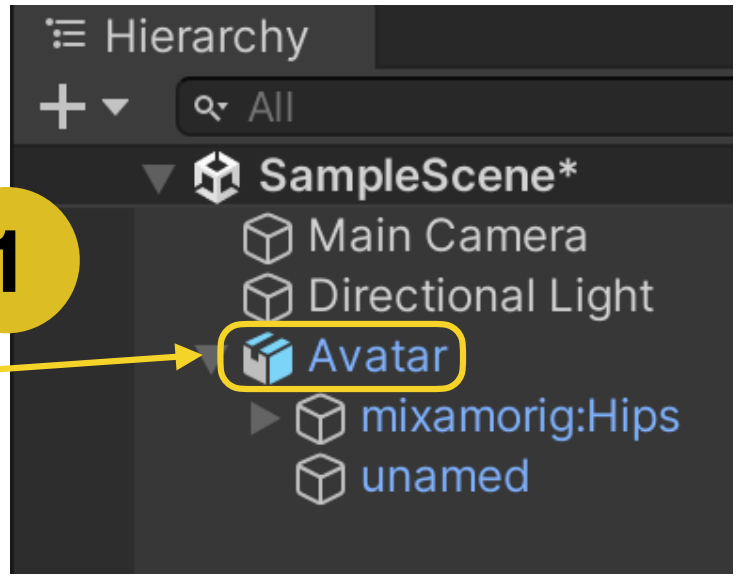
1 プロジェクトタブより「Animation Controller」を新規作成し、適当に名前を変えます。

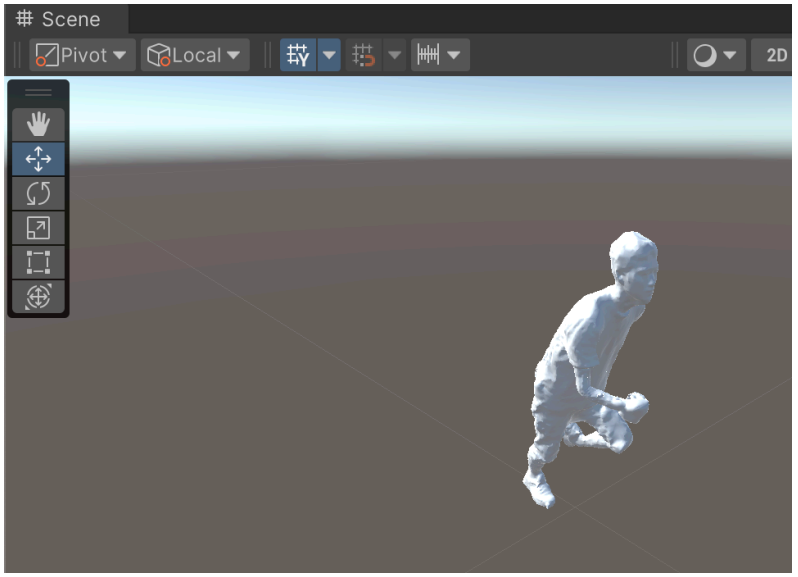
23 新規作成したアニメーターをダブルクリックしてAnimator Vlewを開き、先に複製した3つのAnimation Clipをドラッグ&ドロップします。この際、最初にFlairを入れて、EntryからのTransitionがつながるようにしてください。



アバターを配置し、テクスチャを貼る





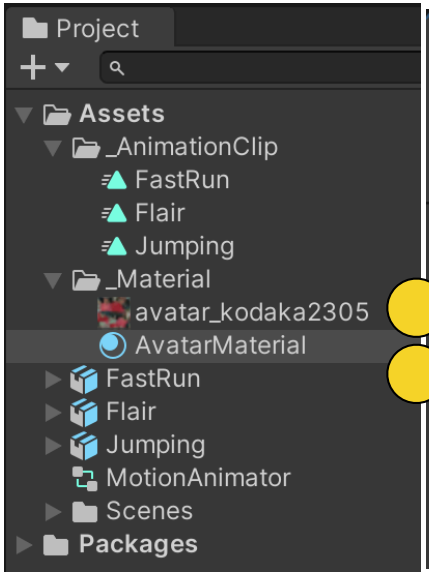


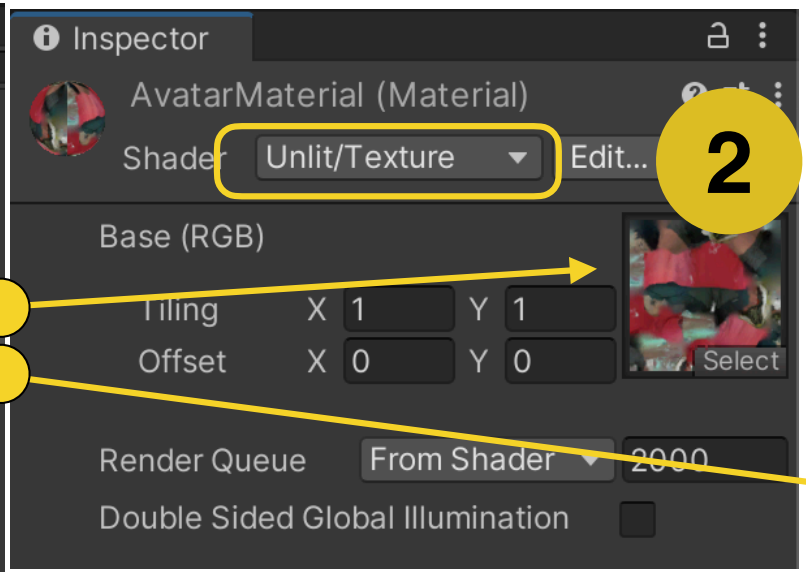
1

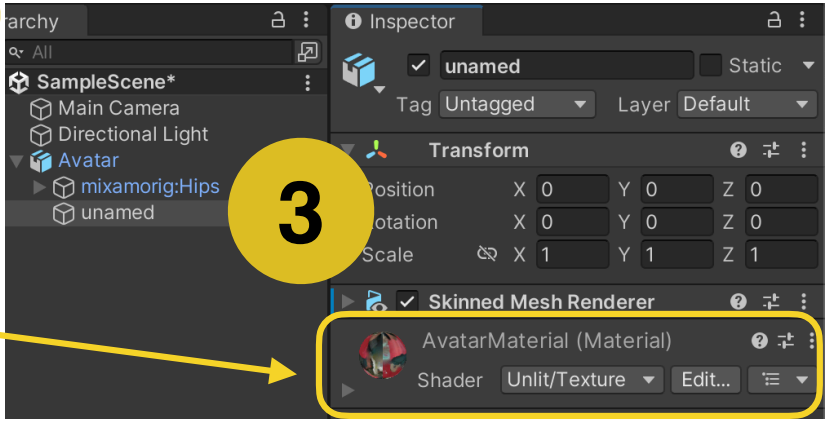
Mixamo経由のFBXファイル（どれでもよい）をヒエラルキービューに放り込み、Avatarなどのわかりやすい名前に変更してください。

23

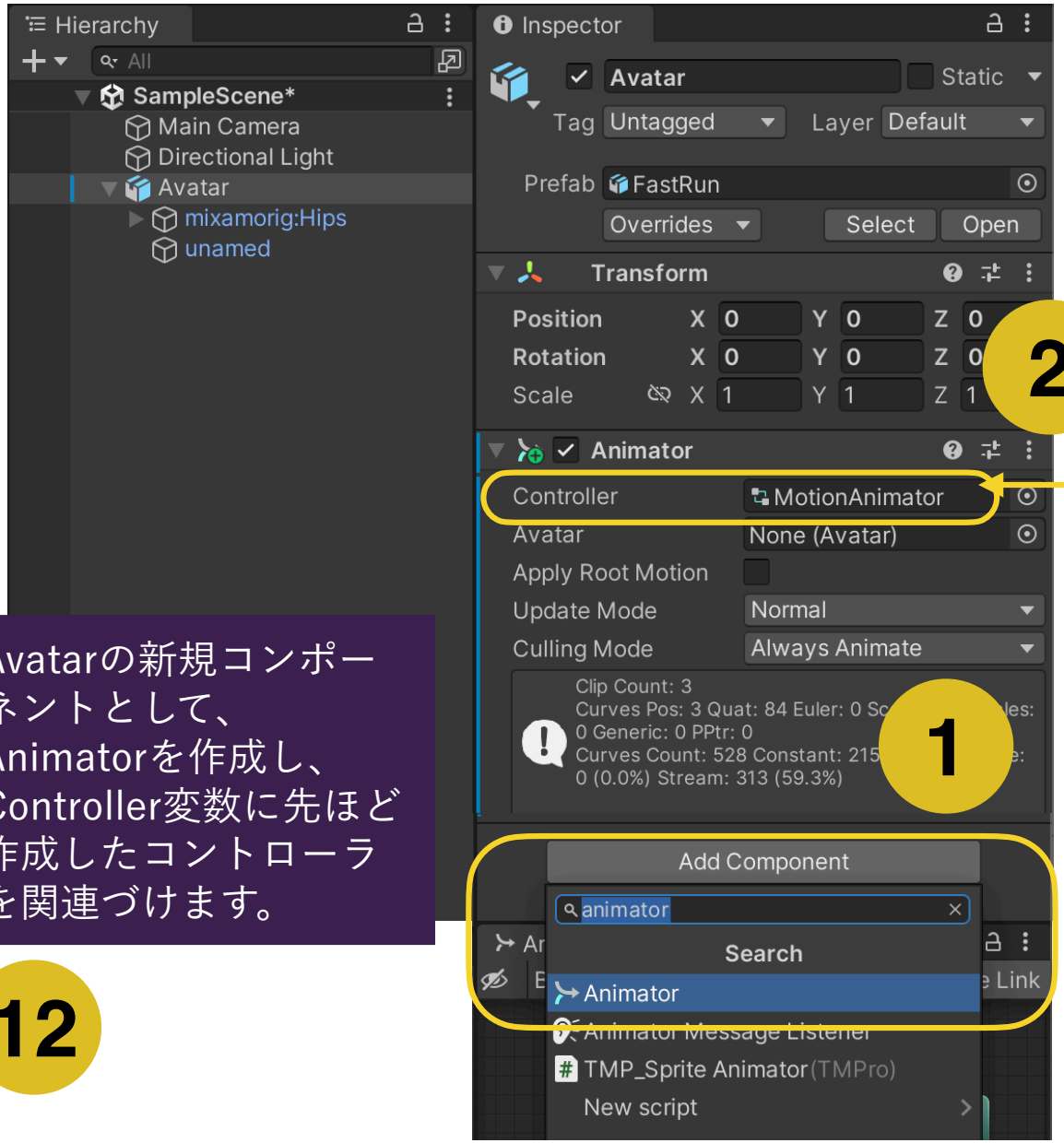
新規作成したマテリアルのShaderを「Unlit/Texture」とし、Metashapeから出力されたアバターのテクスチャ（jpg）を貼ってください。最後に、Avatar/unamedに、マテリアルを紐づけます。

