

# 演習：フィジカル・コンピューティング（後半）

12/11 3限・4限

## 演習5 | UnityのアニメーションとArduinoの接続

12/18 3限・4限

### 自由課題（休講）

- 余裕があればオンデマンド補講（MediaPipe / MAX）

01/08 3限・4限

### 自由課題

01/22 3限・4限

### 講評会（課題提出）

# MIXAMよりFBXのダウンロード

The screenshot shows the Mixamo website interface. At the top, there's a navigation bar with the Adobe logo, the Mixamo logo, and links for 'Characters' and 'Animations'. A user profile 'Kenri' is also visible. Below the navigation is a search bar with the placeholder 'Search' and a magnifying glass icon. To the right of the search bar are dropdown menus for '48 Per page' and settings. The main content area displays search results for 'Fast Run', showing 1-48 of 55 results. The results are presented in a grid format with three columns. The first column contains three thumbnails: 'Fast Run' (red), 'Run To Stop', and 'Running To Turn'. The second column contains two thumbnails: 'Two' and 'Jumping'. The third column shows a large preview image of a male character in mid-stride during a fast run. To the right of the preview is a control panel with a 'DOWNLOAD' button and options for 'SEND TO AERO' and 'UPLOAD CHARACTER'. Below these are sliders for 'Overdrive' (set to 50), 'Character Arm-Space' (set to 50), and 'Trim' (set to 17 total frames, with a range from 0 to 100). There are also checkboxes for 'Mirror' and 'In Place'. At the bottom left, there are two more thumbnails: 'Flair' (red) and 'Jumping' (blue). A yellow banner at the bottom displays the URL <https://www.mixamo.com>. A dark purple box on the right side contains the text: '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。' (Create your own avatar's OBJ file and create 3 pattern FBX files, then download them). At the bottom right, there's a folder icon labeled '\_mixamo' with a blue arrow pointing right, and three file icons labeled 'FastRun.fbx', 'Flair.fbx', and 'Jumping.fbx'.

<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。

\_mixamo

FastRun.fbx

Flair.fbx

Jumping.fbx

# MIXAMよりFBXのダウンロード

The screenshot shows the Mixamo character creation interface. On the left, a 3D model of a character in a dynamic pose is displayed against a grid background. To its right is a smaller preview window showing a red humanoid model in a similar pose. Below the preview window is a button labeled "Fast Run". On the far right, there is a control panel with several settings:

- A large orange "DOWNLOAD" button.
- A "SEND TO AERO" button.
- An "UPLOAD CHARACTER" button.
- A section titled "Fast Run" with an "X" icon to close it.
- A slider for "Overdrive" set to 50.
- A slider for "Character Arm-Space" set to 50.
- A slider for "Trim 17 total frames" ranging from 0 to 100, with markers at 0 and 100.
- A checkbox for "Mirror" which is unchecked.
- A checkbox for "In Place" which is checked and highlighted with a red border.
- A red double exclamation mark "!!" located next to the "In Place" button.

**Fast Runのときに、In Placeにチェックを入れるようにしてください。位置はUnityの中で独立に制御することを想定しています。**

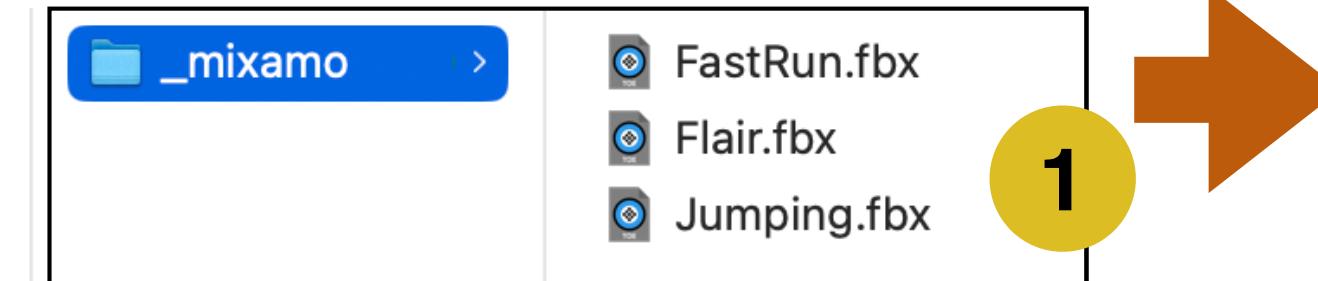
# UNITYへの移植

1

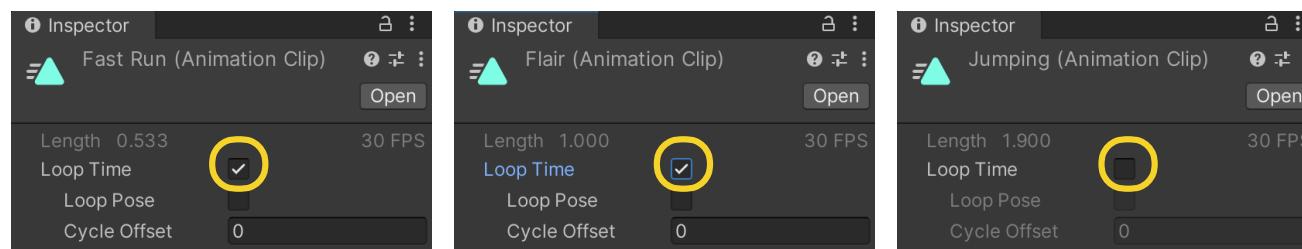
新しく作成したUNITY新規ファイルのプロジェクトビューに、3つのFBXファイルを放り込みます。

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Alt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。

