

# 演習：フィジカル・コンピューティング

12/07

3限

演習 1 | ブレッドボードに親しむ

12/07

4限

演習 2 | センサーの状態を「電圧計」から読み取る

12/07

5限

演習 3 | Arduino (センサー)

12/21

4限・5限

演習 4 A | Arduino - Processing

演習 4 B | Unity - Processing

(アプリケーション間通信)

# 演習：フィジカル・コンピューティング

12/07

3限

演習 1 | ブレッドボードに親しむ

12/07

4限

演習 2 | センサーの状態を「電圧計」から読み取る

12/07

5限

演習 3 | Arduino (センサー)

12/21

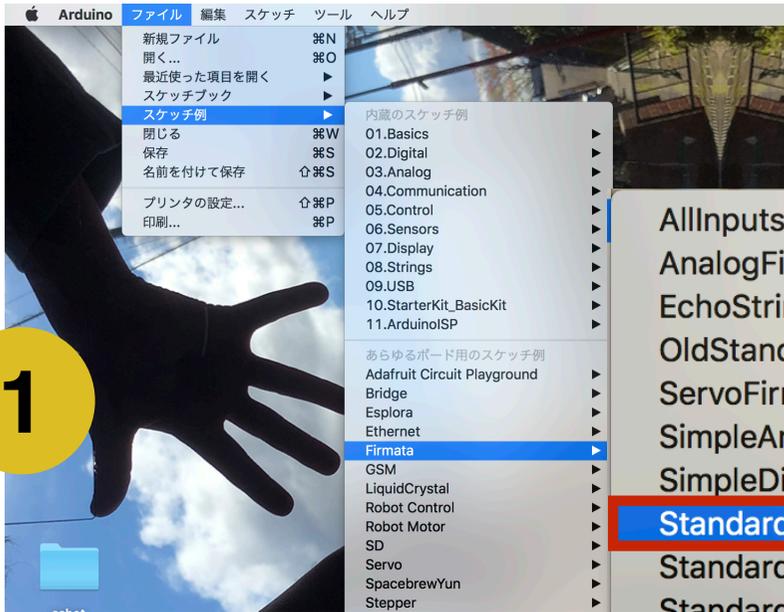
4限・5限

演習 4 A | Arduino - Processing

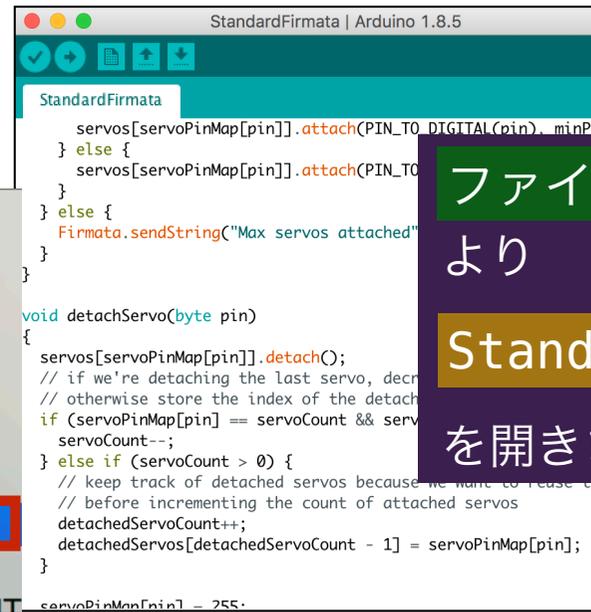
演習 4 B | Unity - Processing

(アプリケーション間通信)