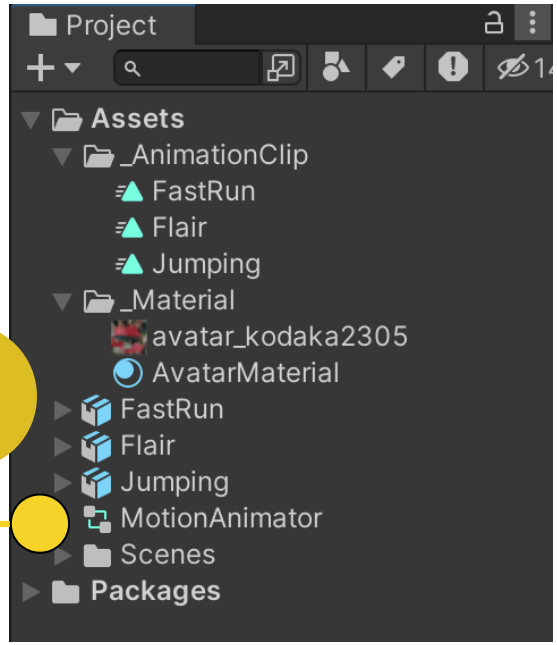






アバターとコントローラの関連付け

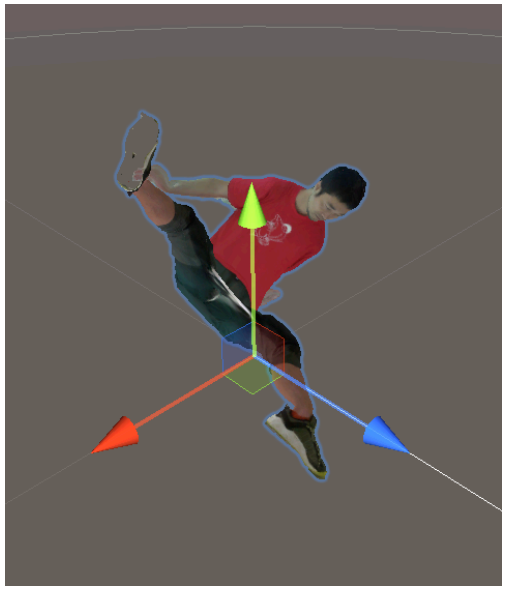




2

1

Avatarの新規コンポーネントとして、Animatorを作成し、Controller変数に先ほど作成したコントローラを関連づけます。

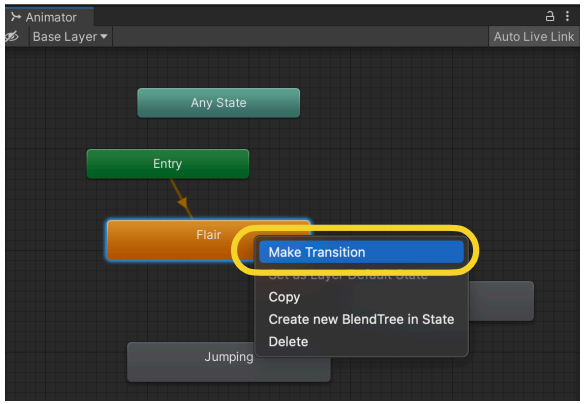


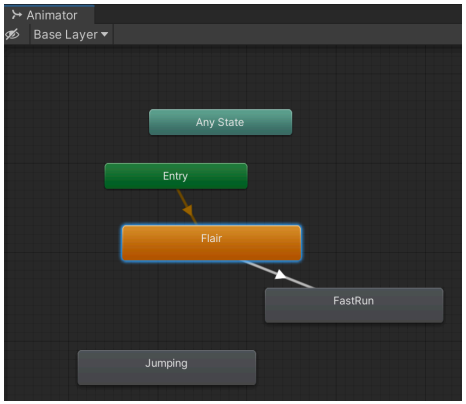
3

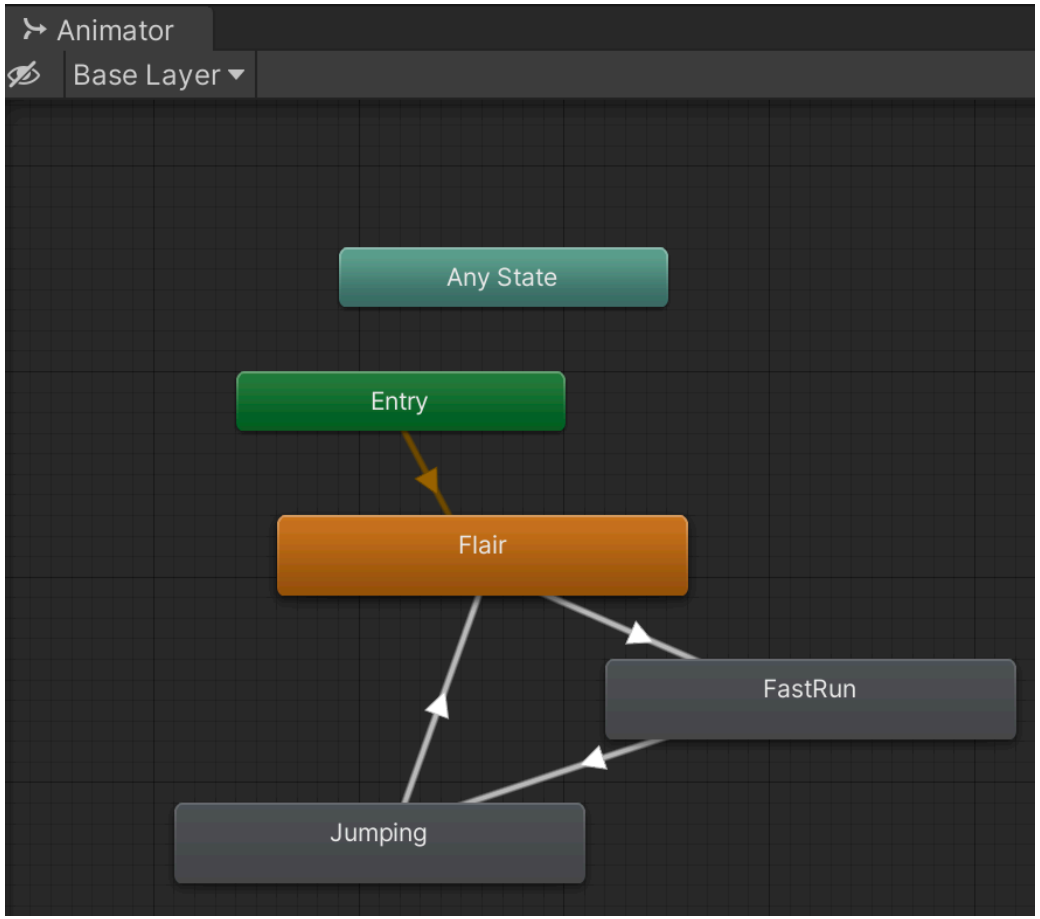
実行するとFlairのアニメーションを踊り続けます。

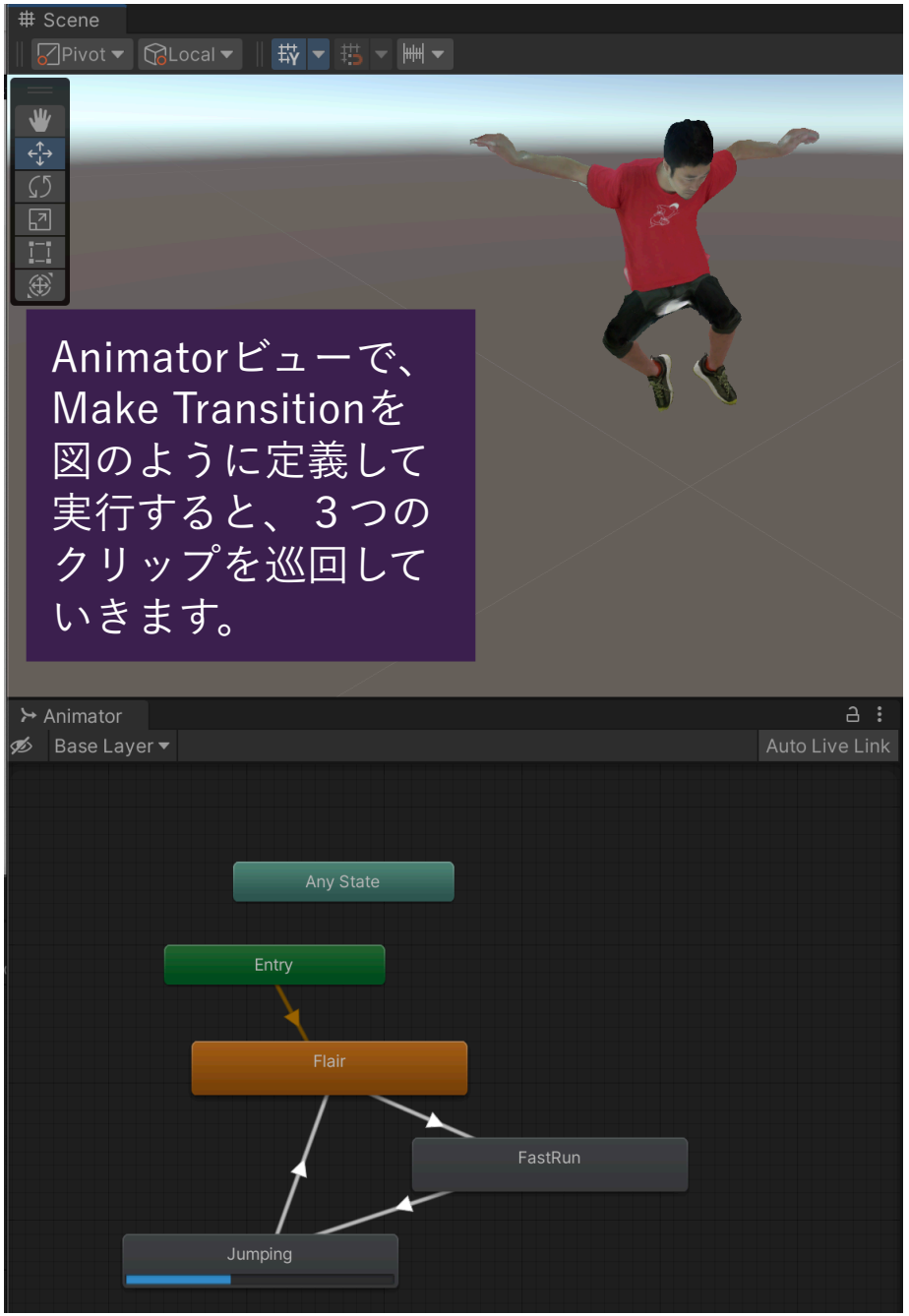
12

複数のアニメーションクリップを遷移させる

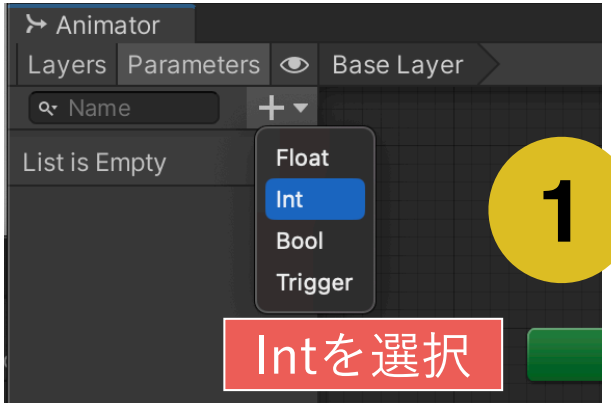




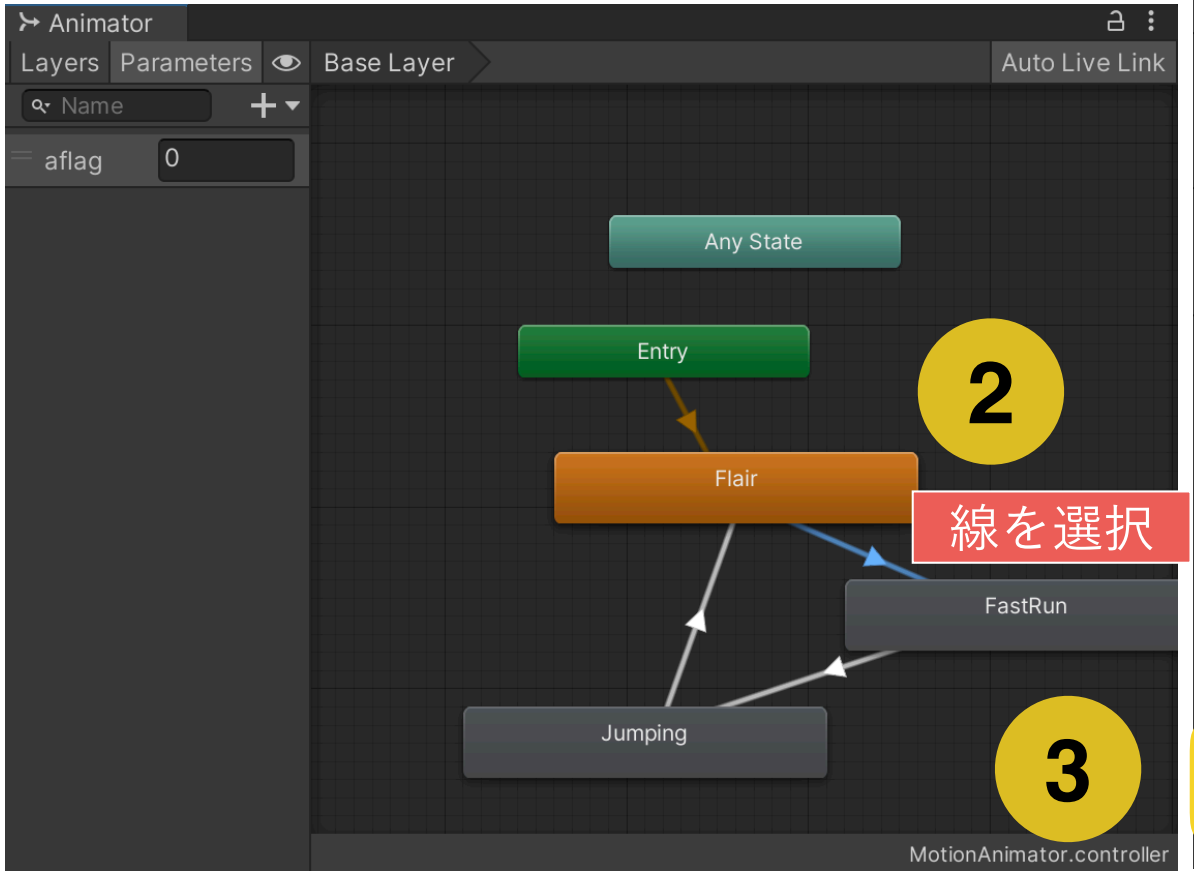


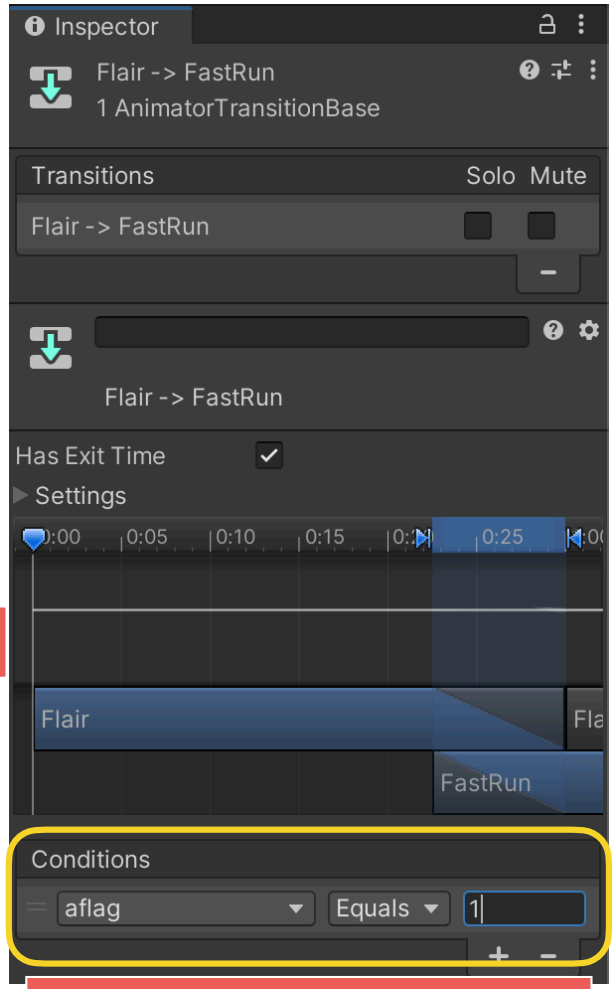


トランジッションを変数で駆動する（1/2）



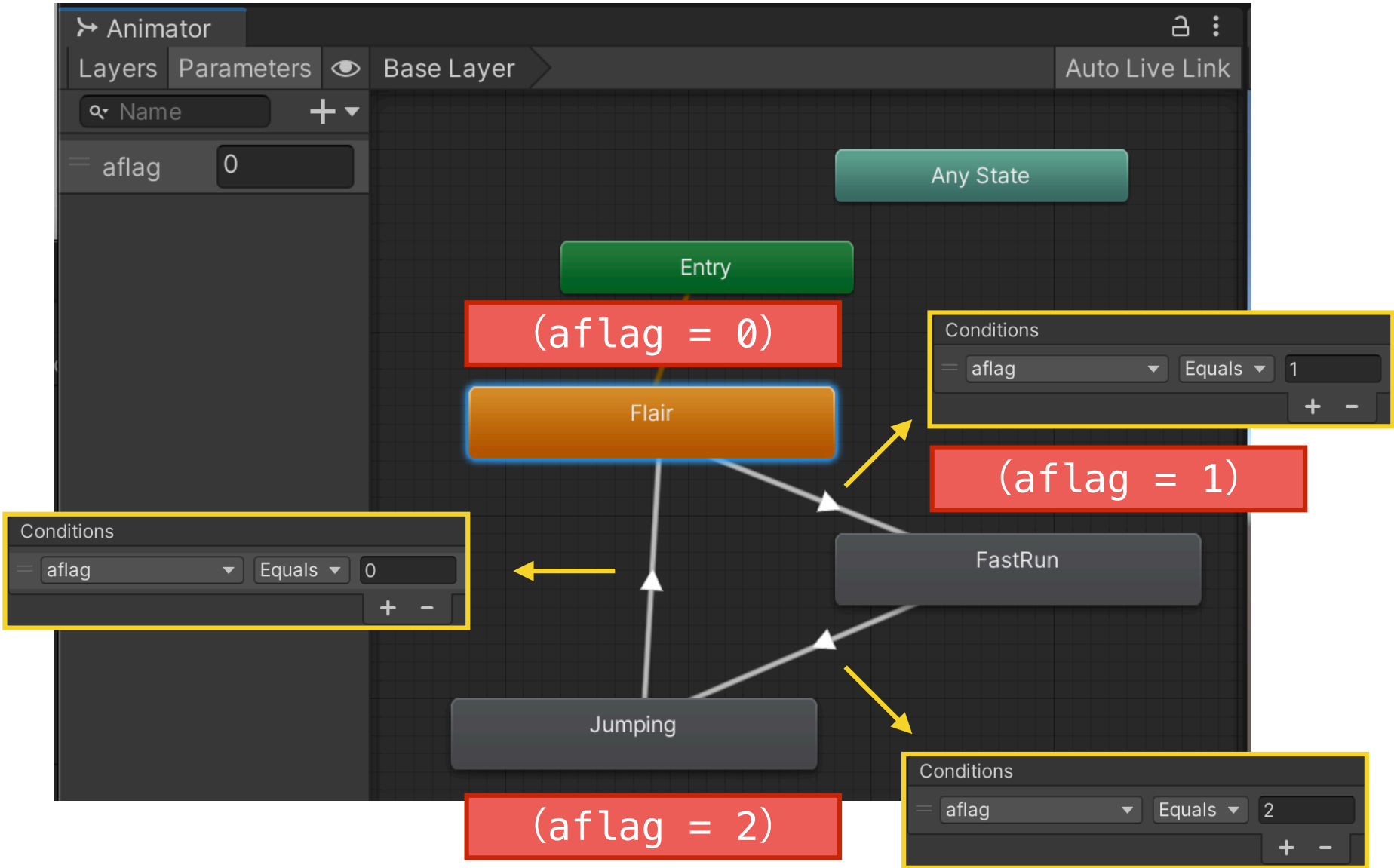
Animatorビューで、int型の変数（aflag：初期値０）を追加し、「Flair→FastRun」の遷移が「aflag=1]となることで発動するように変更します。この状態で実行しても、FastRunへのトランジッションが生じることはありません。