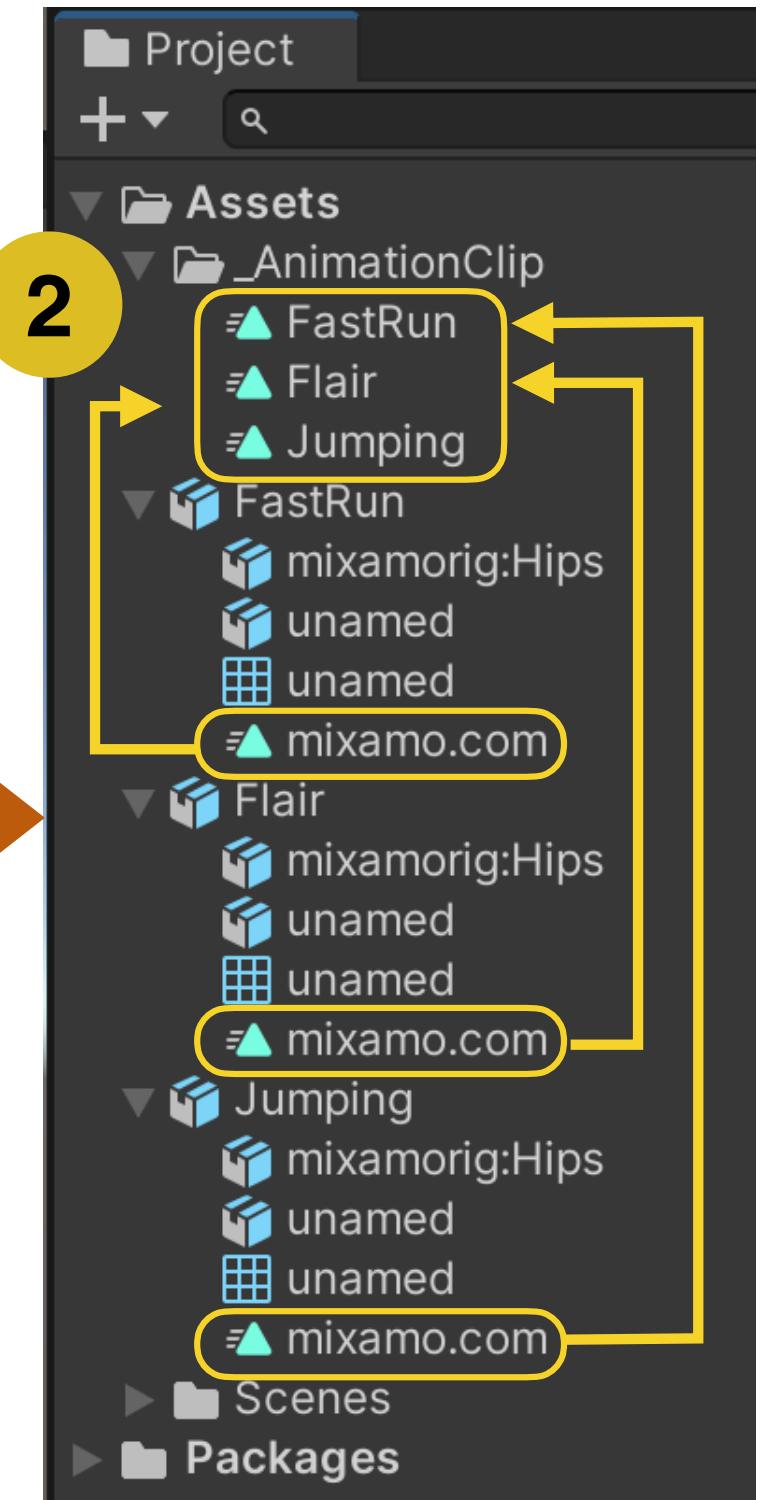


1



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）。



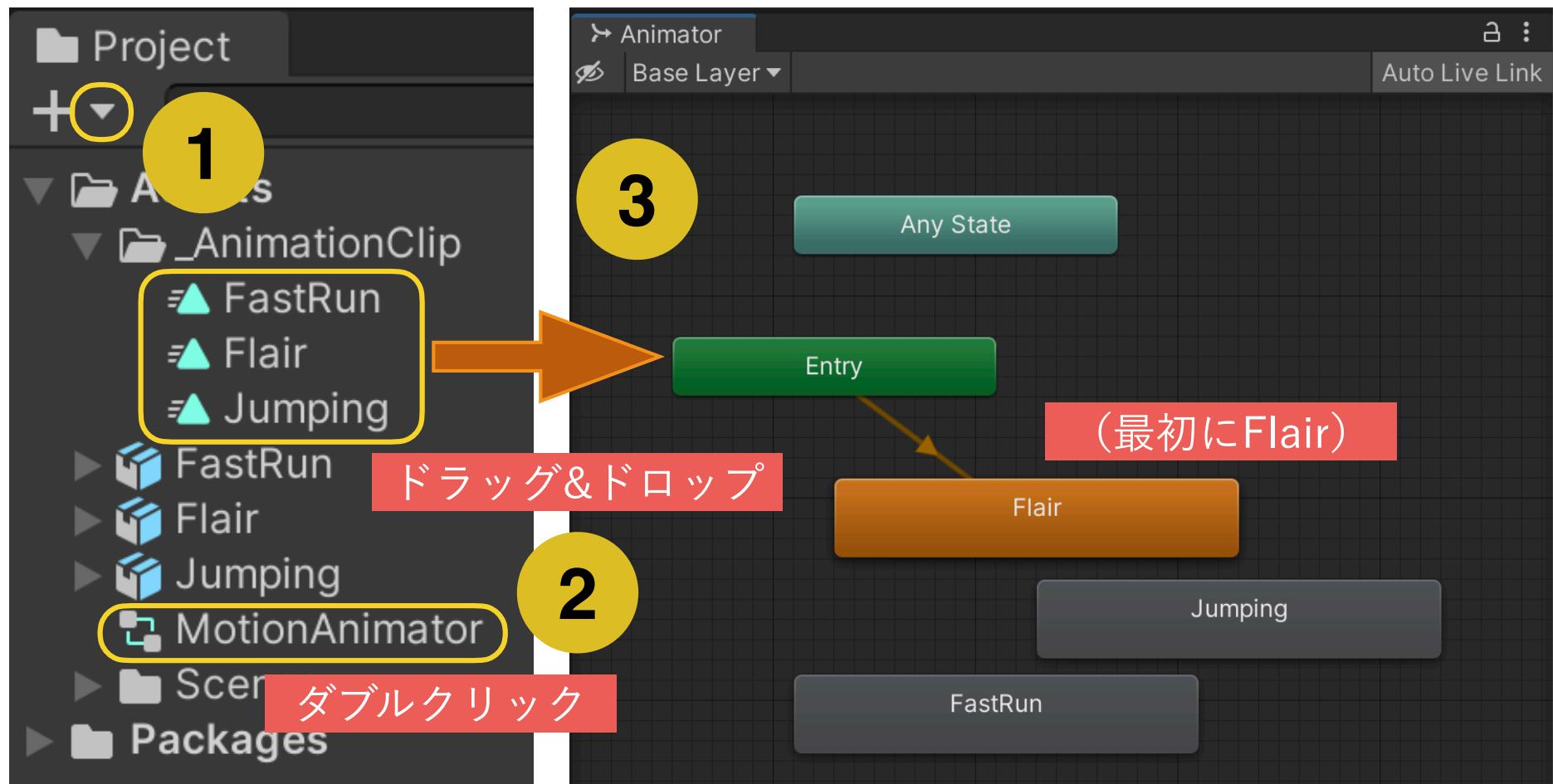
# Animation Controllerの作成

1

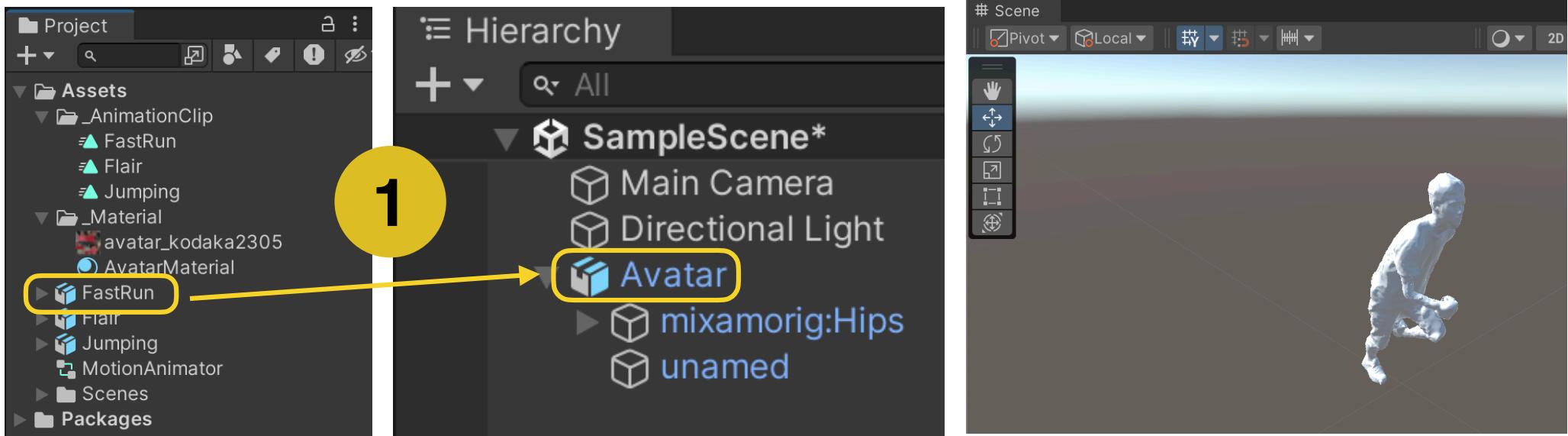
プロジェクトタブより「Animation Controller」を新規作成し、適当に名前を変えます。

23

新規作成したアニメーターをダブルクリックしてAnimator View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

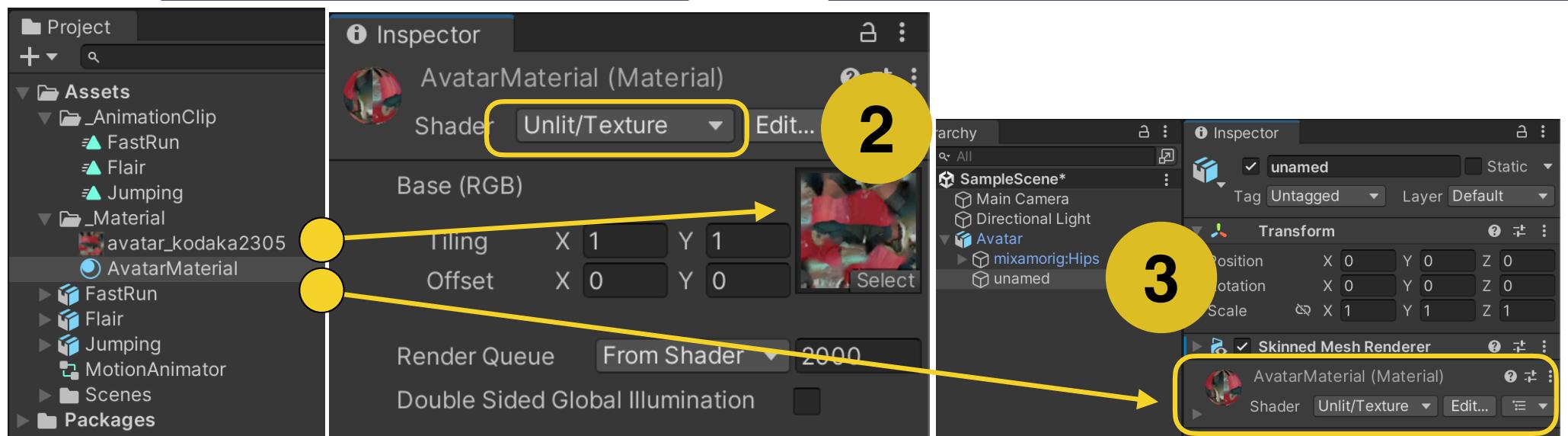


1

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

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。

12

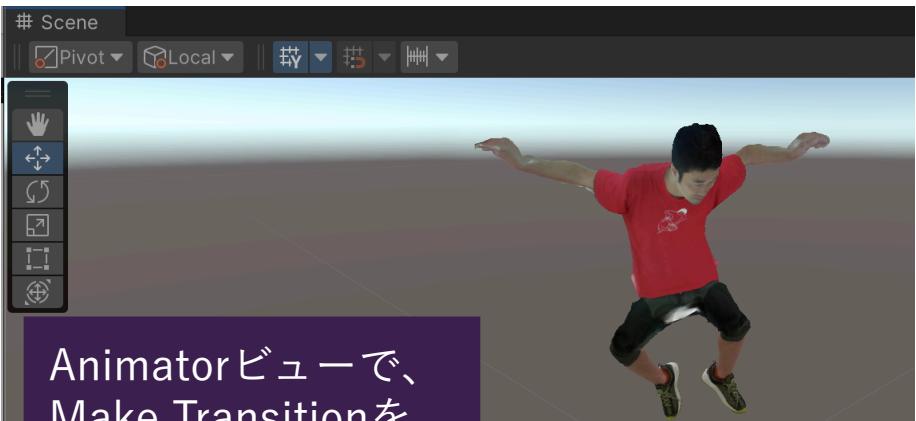
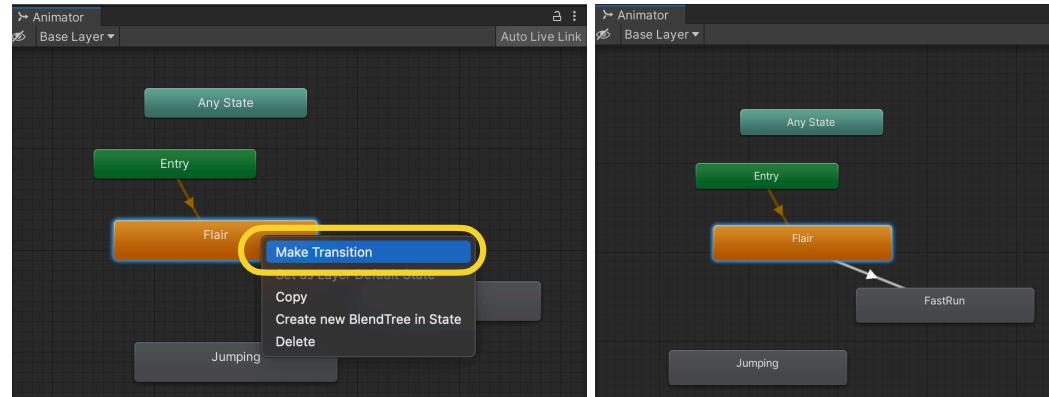
1

2

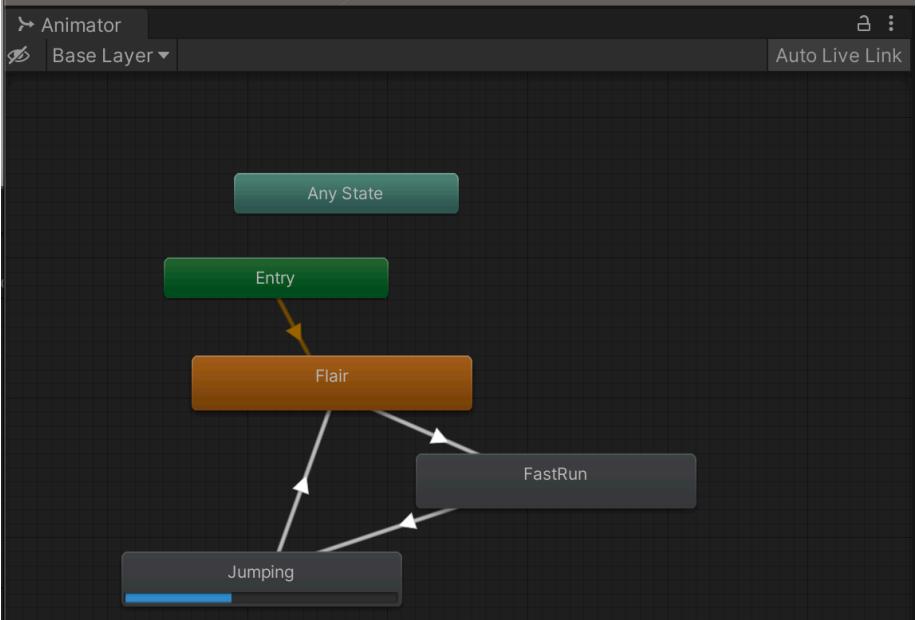
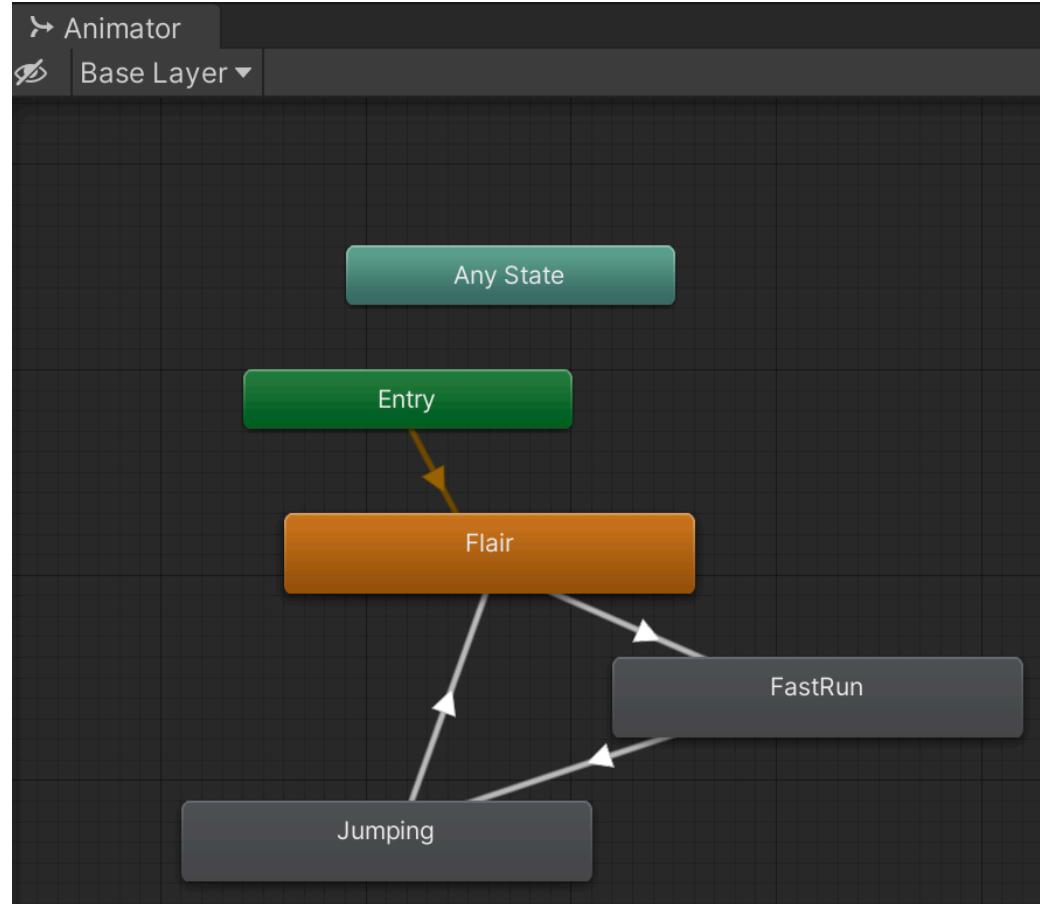
3