# Firmataの導入 (Arduino側)

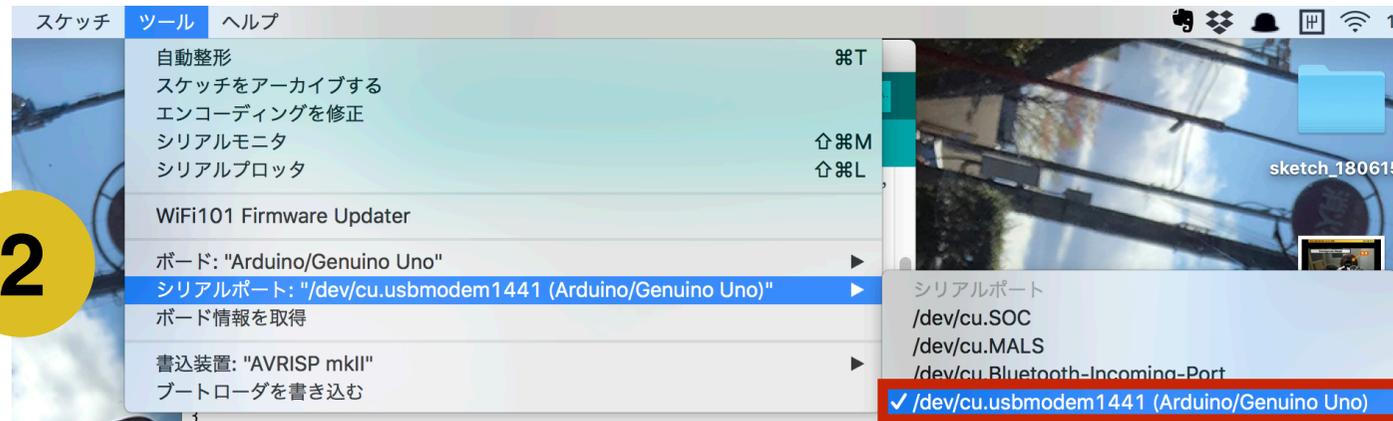


AllInputsFirmata  
AnalogFirmata  
EchoString  
OldStandardFirmata  
ServoFirmata  
SimpleAnalogFirmata  
SimpleDigitalFirmata  
**StandardFirmata**  
StandardFirmataBLE  
StandardFirmataChipKIT



ファイル/スケッチ例  
より

StandardFirmata  
を開きます。



適切なシリアルポートを選  
択するとともに、ポート名  
を覚えておいてください。

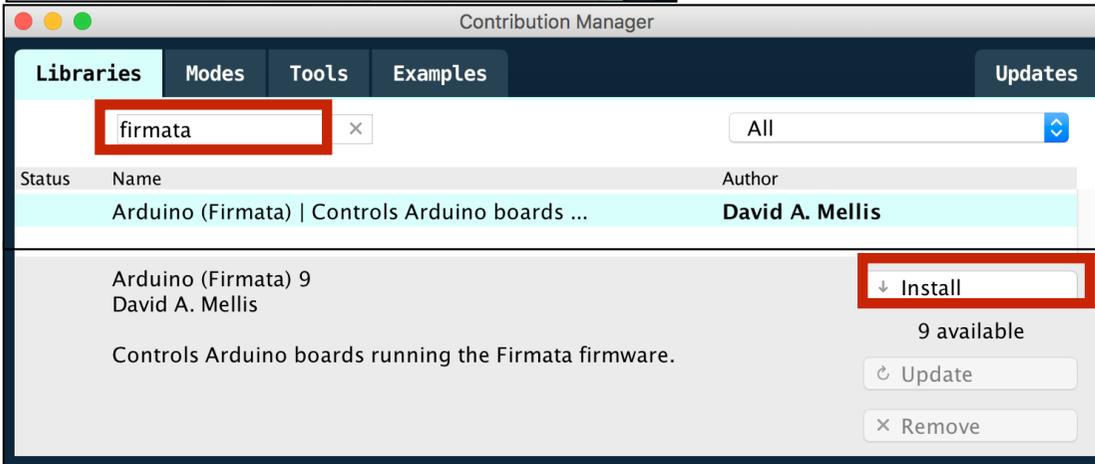


USBを介してArduinoを接続し、  
(ファイルを一切編集することなく) 書き込みを実行してください。

# Firmataの導入 (Processing側)



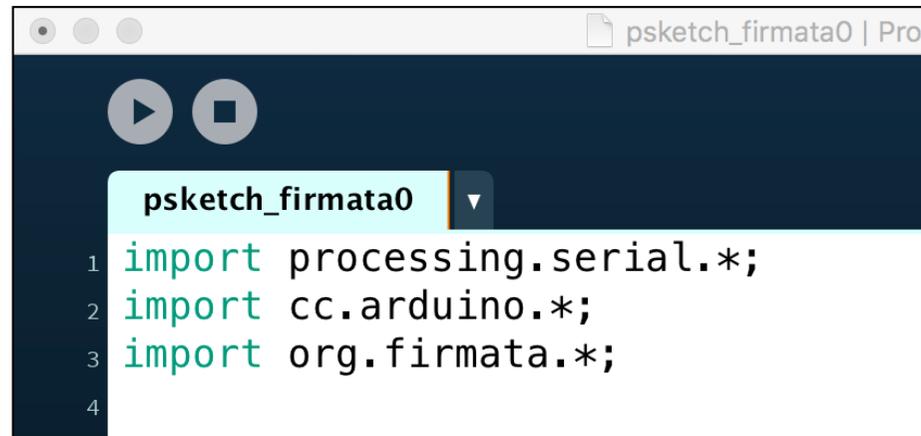
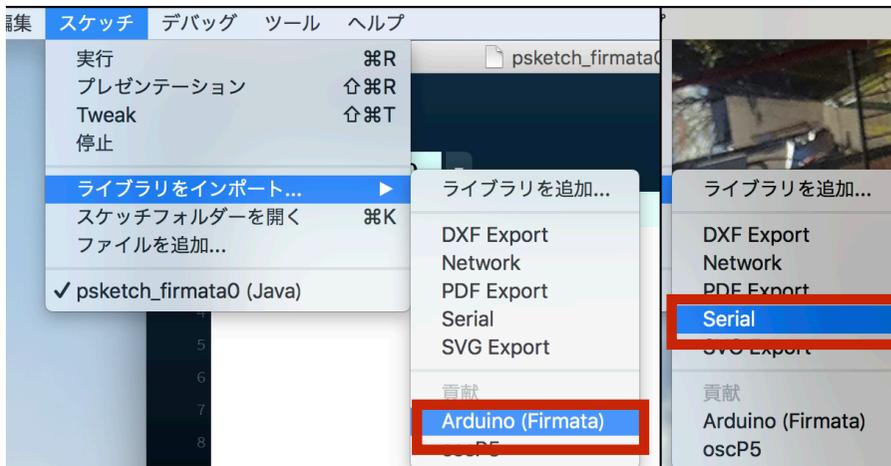
スケッチ/ライブラリをインポート/ライブラリを追加