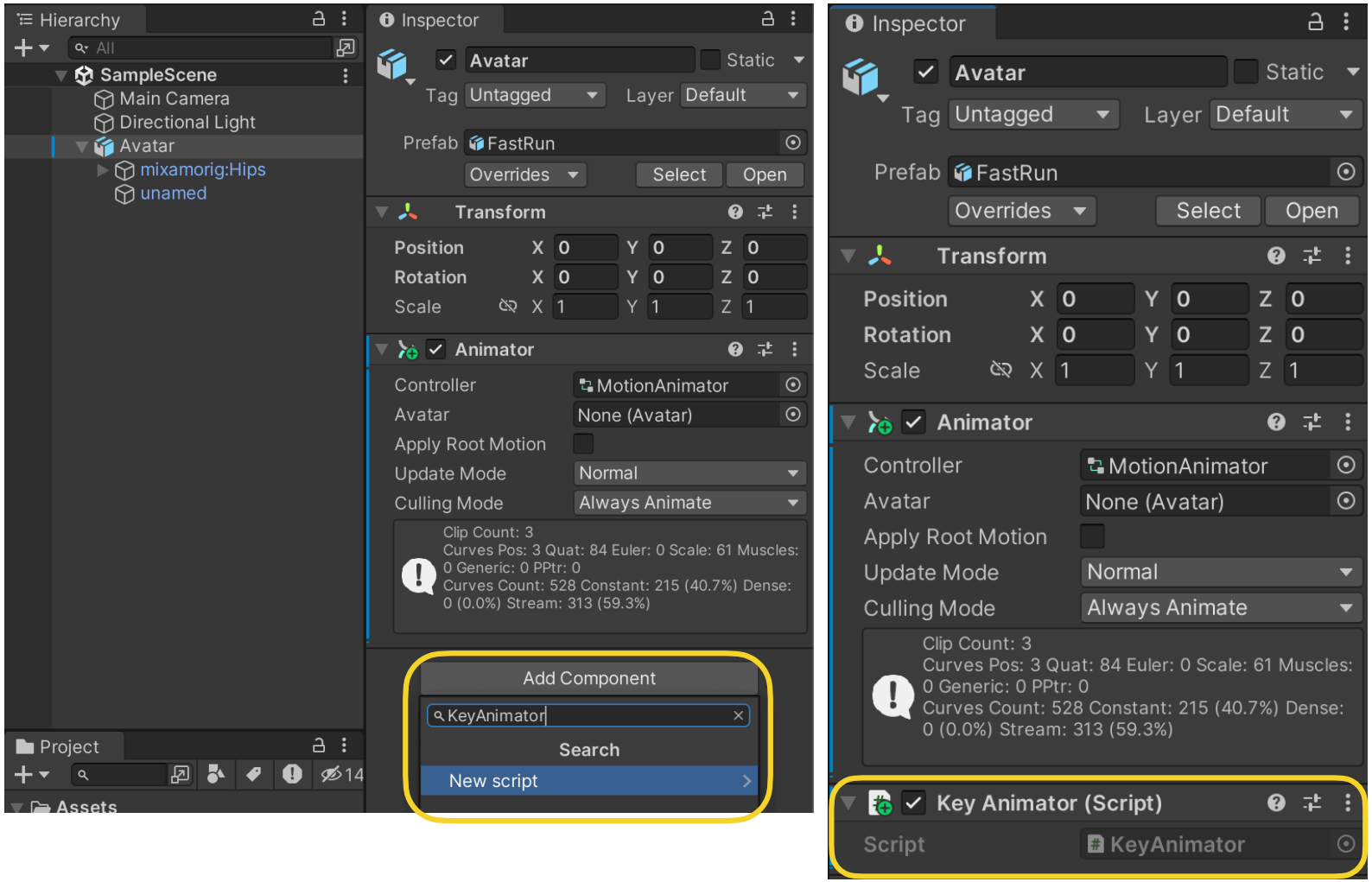
遷移の条件を指定

トランジッションを変数で駆動する（2/2）



同様に3つのトランジッションが、aflagの値（の変更）をフラグにして生じるように設定しておきます。

KEYによるトランジッションの制御（1/4）



Avatarに新たに「KeyAnimator」という名前のスクリプトを生成します。

KEYによるトランジッションの制御（2/4）

KeyAnimator.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class KeyAnimator : MonoBehaviour
6  {
7      public float speed = 0;
8      Animator animator;
9
10
```

public変数としてspeedを定義

```
12     void Start()
13     {
14         animator = GetComponent<Animator>();
15         animator.SetInteger("aflag",0);
16     }
```

animatorコンポーネントを取得

起動時にaflagをゼロにセット

Start()

```
17
18     void Update()
19     {
20
21         if (animator.GetCurrentAnimatorStateInfo(0).IsName("Flair")){
22
23             if(Input.GetKeyDown(KeyCode.UpArrow)){
24                 animator.SetInteger("aflag",1);
25             }
26
27         }
28
```

Animationの状態が「Flair」のとき

KEY上ボタンが押されると、aflagを1にセット

Updatate() 1/3

Updatate() 2/3

Animationの状態が「FastRun」のとき

```
29         if (animator.GetCurrentAnimatorStateInfo(0).IsName("FastRun"))
30         {
31
32             if(Input.GetKeyDown(KeyCode.UpArrow)){
33                 speed = Mathf.Min(speed+1,20);
34
35                 if(speed>=20){
36                     speed = 0;
37                     animator.SetInteger("aflag",2);
38                 }
39             }
40
41             if(Input.GetKeyDown(KeyCode.DownArrow)){
42                 speed = Mathf.Max(speed-1,0);
43             }
44
45         }
```

KEY上ボタンが押されると、speedを1増やす

speedが20になると、aflagを2にセット

KEY下ボタンが押されると、speedを1減らす

Animationの状態が「Jumping」のとき

```
47         if (animator.GetCurrentAnimatorStateInfo(0).IsName("Jumping"))
48         {
49
50             if(Input.GetKeyDown(KeyCode.LeftArrow)){
51                 animator.SetInteger("aflag",0);
52             }
53         }
54
55     }
56
```

KEY左ボタンが押されると、aflagを0にセット

Updatate() 3/3

```
56
57     public void SetSpeed(float val){
58         this.speed = val;
59     }
60
61 }
62
```

SetSpeed()

外部からspeedの値を変更するためのパブリックなメソッドを定義しておきます。

KEYによるトランジッションの制御（2/4）

KeyAnimator.cs

Updatate() 2/3

Animationの状態が「FastRun」のとき

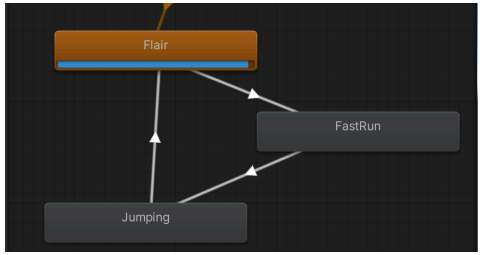
```
if (animator.GetCurrentAnimatorStateInfo(0).IsName("FastRun")){

    if(Input.GetKeyDown(KeyCode.UpArrow)){
        speed = Mathf.Min(speed+1,20);

        if(speed>=20){
            speed = 0;
            animator.SetInteger("aflag",2);
        }
    }

    if(Input.GetKeyDown(KeyCode.DownArrow)){
        speed = Mathf.Max(speed-1,0);
    }