実行すると Flair のアニメーションを踊り続けます。

# 複数のアニメーションクリップを遷移させる



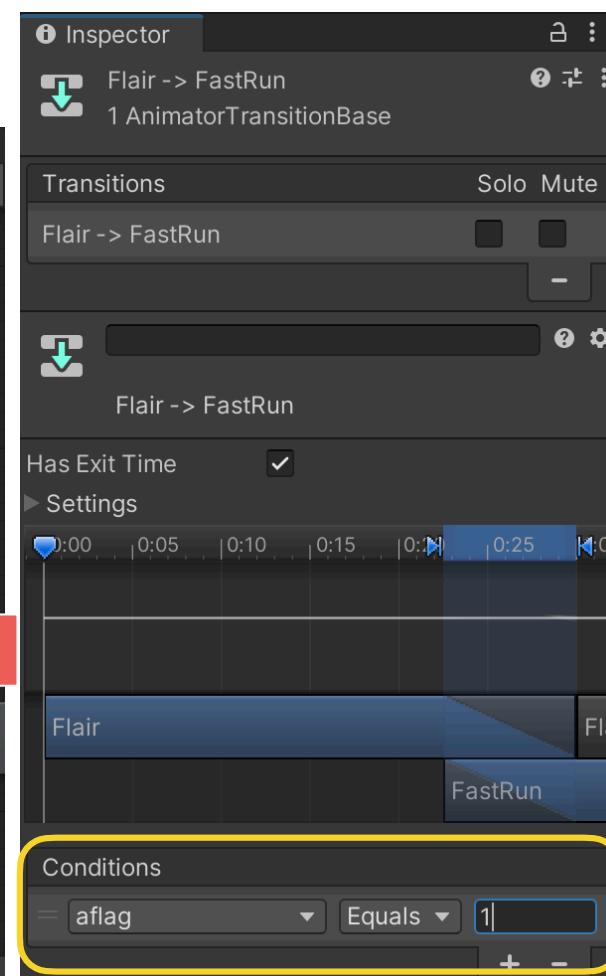
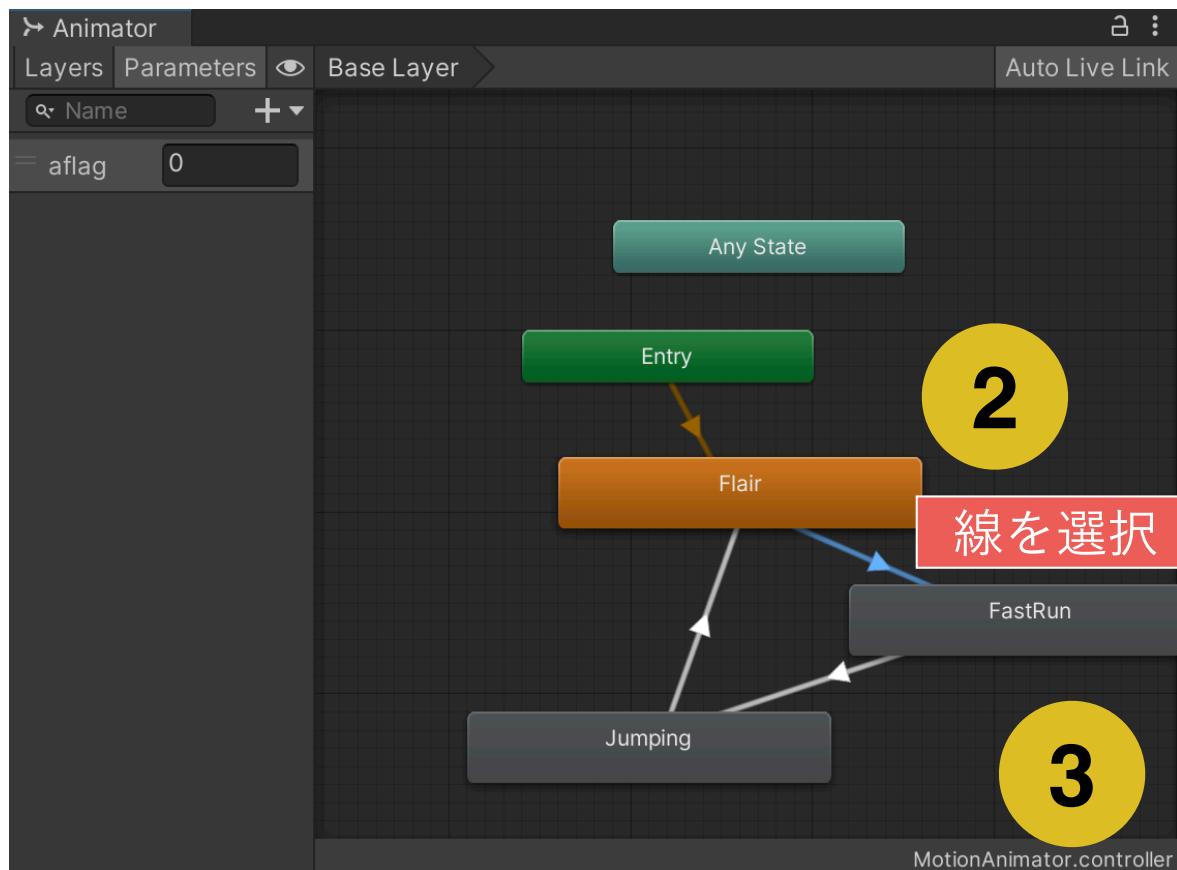
Animatorビューで、  
Make Transitionを  
図のように定義して  
実行すると、3つの  
クリップを巡回して  
いきます。



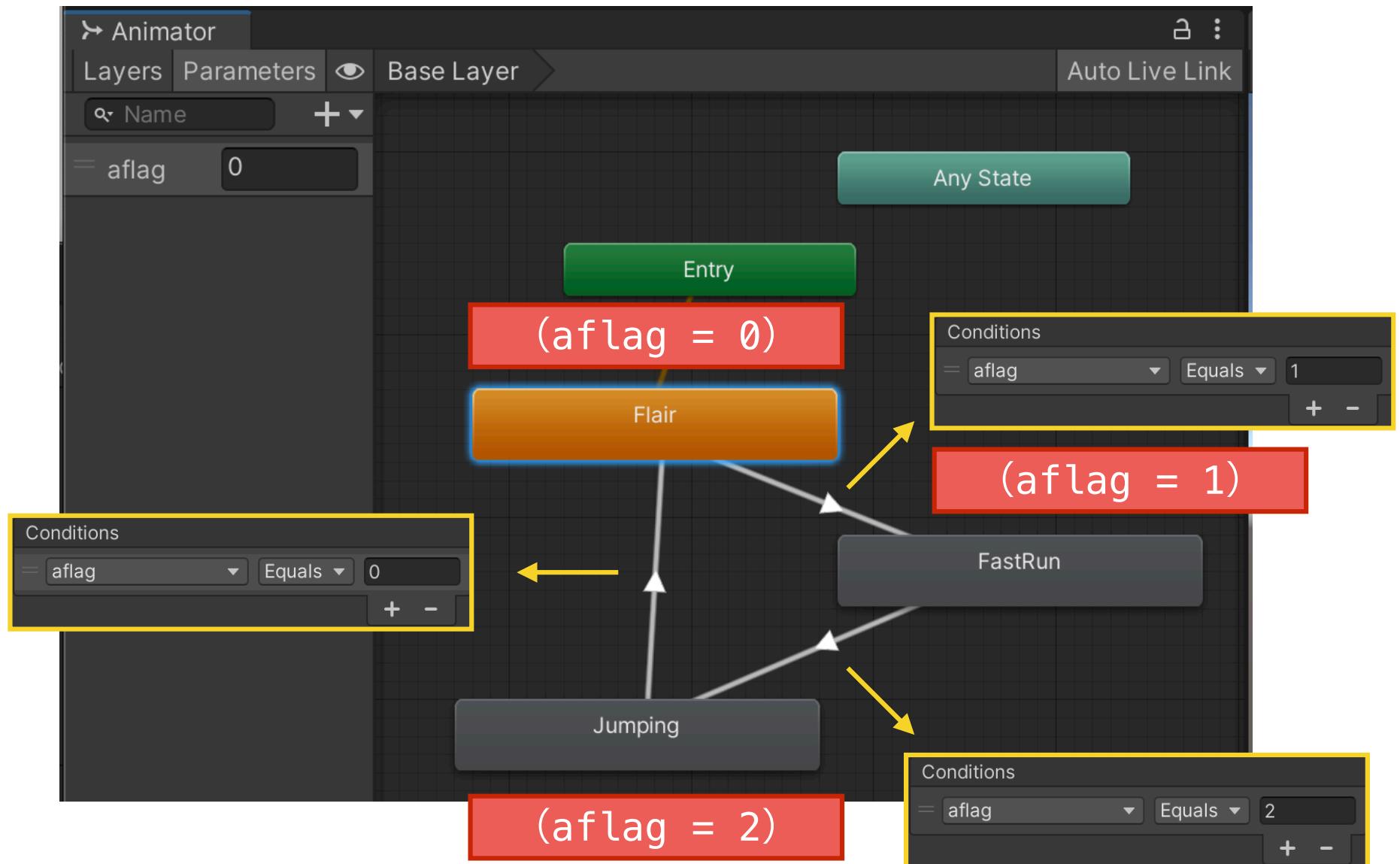
# トランジションを変数で駆動する (1/2)



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へのトランジションが生じることはありません。

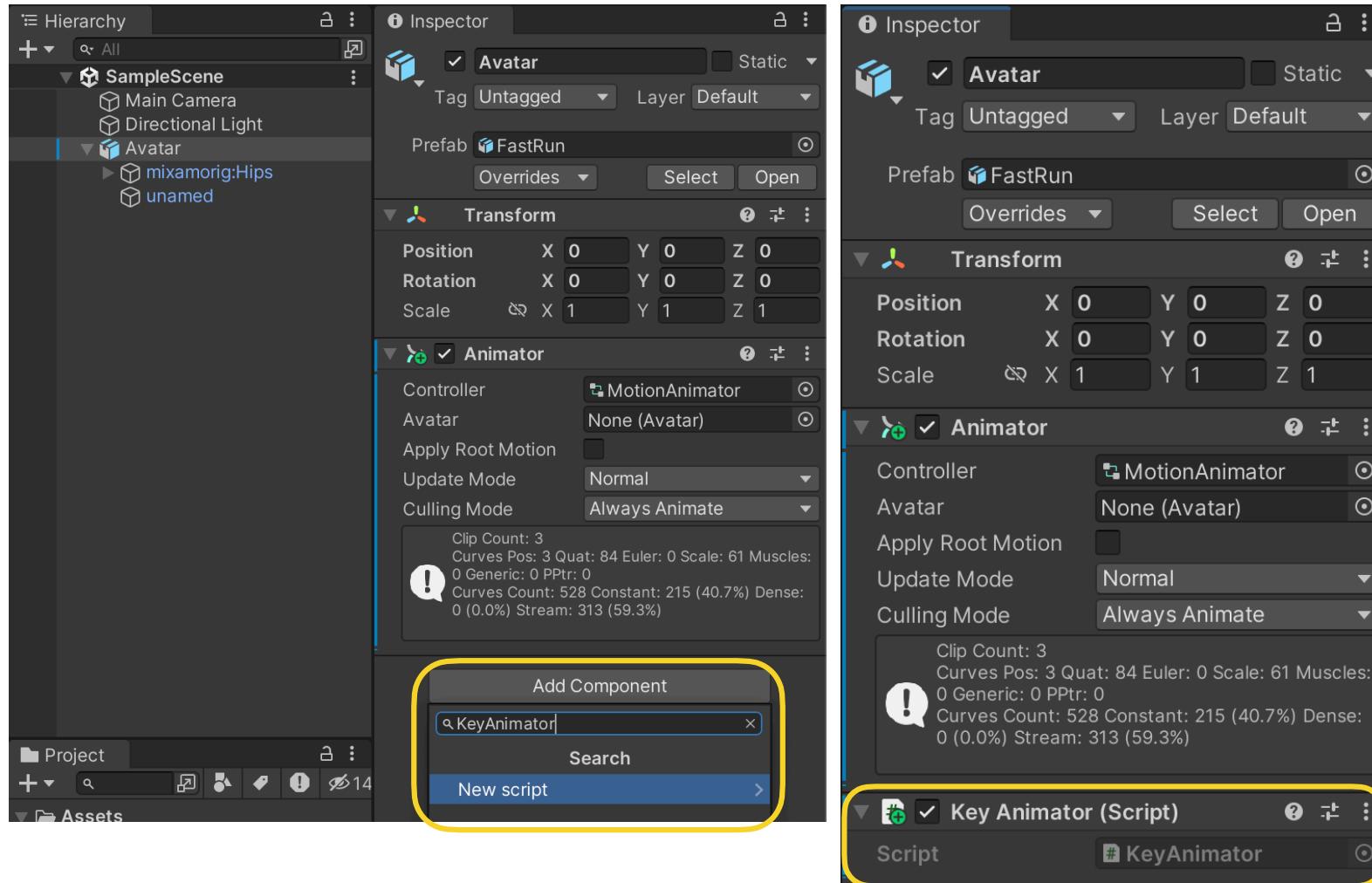


# トランジションを変数で駆動する (2/2)



同様に3つのトランジションが、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変数としてspeedを定義
8      public float speed = 0;
9      Animator animator;
10 }
```

```
12 void Start() animatorコンポーネントを取得
13 {
14     animator = GetComponent<Animator>();
15     animator.SetInteger("aflag",0);
16 }
起動時にaflagをゼロにセット
```

## Start()

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

## Update() 1/3

KEY上ボタンが押されると、  
aflagを1にセット

## Update() 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
46 }
```

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

```
46
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

## Update() 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によるトランジションの制御 (3/4)

KeyAnimator.cs

Animationの状態が「FastRun」のとき

Update() 2/3

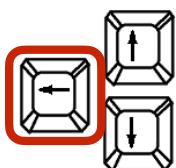
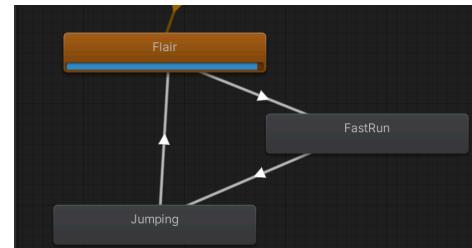
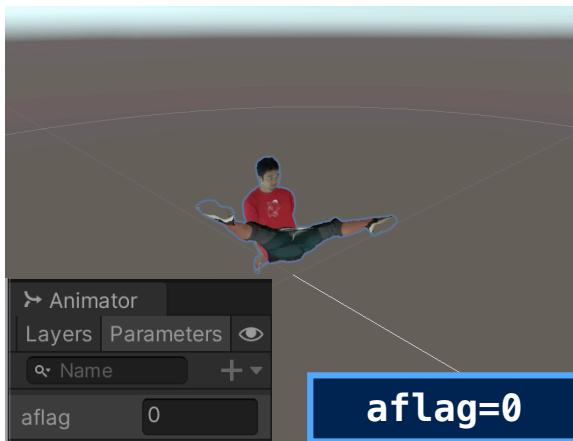
```
if(animator.GetCurrentAnimatorStateInfo(0).IsName("FastRun")){  
  
    if(Input.GetKeyDown(KeyCode.UpArrow)){  
        speed = Mathf.Min(speed+1,20);  
  
        if(speed>=20){  
            speed = 0;  
            animator.SetInteger("aflag",2);  
        }  
    }  
    if(Input.GetKeyDown(KeyCode.DownArrow)){  
        speed = Mathf.Max(speed-1,0f);  
    }  
}
```

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。