    Vector3 pos0 = this.transform.position;
    Vector3 pos1 = new Vector3(pos0.x,pos0.y,pos0.z + speed * 0.001f);
    this.transform.position = pos1;
}
```

speedで位置に反映させたい場合、例えば以下のようなコードを入れてください。

KEYによるトランジッションの制御（3/4）

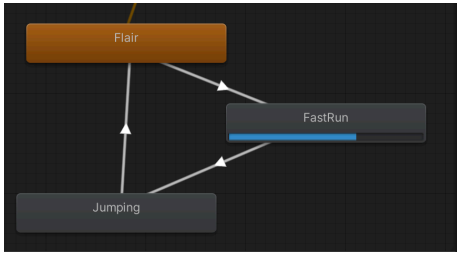
Flair

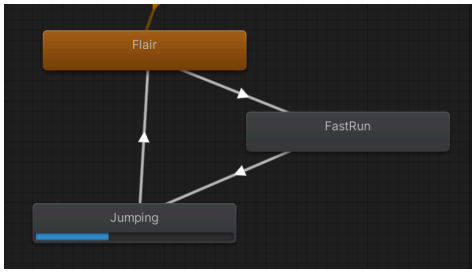
aflag=0



speed = 0;

FastRun





aflag=2

speedが20以上となった時

aflag=1

Jumping

speed++;

speed−;

OSC環境の設定（復習です）

UNITY側の準備

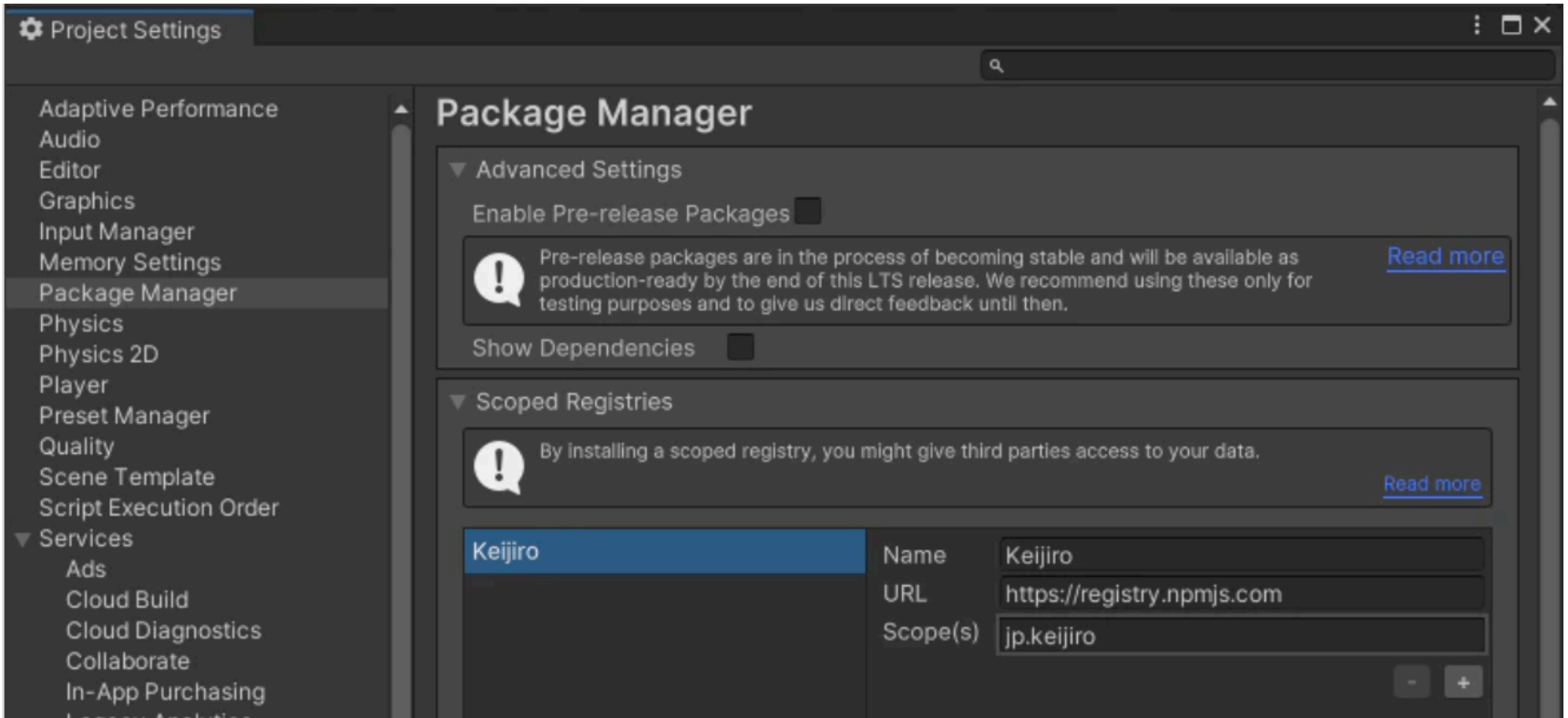
OscJackのインポート（1/2）

Edit→Project Settings→Package ManagerのScoped Registiesで以下のリポジトリ情報を登録

Name : Keijiro

URL : https://registry.npmjs.com

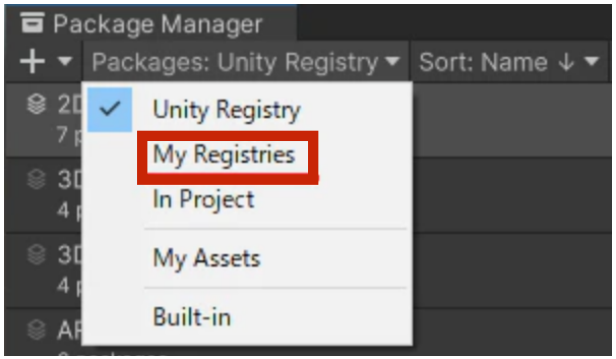
Scope(s) : jp.keijiro



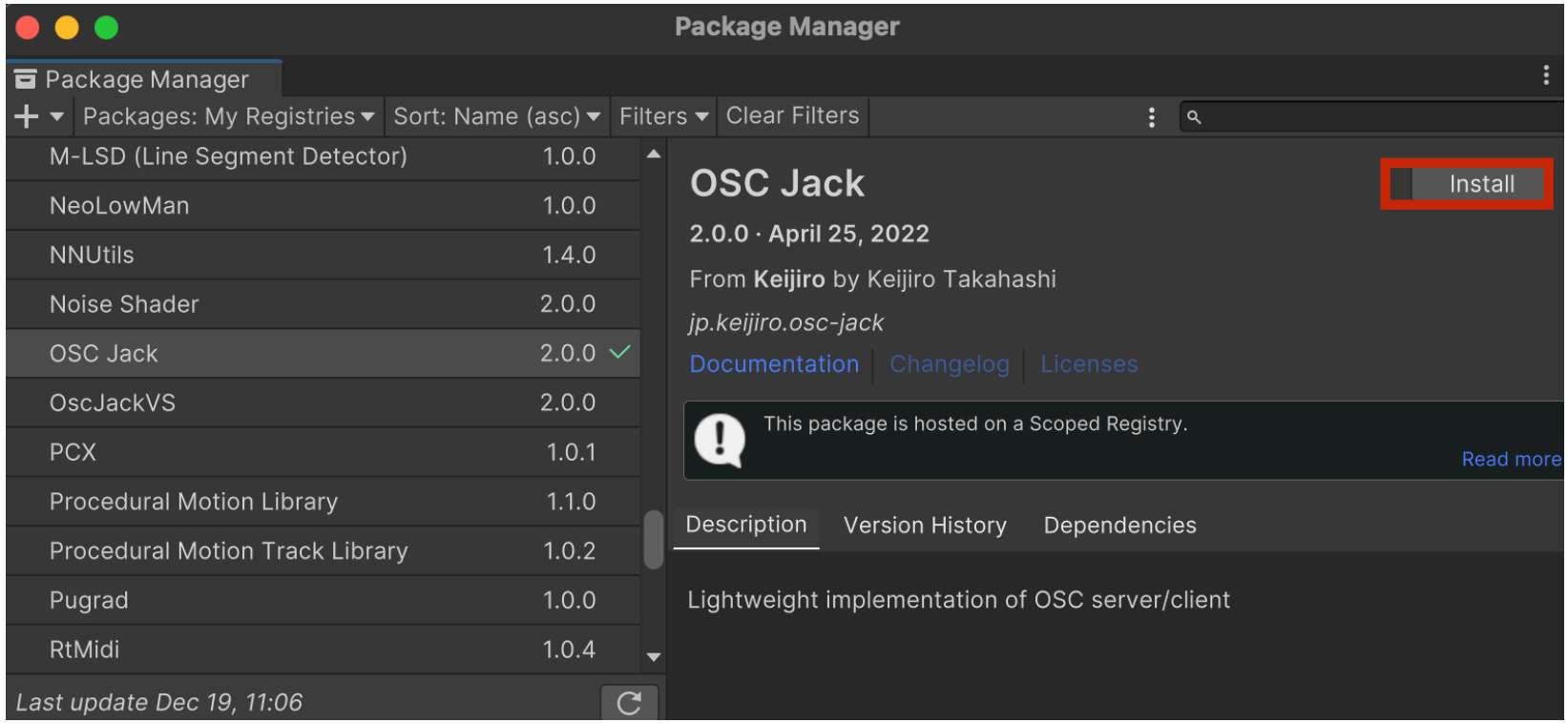
UNITY側の準備

OscJackのインポート（2/2）

Window→Package Managerを開き、
ウィンドウ左上のPackages:の項目をMy Registriesに変更する.



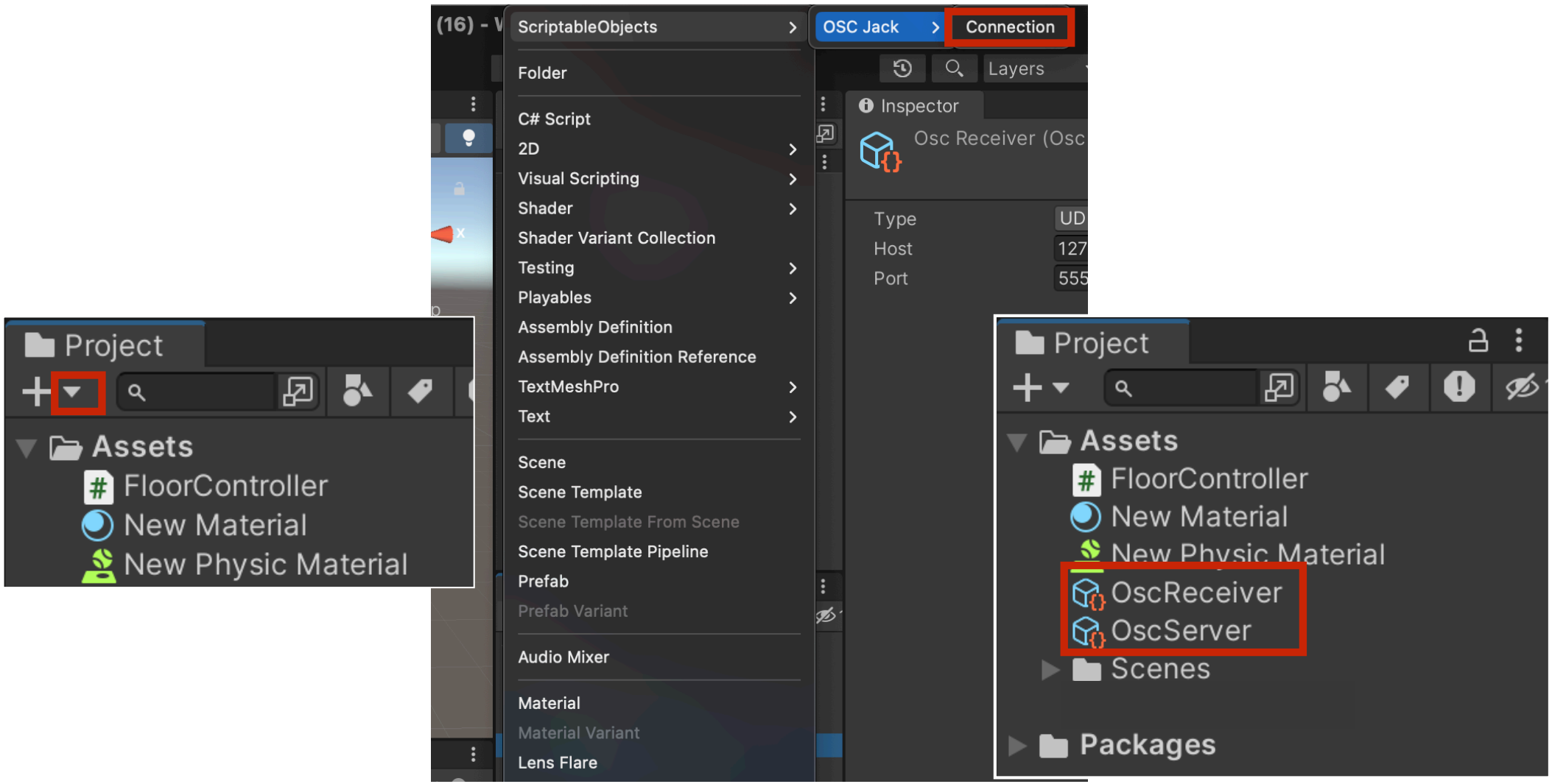
リストから OSCJackを選択し、Installボタンを押す. これでインポートは完了.



UNITY側の準備

ポート情報を管理するConnectionの設定

Projectタブ左上▼→ScriptableObjects→OSC Jack→Connection
以上の接続情報を管理するオブジェクトを2つ生成し（Unity側とProcessing側）、それぞれOscServer、OscReceiverとリネームしましょう。



UNITY側の準備

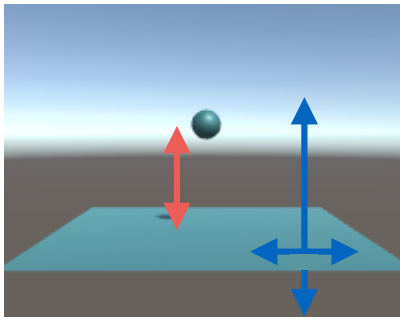
受信編

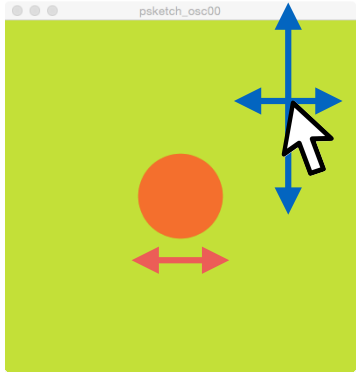
127.0.0.1 / 6666

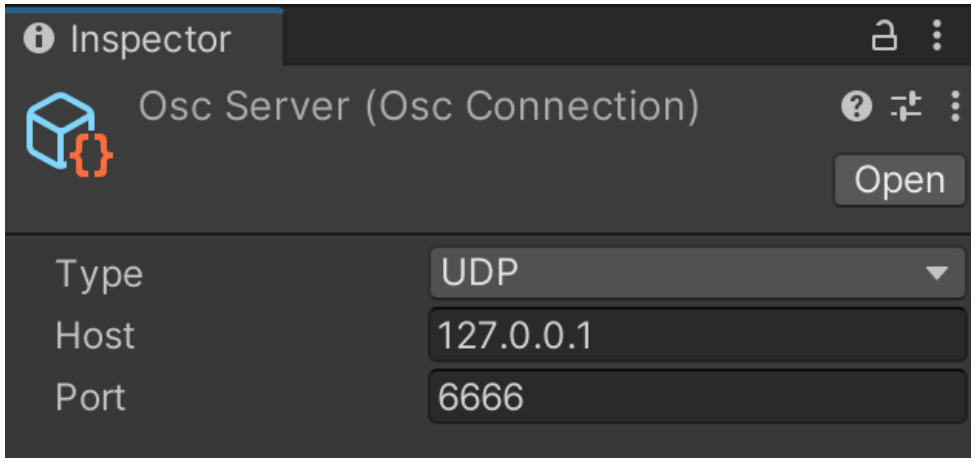
127.0.0.1 / 5555

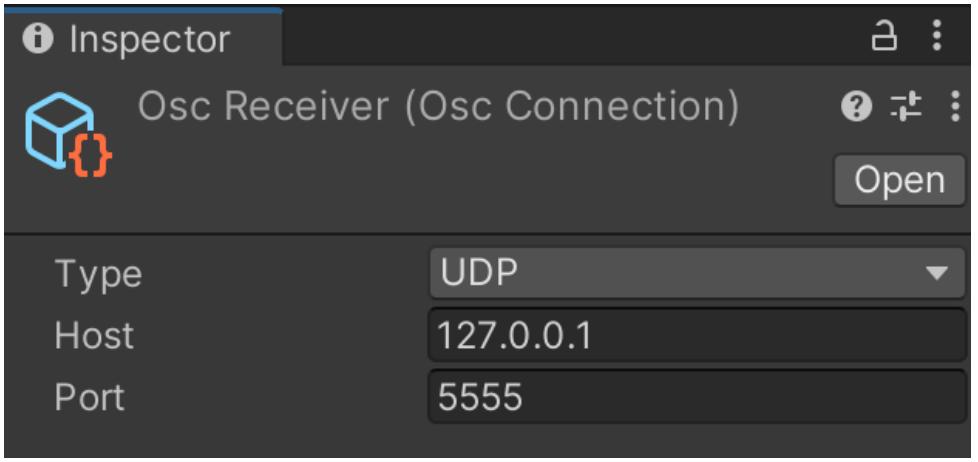
SERVER

Client





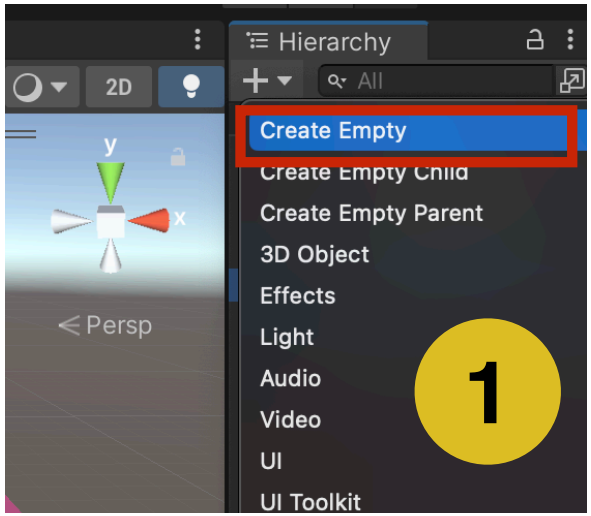




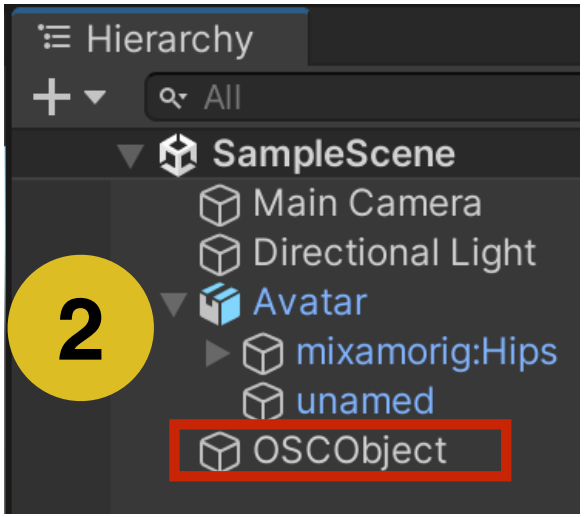
UNITY側の準備

受信編

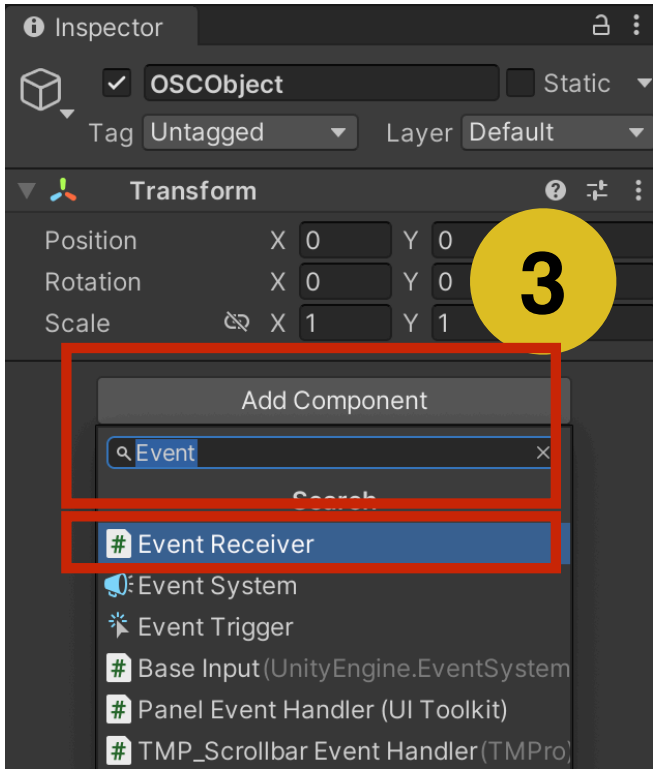
EventReceiverの設定

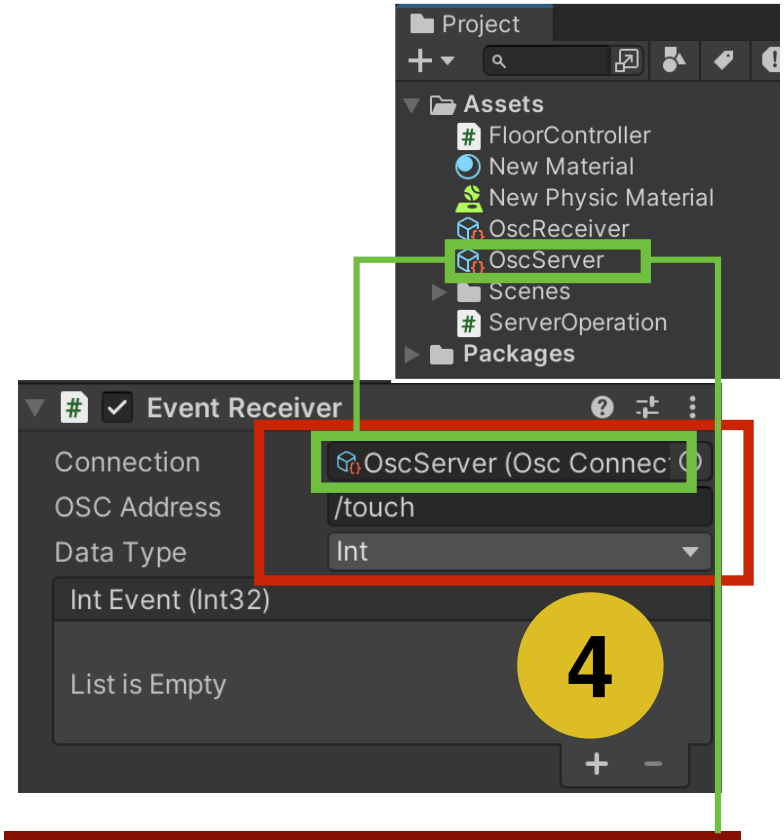


Add Componentより「EventReceiver」を2つ追加.します。受信するOSC情報を管理するためのコンポーネントです。



空のゲームオブジェクトを作成し、OSCObjectとリネームする。





Connectionにすでに作成した「OscServer」をアタッチし、OSCAddressに/touchを、Data Typeを「Int」に設定しておきます。

圧力センサの値をビジュアライズする

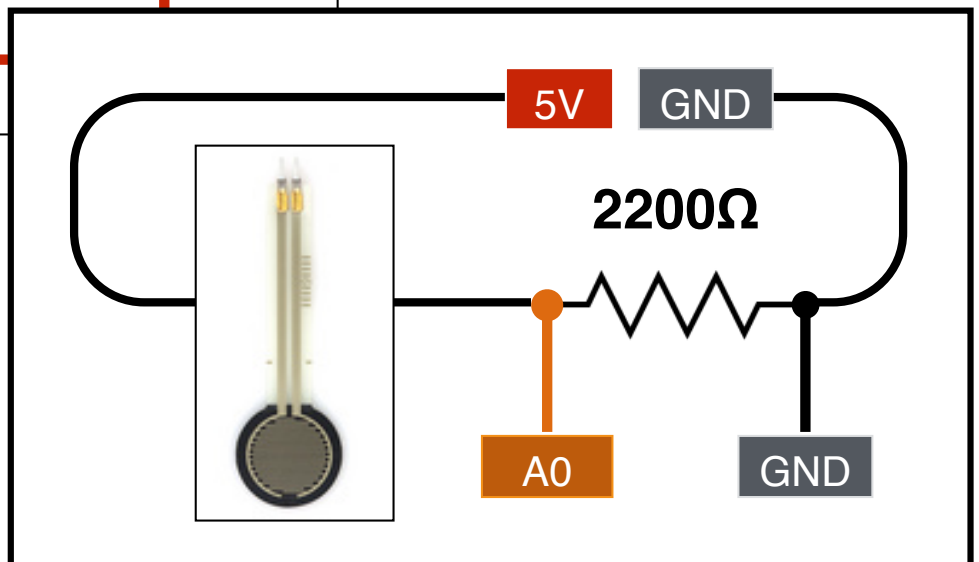
psketch_Touch.pde