```
Vector3 pos0 = this.transform.position;  
Vector3 pos1 = new Vector3(pos0.x,pos0.y,pos0.z+speed*0.01f);  
this.transform.position = pos1;  
animator.speed = 1 + speed * 0.05f;
```

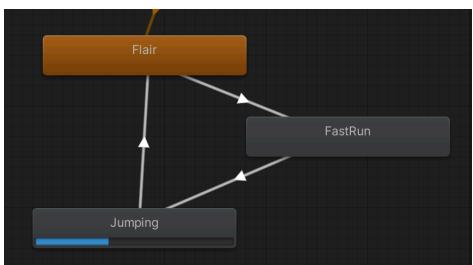
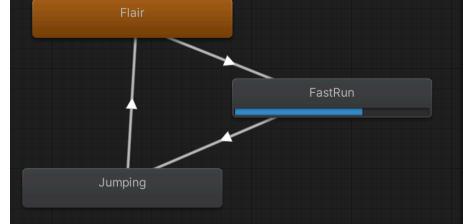
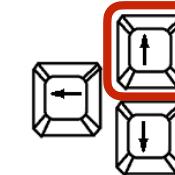
# KEYによるトランジションの制御 (4/4)

Flair

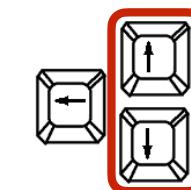


speed = 0;

FastRun



speedが20以上とな  
った時

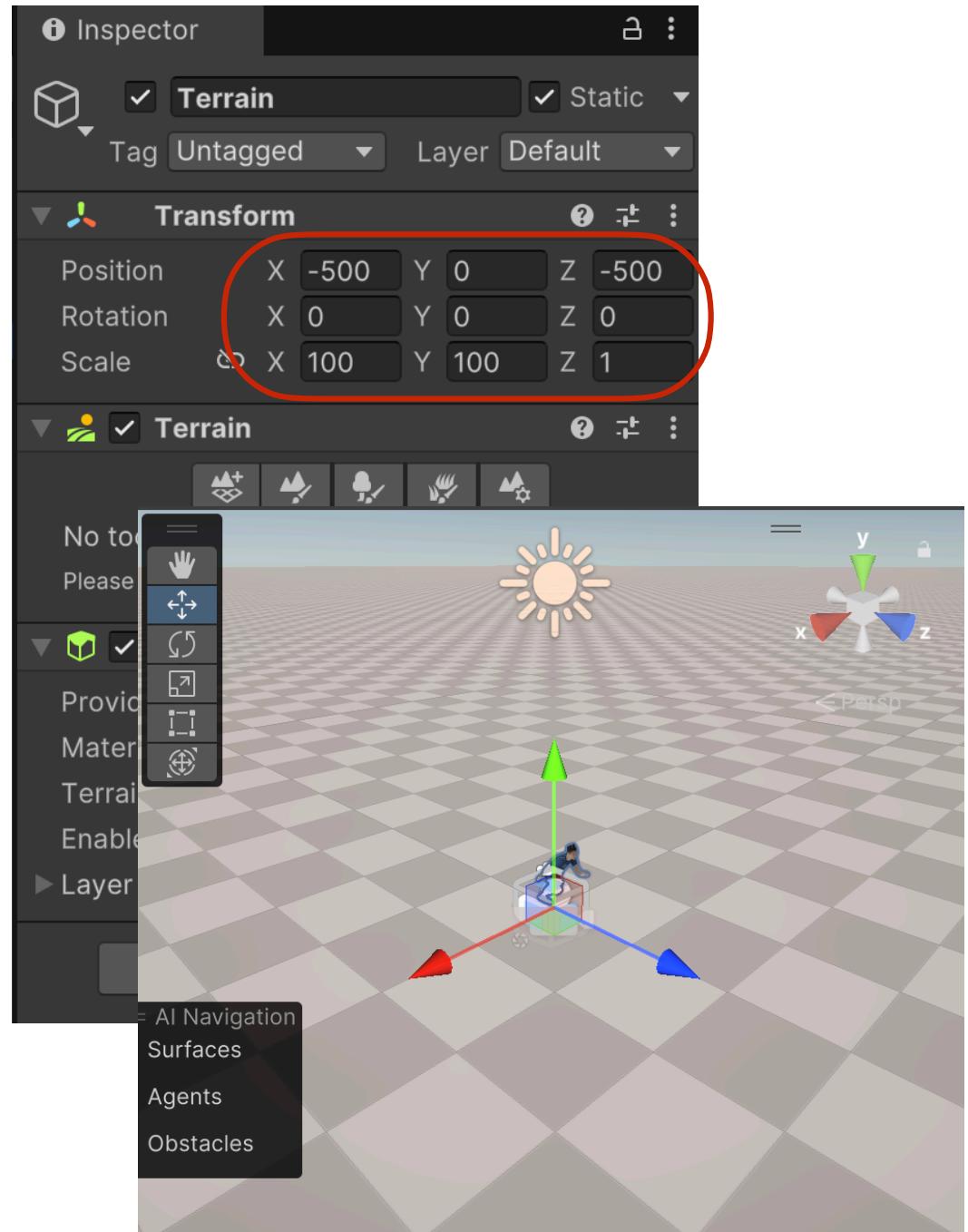
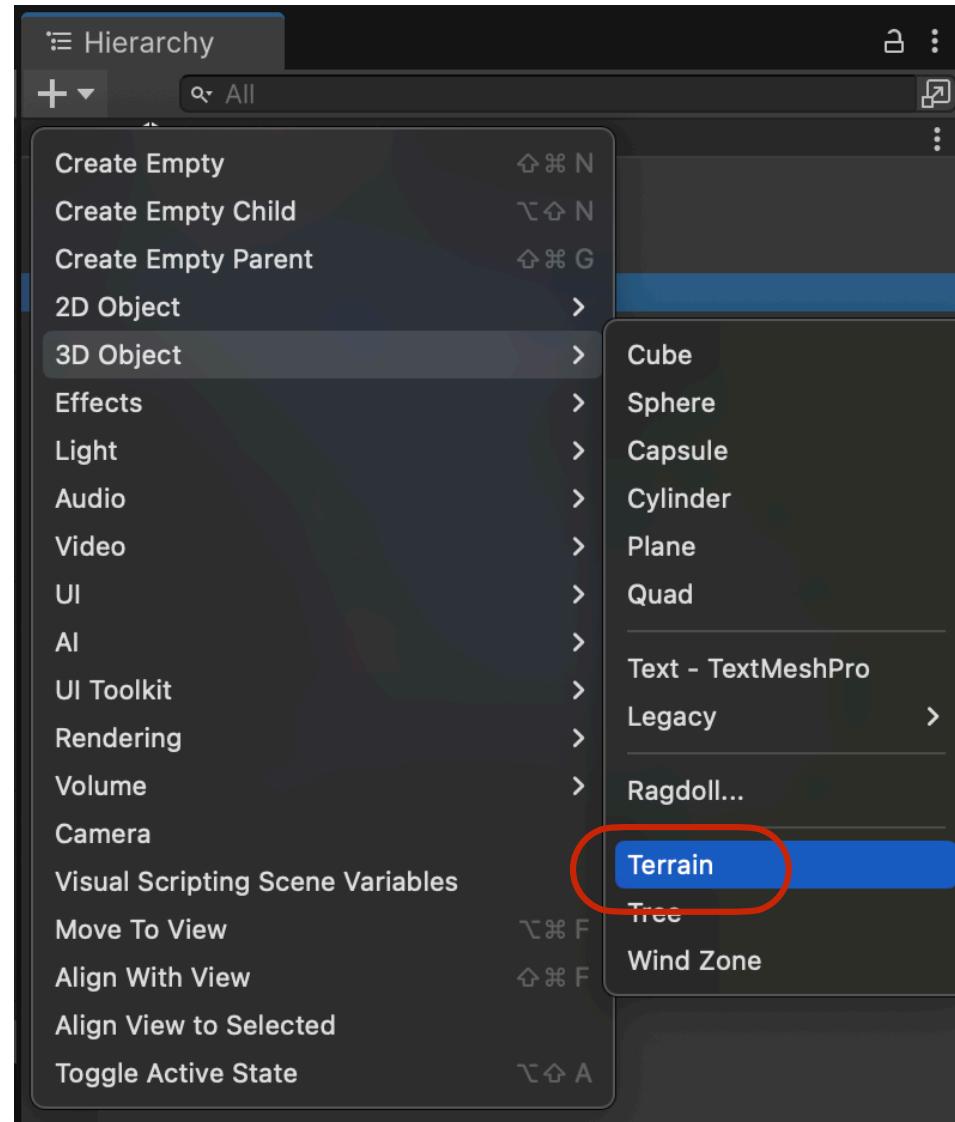


speed++;

speed-;

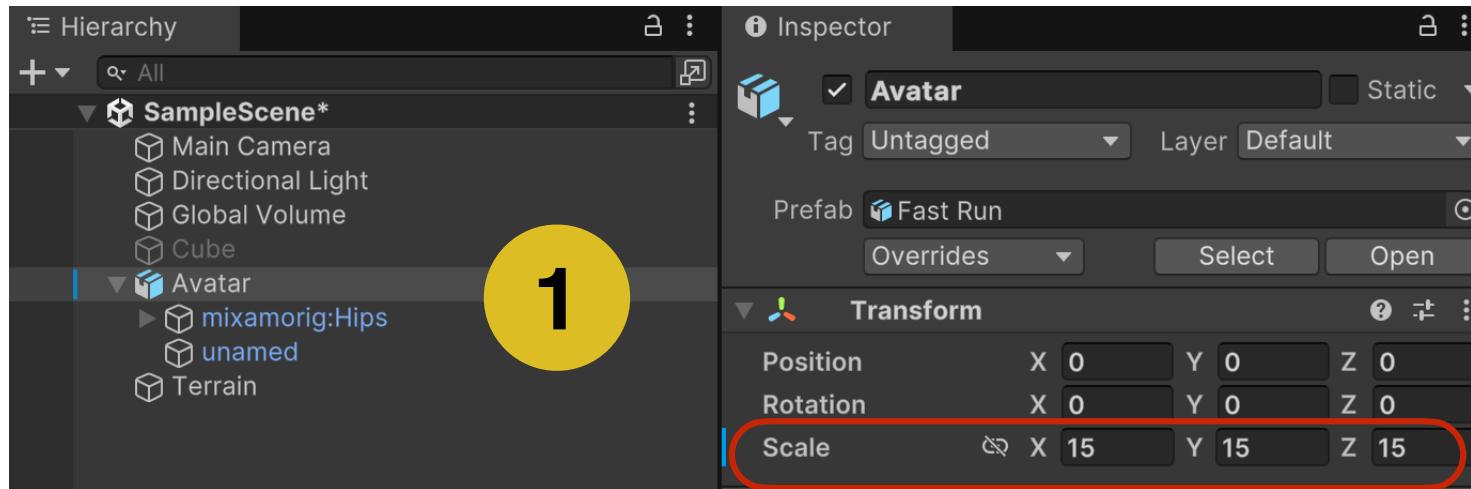
Jumping

# 補足 1 (床の配置)

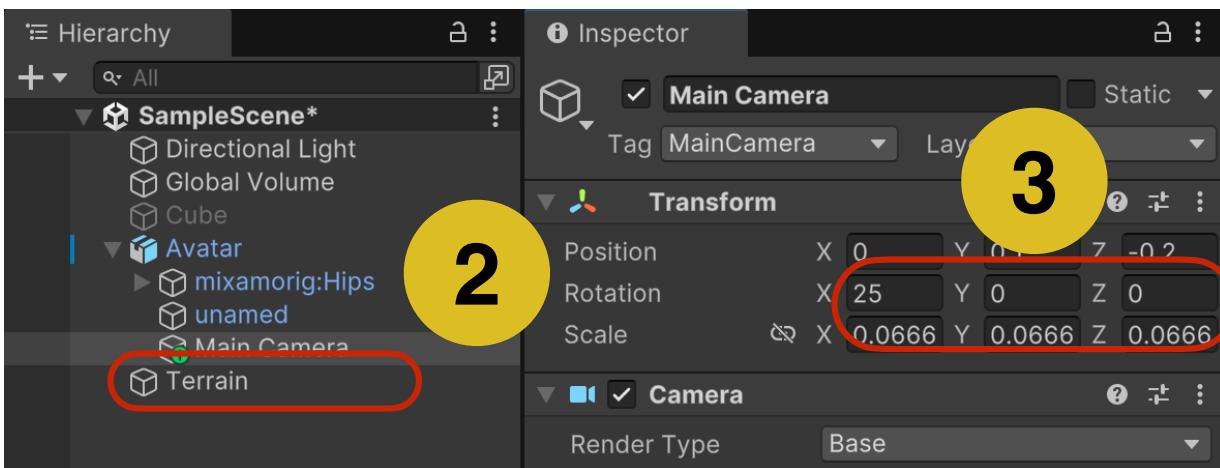


デフォルトで用意されているグリッドパターンの床を適当な位置に配置しておきます。

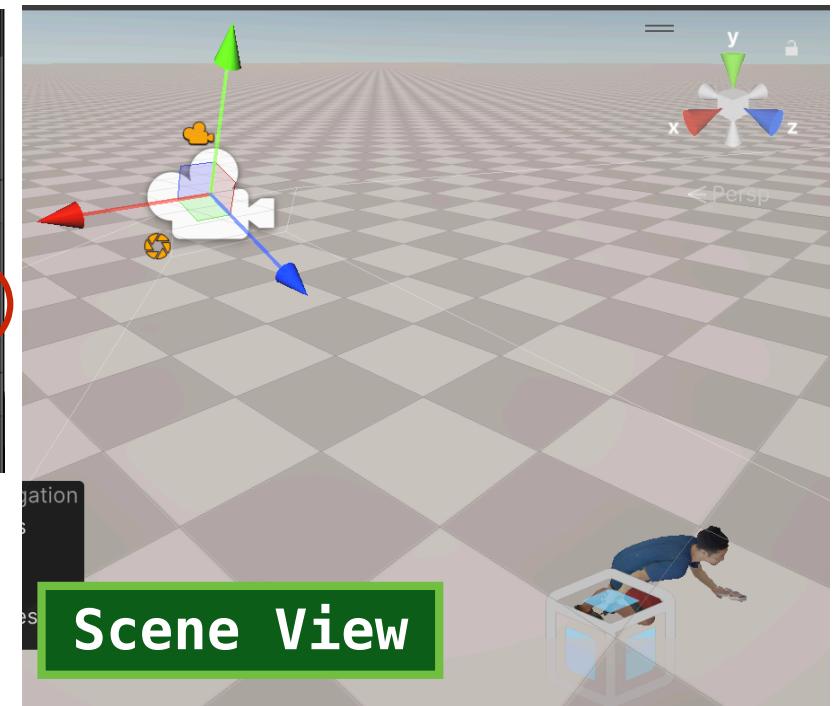
## 補足2（アバターのスケール、カメラの位置）



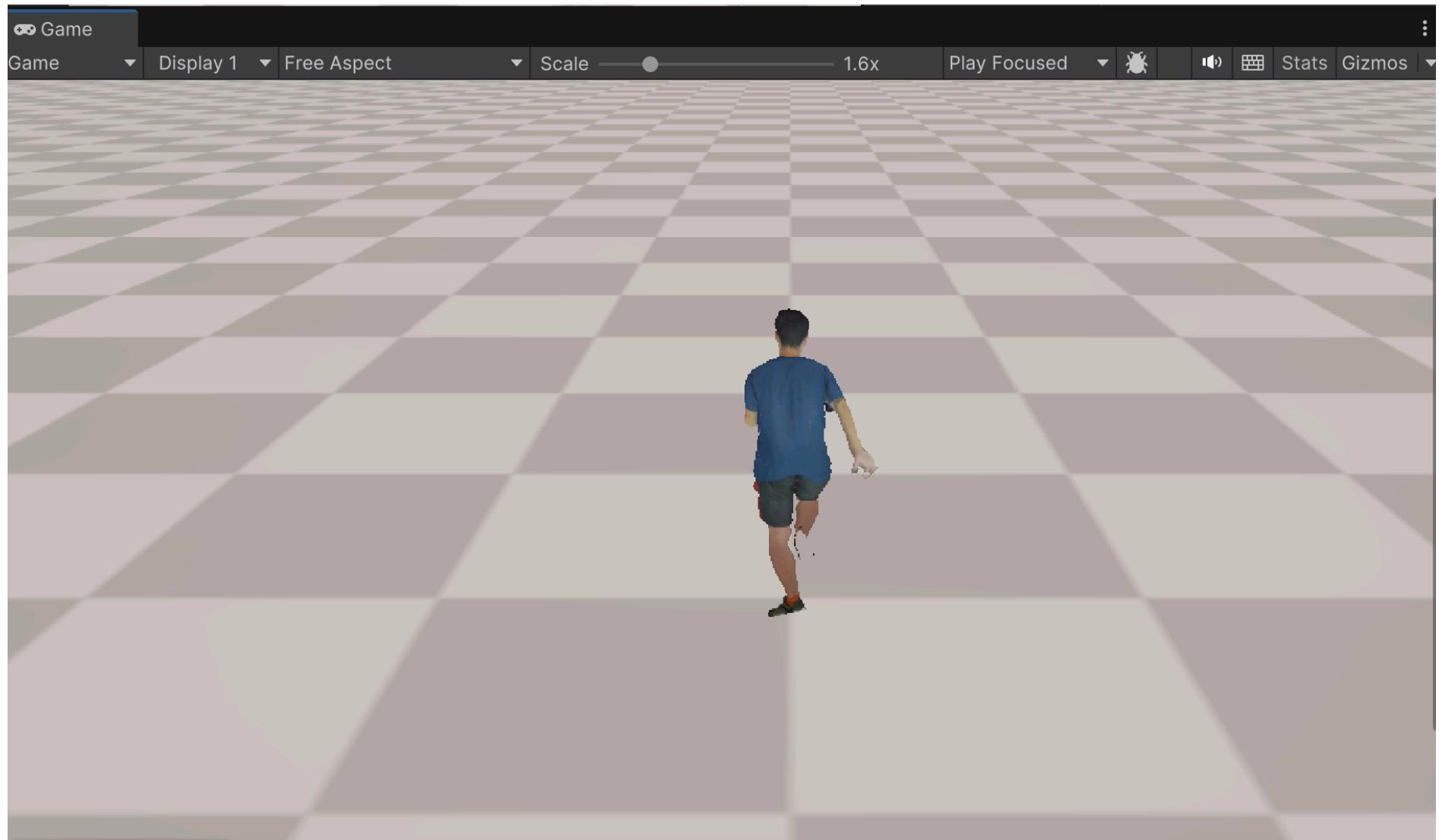
アバターの背の高さが適切となるようにスケールを変えます（mixamoのデータは10cm?）



カメラをアバターの子オブジェクトとして、アバターを常に見下ろせるようにPositionとRotationを調整します。



## 補足2（アバターのスケール、カメラの位置）



ゲームビューでは常に背後からアバターの動きを追うようなレイアウトとなります。

Game View

OSC環境の設定（復習です）

# UNITY側の準備

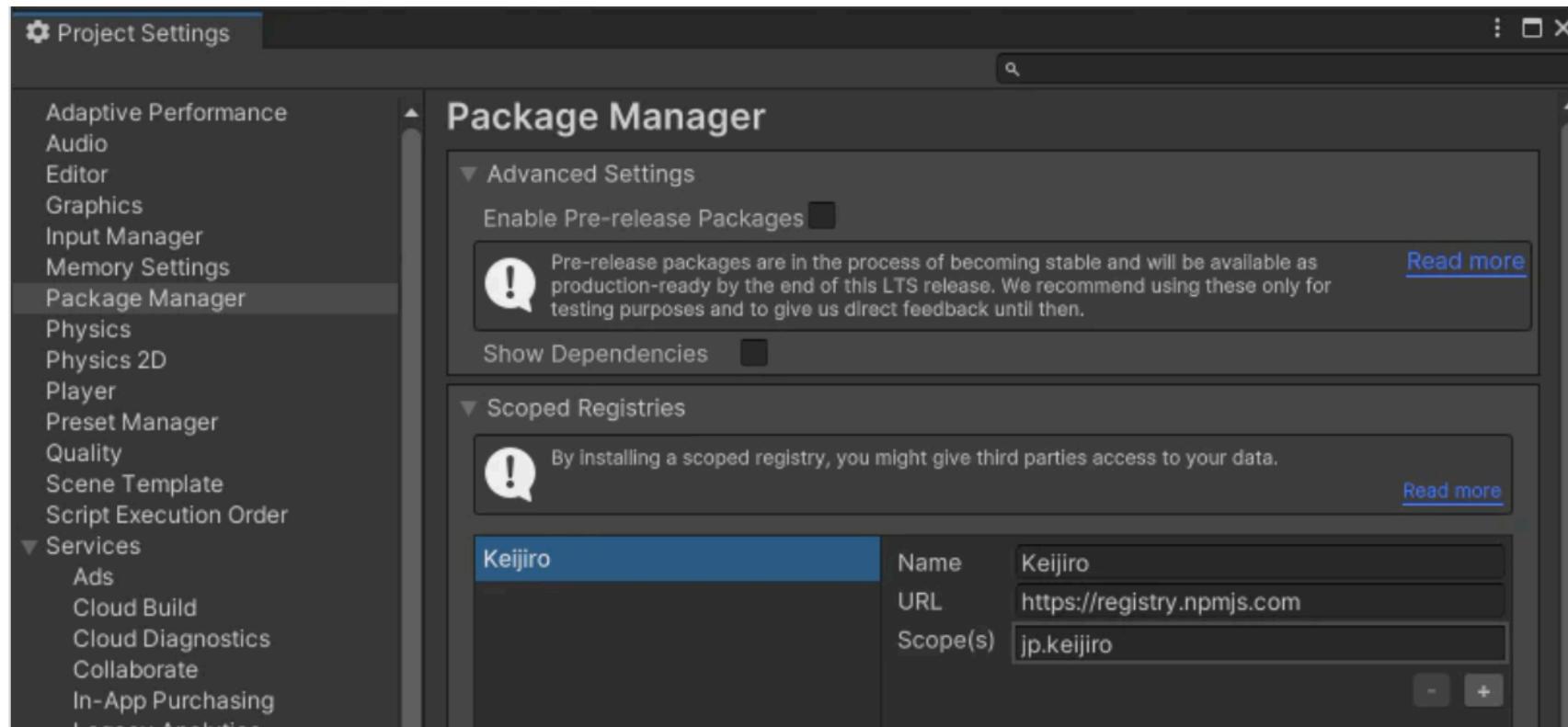
## OscJackのインポート (1/2)

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

Name : Keijiro

URL : <https://registry.npmjs.com>

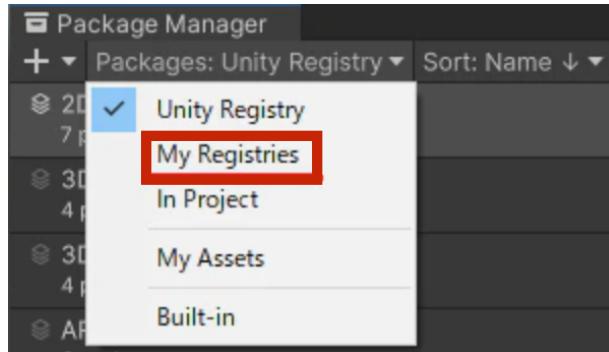
Scope(s) : jp.keijiro



# 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。

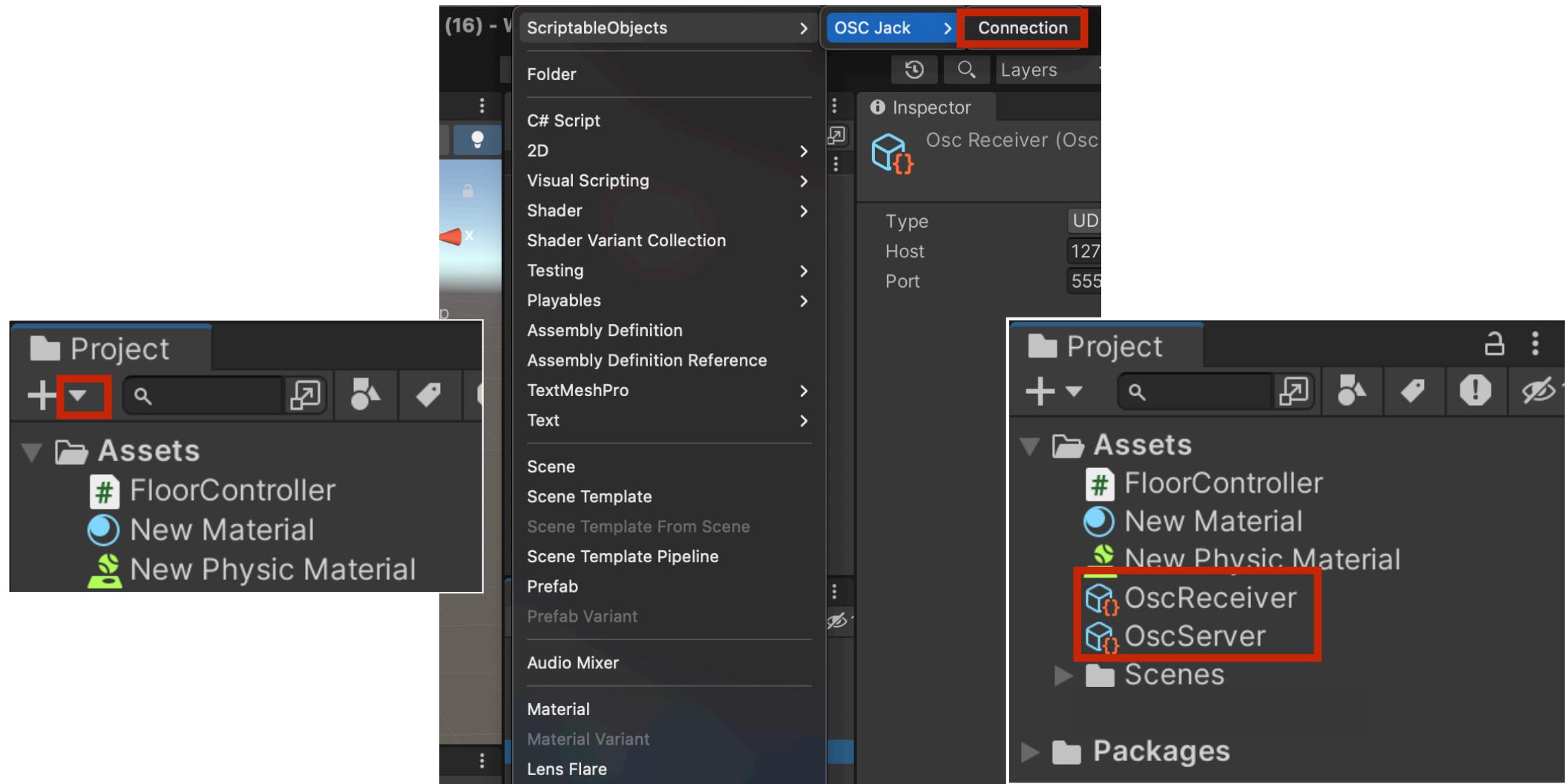