```
1  import processing.serial.*;
2  import cc.arduino.*;
3  import org.firmata.*;
4
5  Arduino arduino;
6  int sensorPin = 0; //A0に対応
7
8  void setup(){
9    //println(Arduino.list());
10   arduino = new Arduino(this, Arduino.list()[3] ,57600);
11   arduino.pinMode(sensorPin, Arduino.INPUT);
12   size(800,100);
13 }
14
15 void draw(){
16   float val = arduino.analogRead(sensorPin);
17   println(4.63 * val / 1023.);
18
19   background(0);  fill(255,100,0);  stroke(255);
20   rect(0,0,width*val/1024.,height);
21
22 }
```

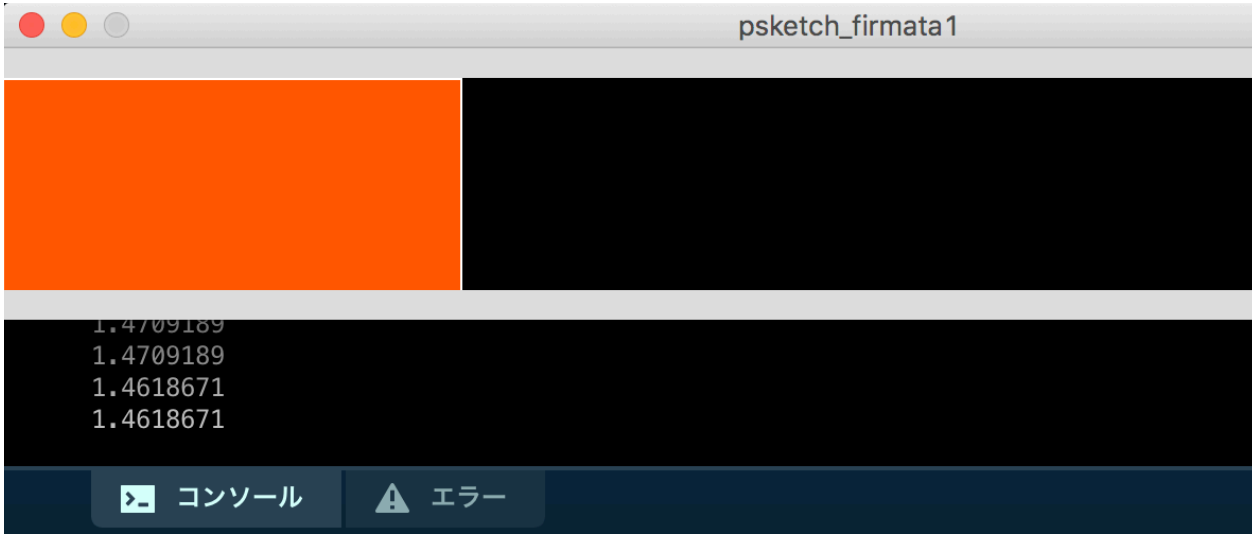
ArduinoのANピンは, Processingでは整数型のNに対応します.

0からMax（V）を0から1023のレンジで読み込みます. この例でのMaxは4.63Vです.

背景は黒、塗りつぶしオレンジ、枠線の色は白で、valに応じて長方形の幅を変化させています.

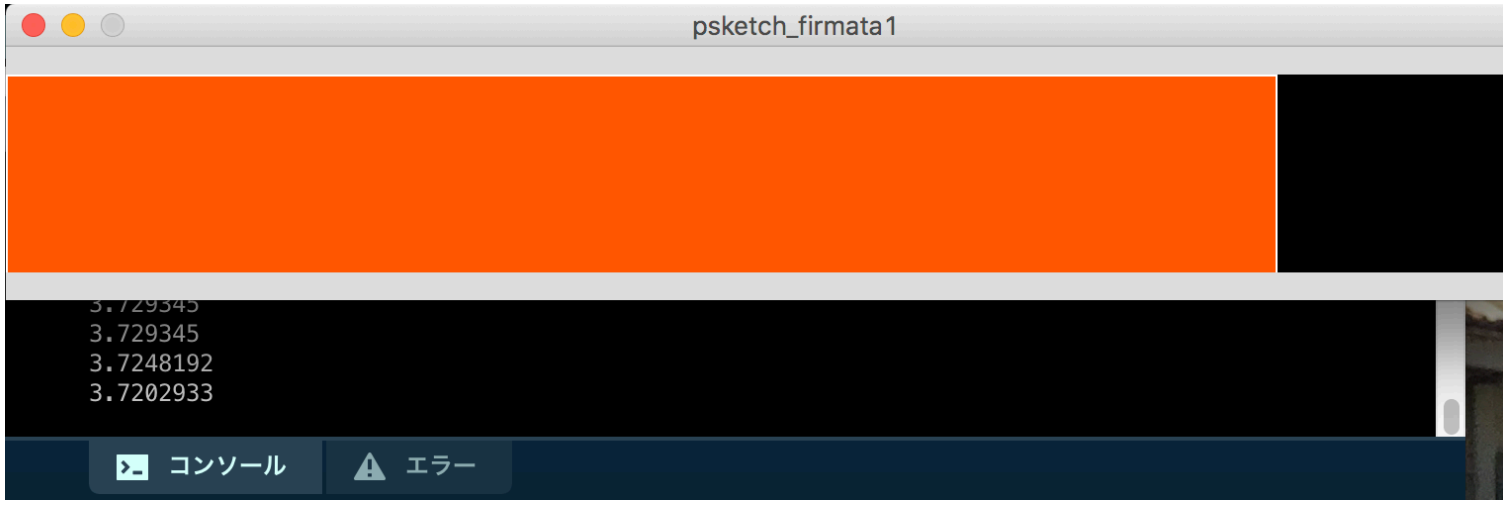


圧力センサの値をビジュアライズする



実行結果

圧力センサを押すと, バーの長さが伸び縮みします.



圧力センサの値をOSC通信で送信

psketch_Touch_osc.pde