The Unity Package Manager window is shown again, but now the 'My Registries' tab is selected in the top bar. The left sidebar lists several packages: M-LSD, NeoLowMan, NNUtils, Noise Shader, OSC Jack, OscJackVS, PCX, Procedural Motion Library, Procedural Motion Track Library, Pugrad, and RtMidi. The 'OSC Jack' package is highlighted with a green checkmark next to its version number. On the right, the detailed view for 'OSC Jack' is displayed. It shows the version 2.0.0 (released April 25, 2022), the developer 'Keijiro by Keijiro Takahashi', and the GitHub repository 'jp.keijiro.osc-jack'. It also includes links for 'Documentation', 'Changelog', and 'Licenses'. A note at the bottom states 'This package is hosted on a Scoped Registry.' and has a 'Read more' link. At the very bottom of the window, the text 'Last update Dec 19, 11:06' is visible.

# UNITY側の準備

## ポート情報を管理するConnectionの設定

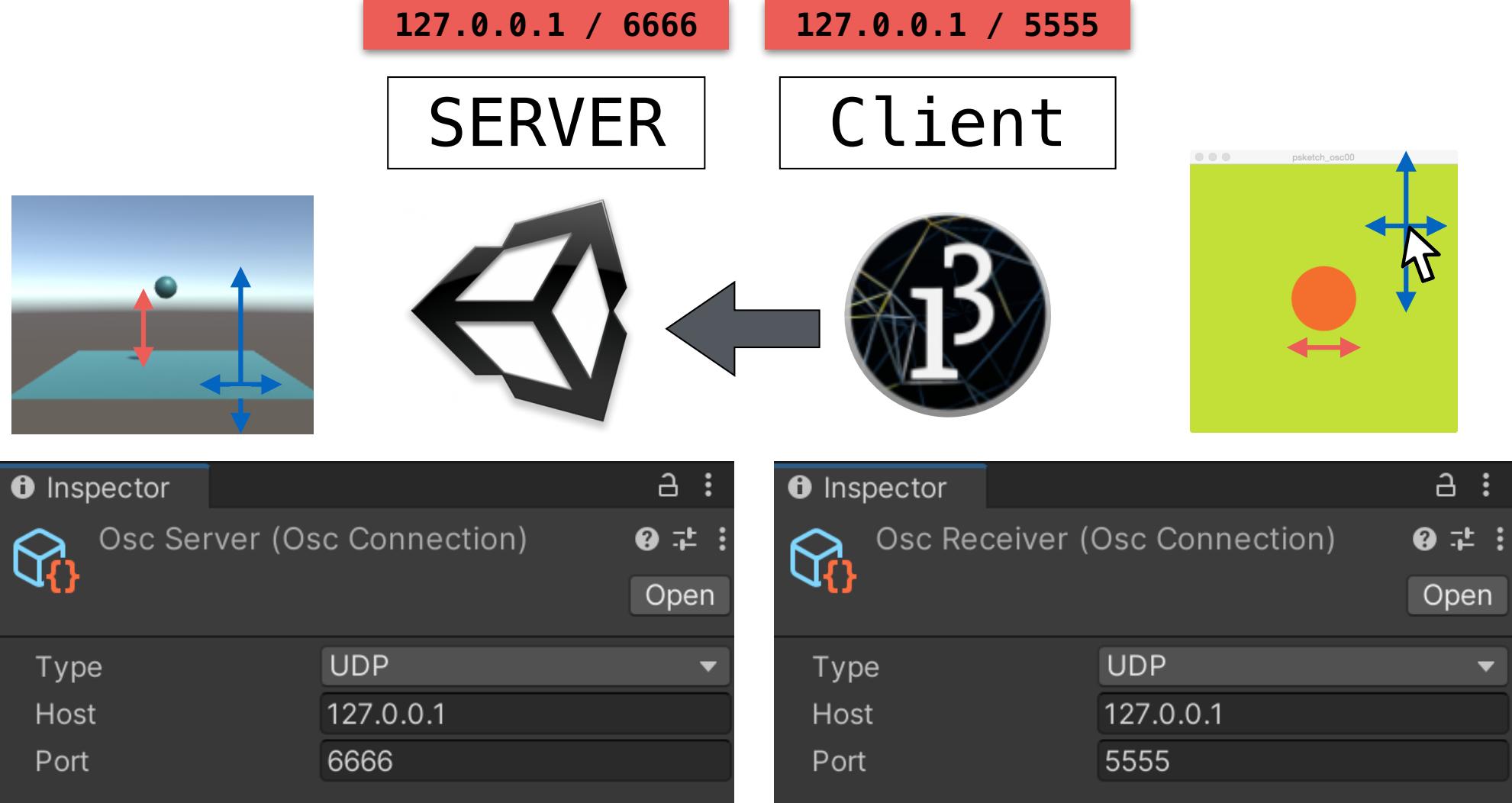
Projectタブ左上▼→ScriptableObjects→OSC Jack→Connection

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。

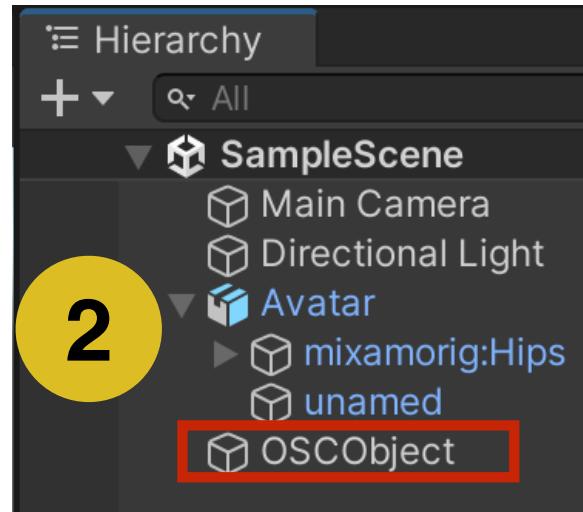
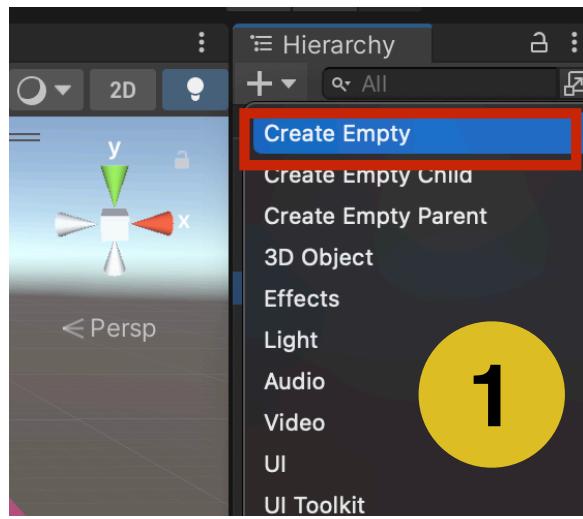


# UNITY側の準備

受信編

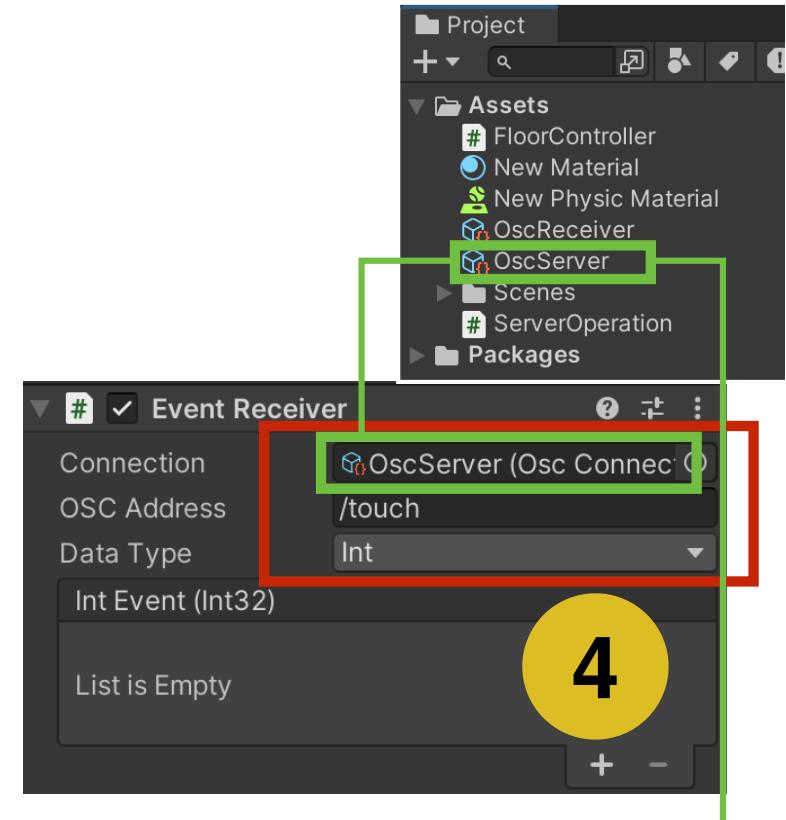
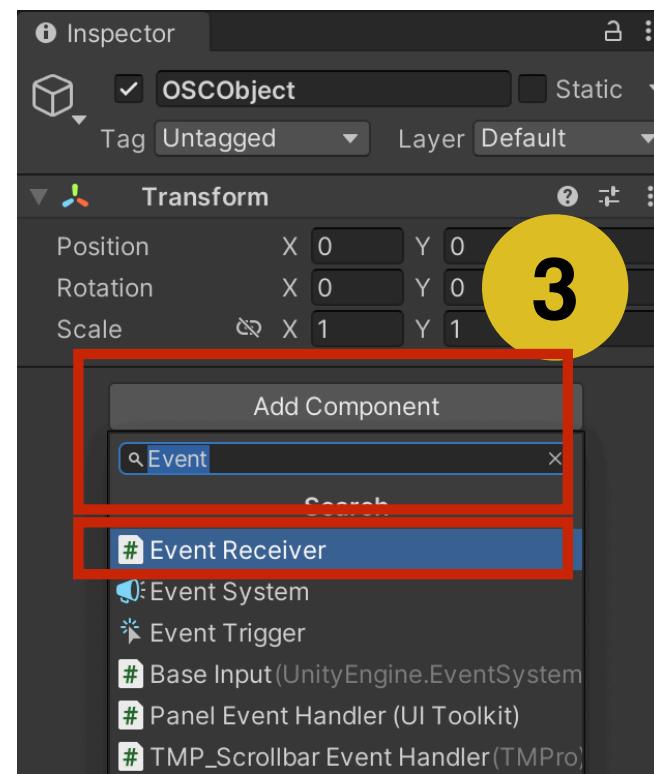


## EventReceiverの設定



空のゲームオブジェクトを作成し、**OSCOBJ**とリネームする。

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。



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

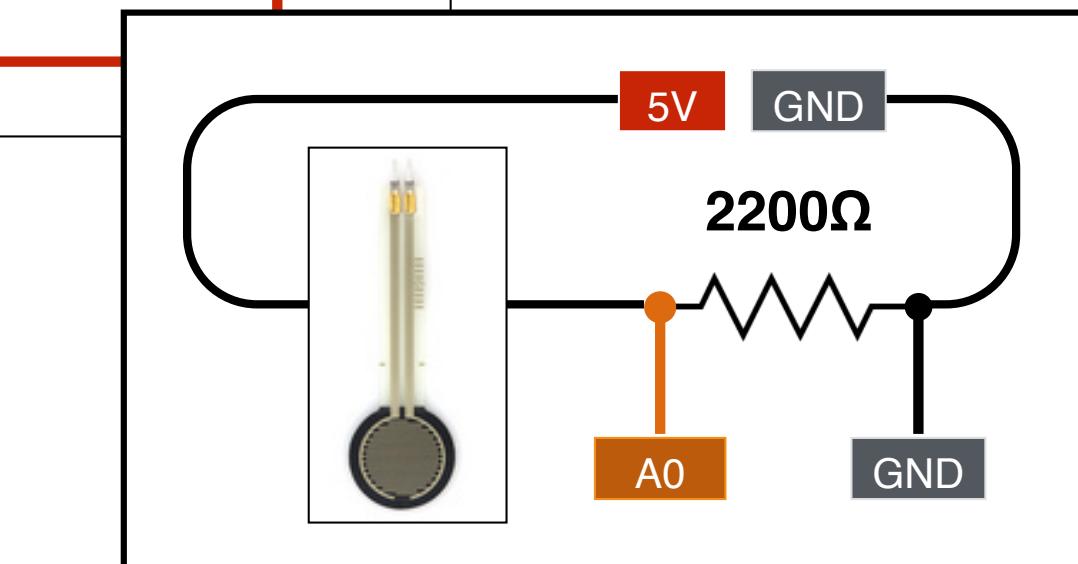
psketch\_Touch.pde

```
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 }
```

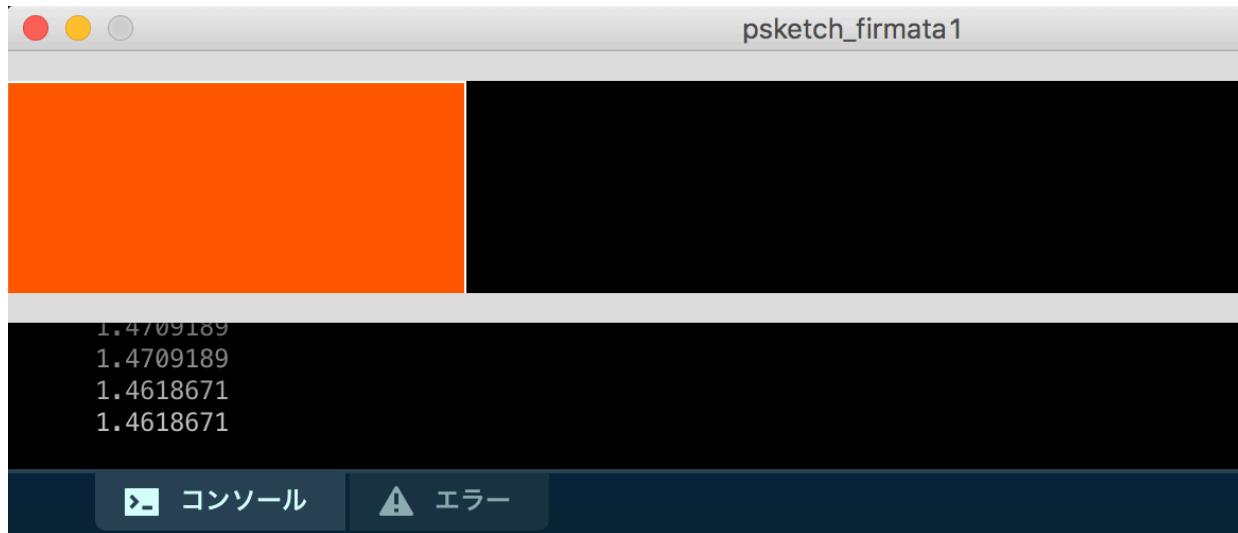
ArduinoのANピンは、Processingでは整数型のNに対応します。

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。

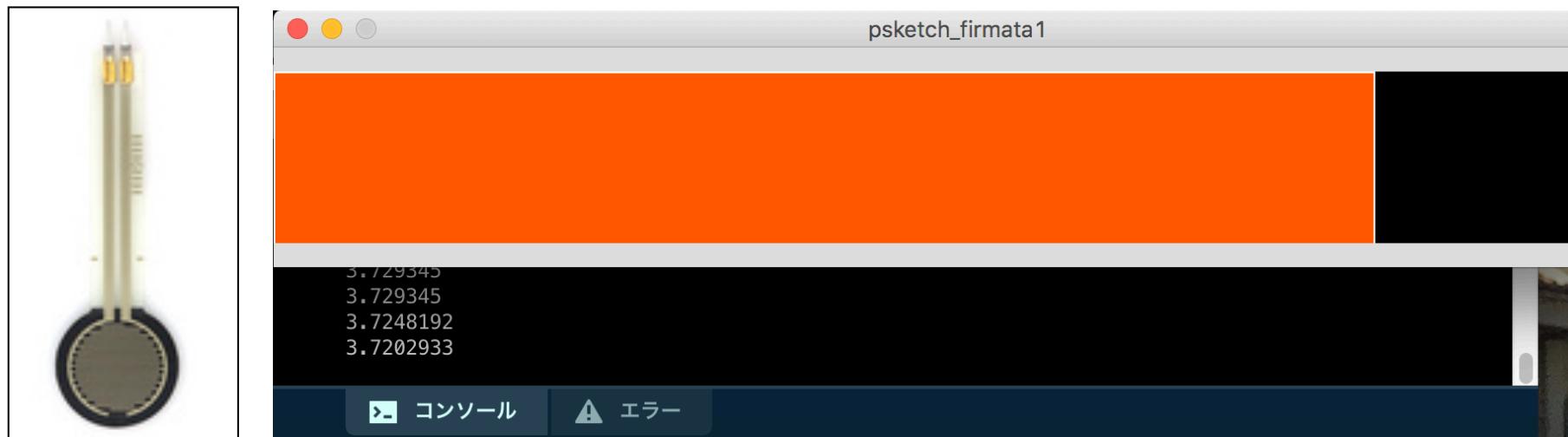


# 圧力センサの値をビジュアライズする



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