```
1  import processing.serial.*;
2  import cc.arduino.*;
3  import org.firmata.*;
4
5  import oscP5.*;
6  import netP5.*;
7
8  Arduino arduino;
9  int sensorPin = 0;
10
11 OscP5 oscP5;
12 NetAddress myRemoteLocation;
13
14 void setup(){
15   arduino = new Arduino(this,Arduino.list()[1],57600);
16   arduino.pinMode(sensorPin, Arduino.INPUT);
17   size(800,100);
18
19   oscP5 = new OscP5(this,12000);
20   myRemoteLocation = new NetAddress("127.0.0.1",6666);
21 }
22
23
24 void draw(){
25
26   float val = arduino.analogRead(sensorPin);
27   println(5.17* val / 1023.);
28   background(0); fill(255,100,0); stroke(255);
29   rect(0,0,width * val/1024., height);
30
31   OscMessage myMessage = new OscMessage("/touch");
32   myMessage.add(int(val)); /* add an int to the osc message */
33   oscP5.send(myMessage, myRemoteLocation);
34 }
```

psketch_Touch.pdeにOSC通信のコードを追加します。

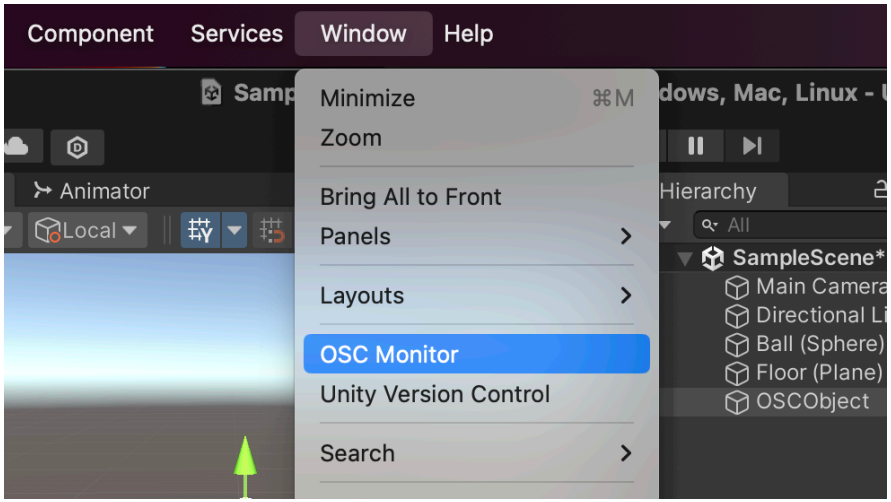
追記部分

UNITY側の準備

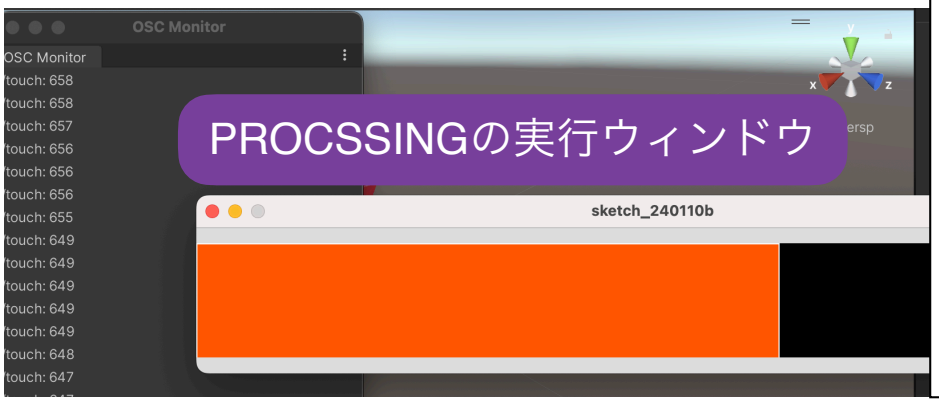
受信編

OSC Monitorによる受信状況の確認

この状態で、Window→OSC Monitorを開いてUnityを走らせる一方、Processing側も起動して、圧力センサを押すと、OSC Monitorに圧力センサの数値が届いていることがわかります。



OSC Monitor

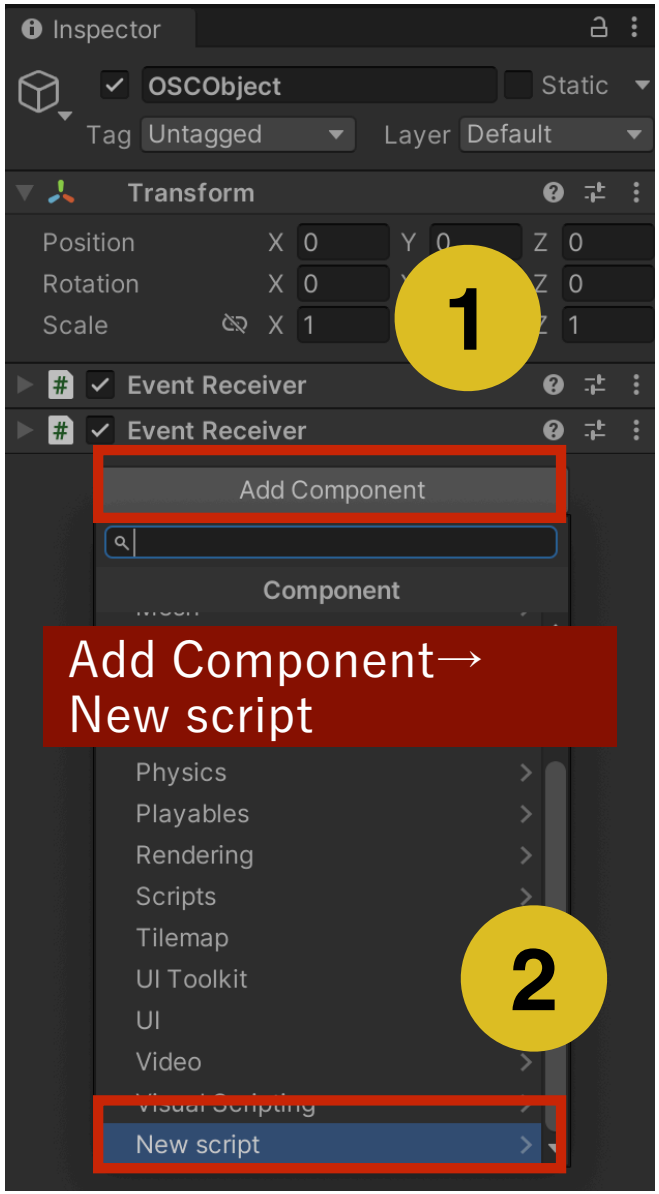


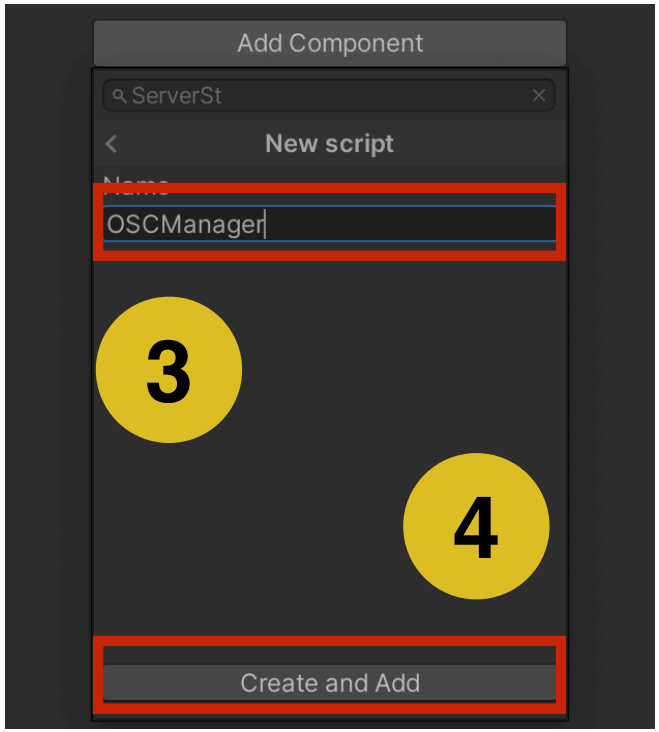
UNITY側の準備

受信編

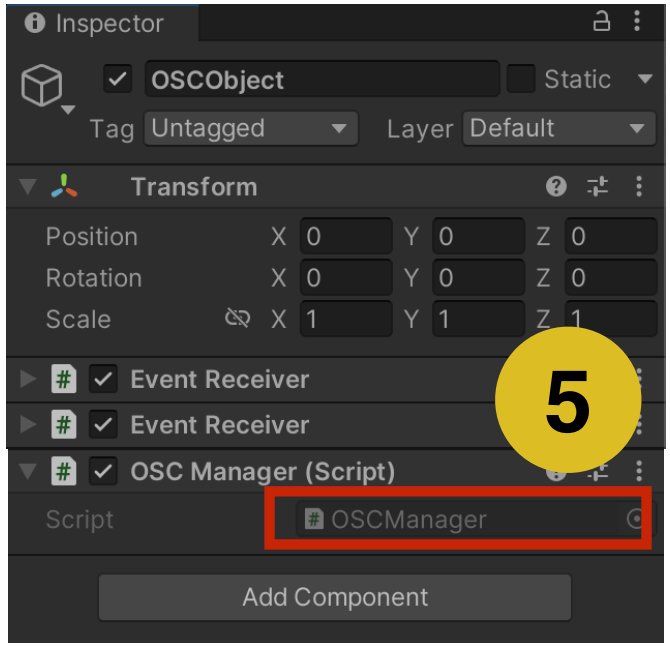
コールバック関数の作成と関連付け（1/3）

OSCObjectのコンポーネントとして、新しいスクリプトファイル（OSCManager）を作成します。





スクリプトの名前を「OSCManager」とし「Create and Add」して作成



新たに作成された、OSCManagerをダブルクリックして、編集エディタを開きます。

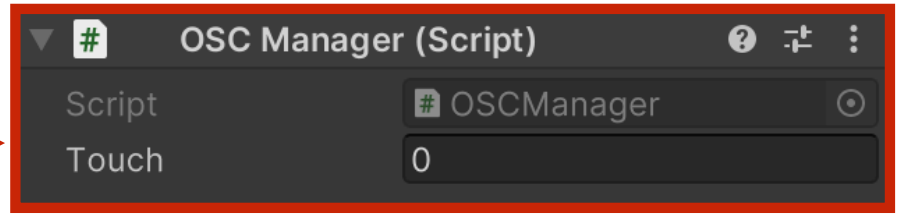
UNITY側の準備

受信編

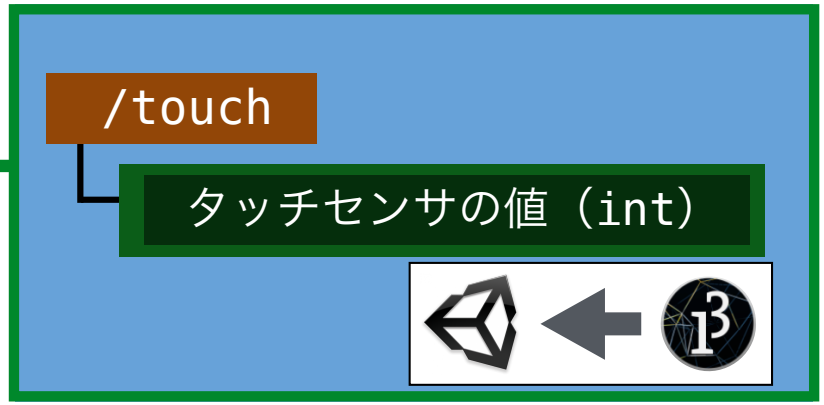
コールバック関数の作成と関連付け（2/3）

デフォルトのstart関数とupdate関数は消去してください。

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class OSCManager : MonoBehaviour
6   {
7
8       public int touch;
9
10      public void getTouch(int val){
11
12          touch = val;
13
14      }
15  }
```