```
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 }
```

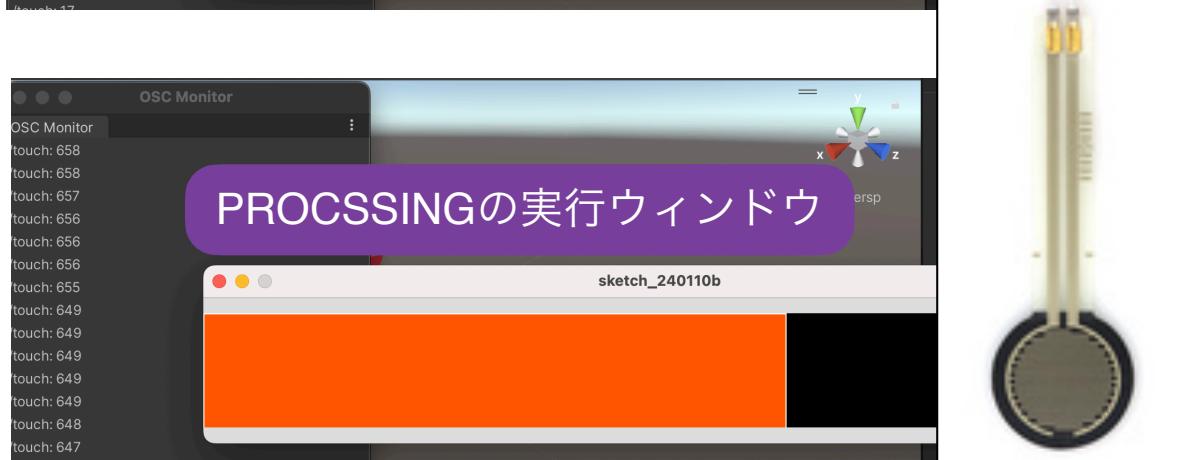
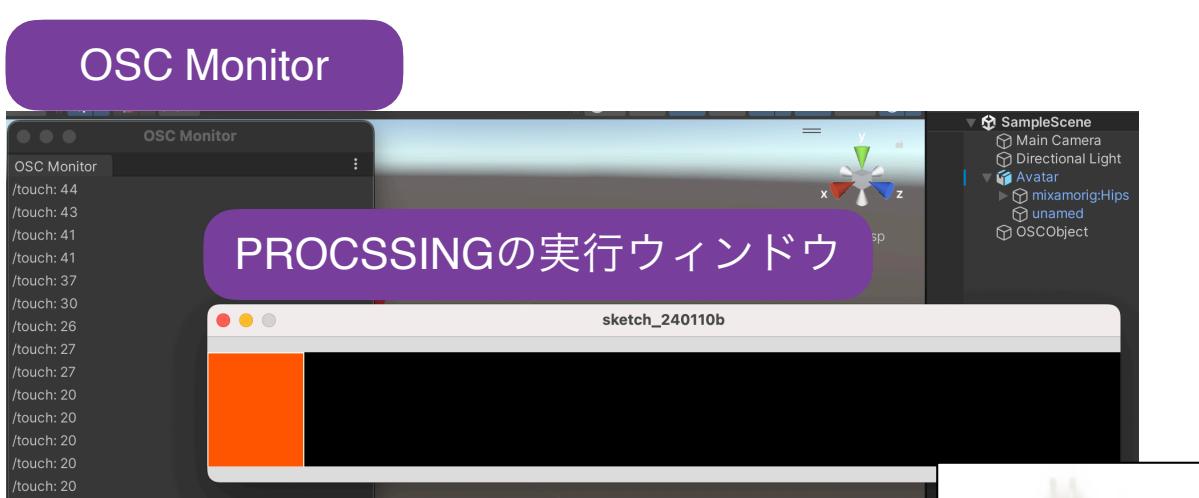
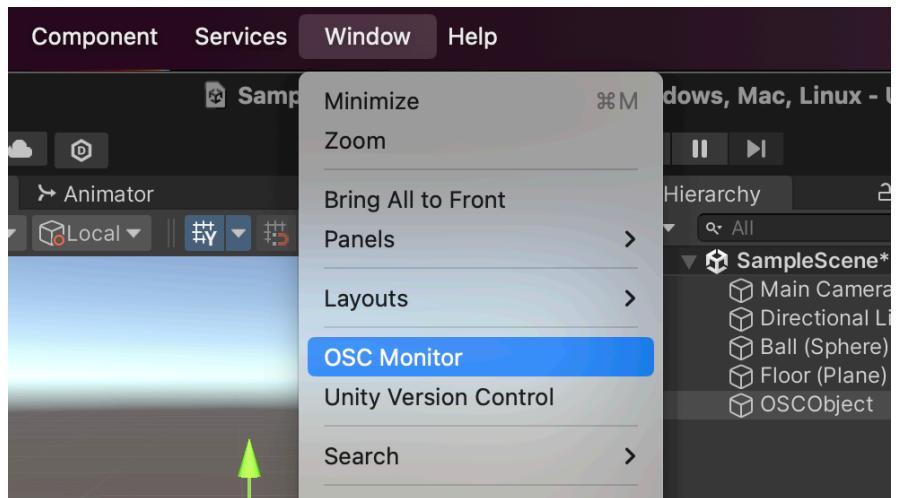
psketch\_Touch\_osc.pde

psketch\_Touch.pde  
にOSC通信のコード  
を追加します。

追記部分

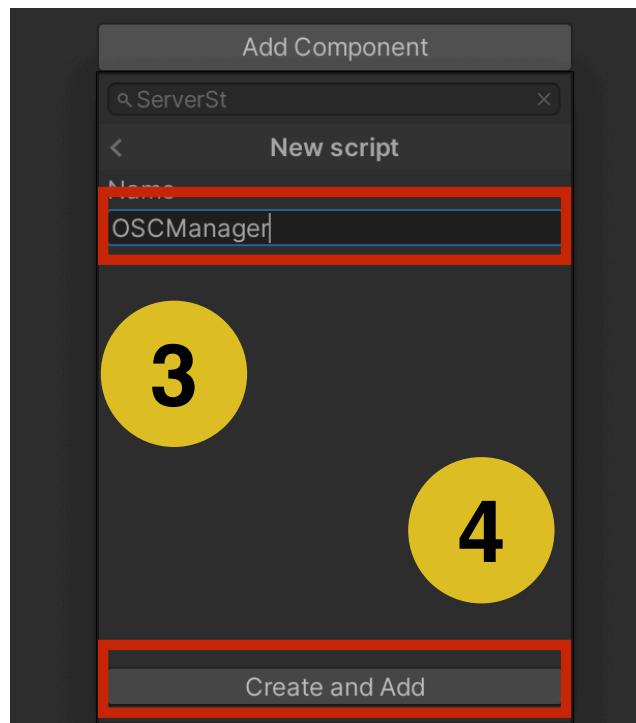
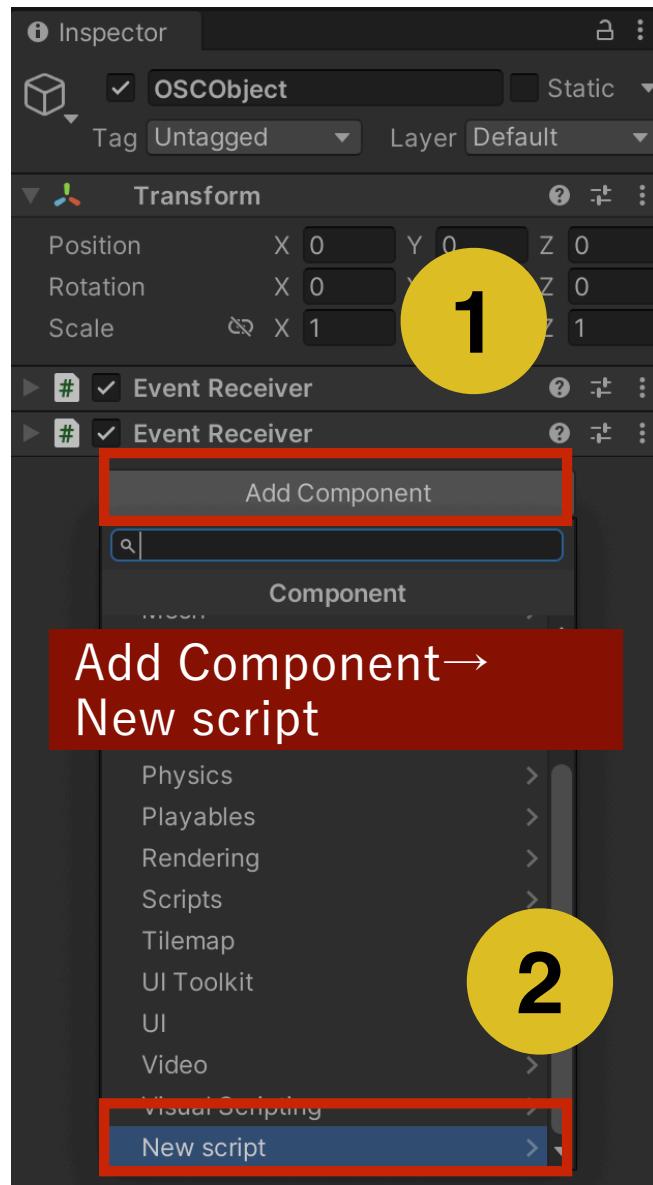
## OSC Monitorによる受信状況の確認

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。

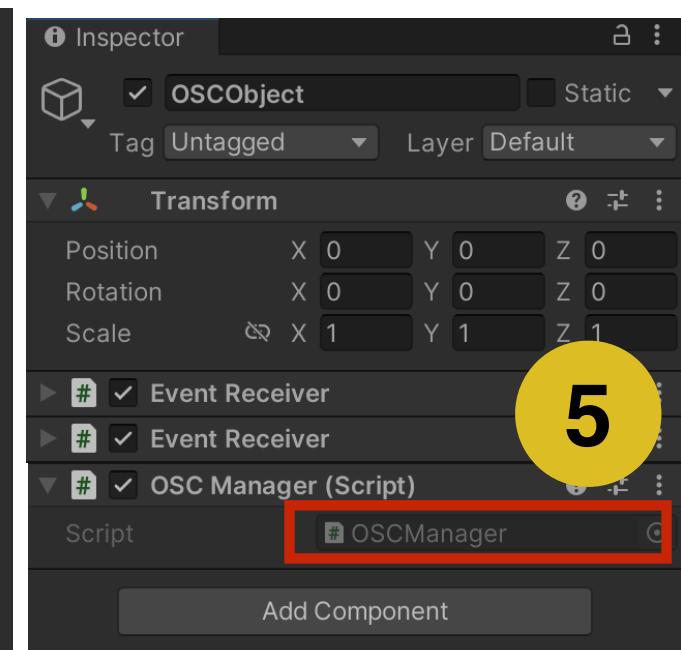


## コールバック関数の作成と関連付け (1/3)

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。

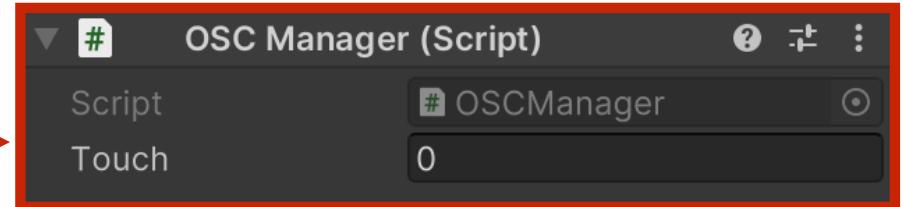
## コールバック関数の作成と関連付け (2/3)

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  
なるとともに  
タイムに値を

/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）

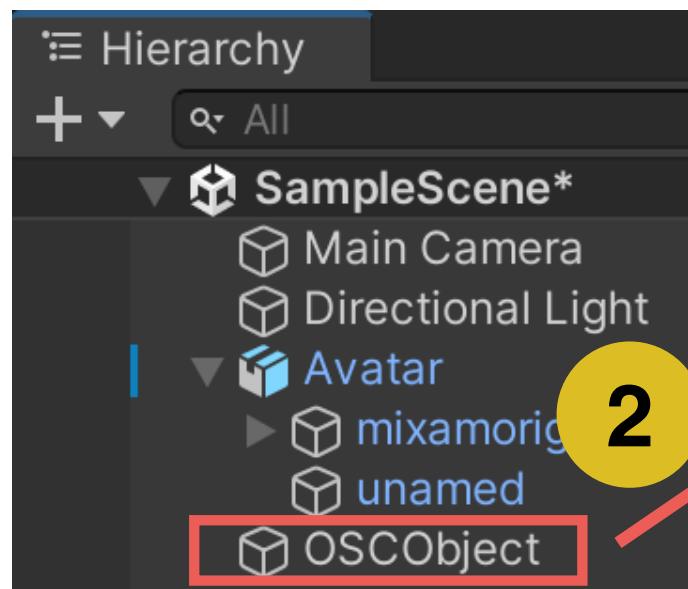


`public`変数は、他のスクリプトファイルから参照可能になるとともに、インスペクタビューに表示され、リアルタイムに値を参照することができるようになります。



## 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。

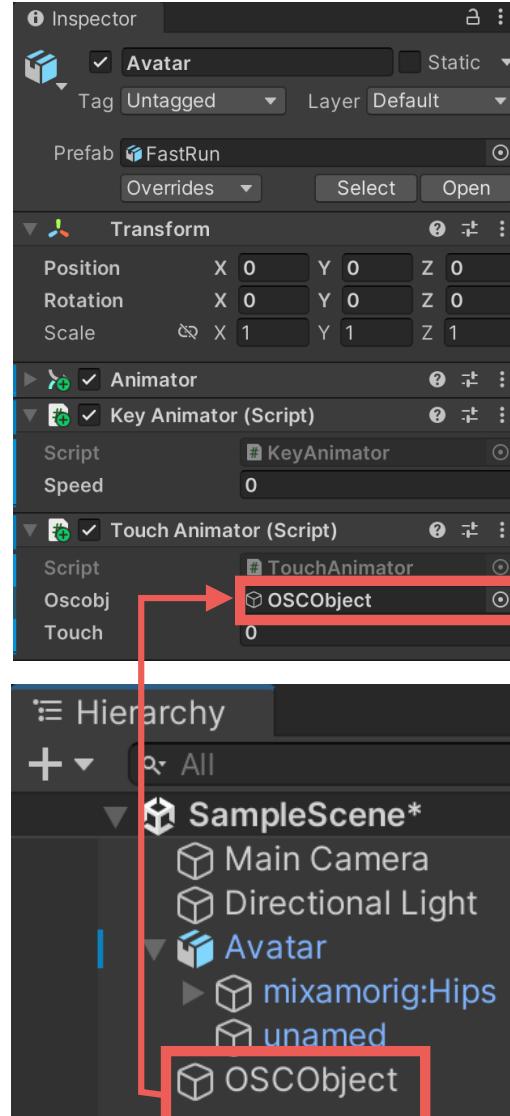


先ほど作成したコールバック関数の入っているゲームオブジェクトであるOSCOBJECTをアタッチします



# 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;           Animationの状態が「Flair」のとき

    if (animator.GetCurrentAnimatorStateInfo(0).IsName("Flair")){
        if(touch>600){
            animator.SetInteger("aflag",1);
        }                           touch>600で、aflagを1にセット
    }
}

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

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