public変数は、他のスクリプトファイルから参照可能になるとともに、インスペクタビューに表示され、リアルタイムに値を参照することができるようになります。



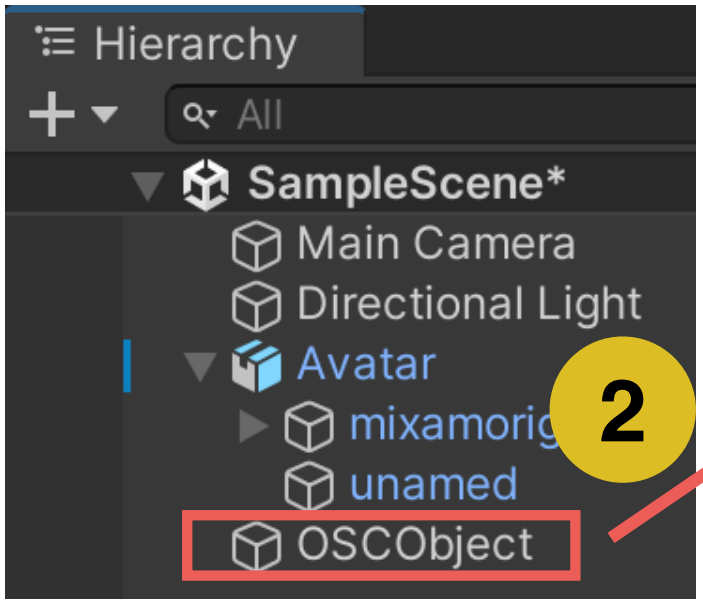
/touchを受け取るためのコールアップ関数を作成しておきます。引数がint型であることに注意。（おそらく複数の引数はとれない模様）

UNITY側の準備

受信編

コールバック関数の作成と関連付け（3/3）

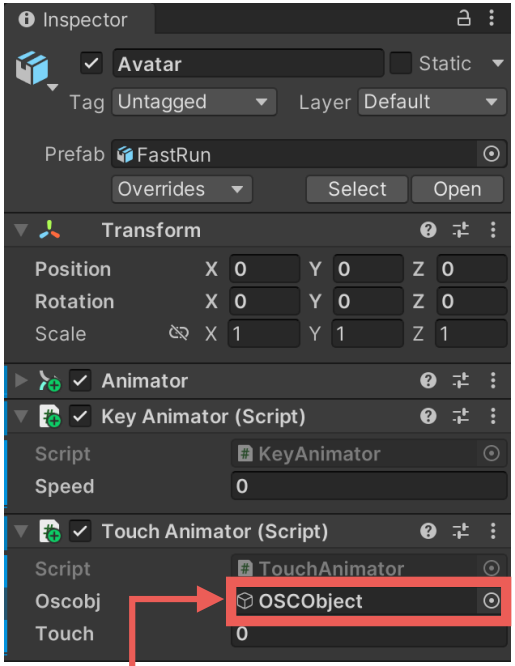
個別のOSCアドレスごとに、すでに作成済みのコールバック関数を対応させていきます。結果的に、圧力センサの値がOSC Managerの変数Touchに反映されることを確認してください。

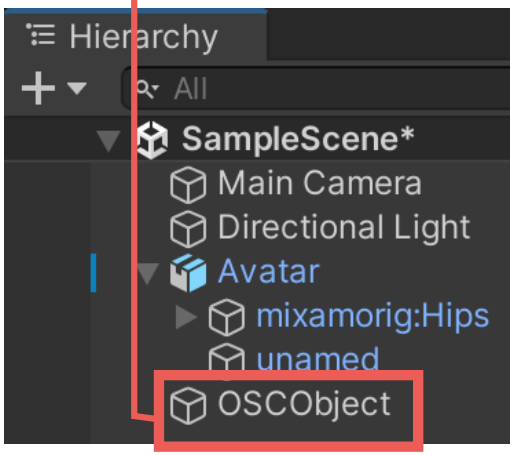


先ほど作成したコールバック関数の入っているゲームオブジェクトであるOSCObjectをアタッチします

Arduinoによるアニメーションの制御（一例）

Avatarのコンポーネントに、新たにTouchセンサの値を使って、アニメーションを制御するスクリプト「TouchAnimator」を生成します。






```
public class TouchAnimator : MonoBehaviour
{
    public GameObject oscobj = null;
    private KeyAnimator keya;
    private OSCManager oscmng;

    public int touch;
    Animator animator;
```

```
    void Start()
    {
        oscmng = (OSCManager)oscobj.GetComponent("OSCManager");
        keya = (KeyAnimator)this.GetComponent("KeyAnimator");
        animator = GetComponent<Animator>();
    }
```

```
    void Update()
    {
        touch = oscmng.touch;

        if (animator.GetCurrentAnimatorStateInfo(0).IsName("Flair")){
            if(touch>600){
                animator.SetInteger("aflag",1);
            }
        }

        if (animator.GetCurrentAnimatorStateInfo(0).IsName("FastRun")){
            float newspeed = keya.speed + 0.01f * (touch - 600f);
            newspeed = Mathf.Clamp(newspeed,0f,100f);
            keya.SetSpeed(newspeed);
        }
    }
}
```

Animationの状態が「Flair」のとき

touch>600で、aflagを1にセット

Animationの状態が「FastRun」のとき

touch=600を境に、KeyAnimatorのspeedを、0～100の範囲で増減させる。