1

Librariesタブで, firmataとタイプして, 対応するライブラリをインストールしてください。

スケッチ/ライブラリをインポート/Arduino (Firmata)



2

Arduino (Firmata) とSerialの二つのライブラリをインポートしてください。

# Firmataの動作確認 (Processing側)

psketch\_firmata0

```
1 import processing.serial.*;
2 import cc.arduino.*;
3 import org.firmata.*;
4
5 //Arduino arduino;
6 //int ledPin = 13;
7
8 void setup(){
9
10  println(Arduino.list());
11
12  //arduino = new Arduino(this,A
13  //arduino.pinMode(ledPin, Ardu
14
15 }
16
17 void draw(){
18
19 }
```

まず, Arduinoを接続しているシリアルポートと一致する<添字>を探してください。

er 16th, 2016

しました。

メモリのうち、スケッチが12602バイト (39%) を使っている、グローバル変数が1099バイト (53%) を使っていて、ロー

/dev/cu.usbmodem1461 Arduino/Genuino Uno

0

1

2

3

/dev/cu.Bluetooth-Incoming-Port /dev/cu.MALS /dev/cu.SOC /dev/cu.usbmodem1461  
/dev/tty.Bluetooth-Incoming-Port /dev/tty.MALS /dev/tty.SOC /dev/tty.usbmodem1461

4

5

6

7

このケースでは、4番目 (添字としては3) のシリアルポート名が一致しています。この3という数字を覚えておきます。

# Firmataの動作確認 (Processing側)

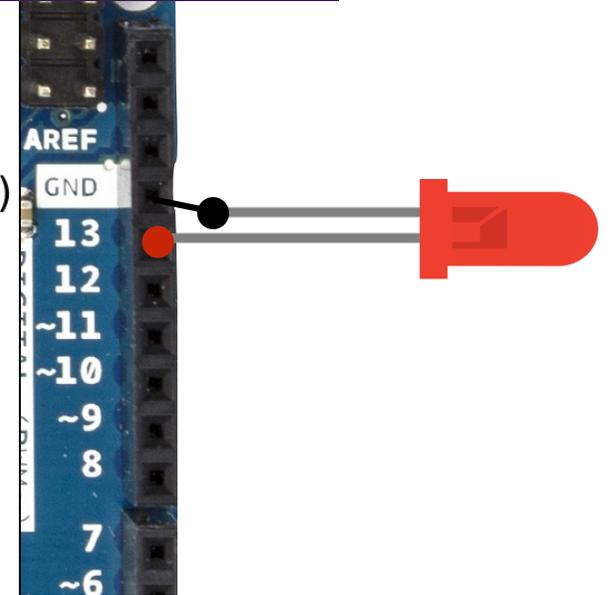
マウスを押す／離すによって、LEDのON・OFFが切り替わり、同時に画面の背景も黒から赤に変わります。なお、実行の際には、Arduinoは終了しておきます (USBをProcessingのために解放するためです)。

psketch\_LED.pde

```
1 import processing.serial.*;
2 import cc.arduino.*;
3 import org.firmata.*;
4
5 Arduino arduino;
6 int ledPin = 13;
7
8 void setup(){
9   //println(arduino.list());
10  arduino = new Arduino(this, Arduino.list()[3], 57600);
11  arduino.pinMode(ledPin, Arduino.OUTPUT);
12 }
13
14 void draw(){
15 }
16
17 void mousePressed(){
18  arduino.digitalWrite(ledPin, Arduino.HIGH);
19  background(color(255,0,0));
20 }
21
22 void mouseReleased(){
23  arduino.digitalWrite(ledPin, Arduino.LOW);
24  background(color(0));
25 }
```

Arduino型のobjectの宣言。以下で、Arduinoのソフトウェアで使用していたメソッドは、このobjectのインスタンスメソッドとして使用することができます。

先に覚えておいた添字に合わせます。



Arduinoのソフトウェアで使用していたシステム変数 (OUTPUT, HIGH, LOW, ...) は、Arduinoクラスのクラス変数として使用できます。

# 圧力センサの値をビジュアライズする

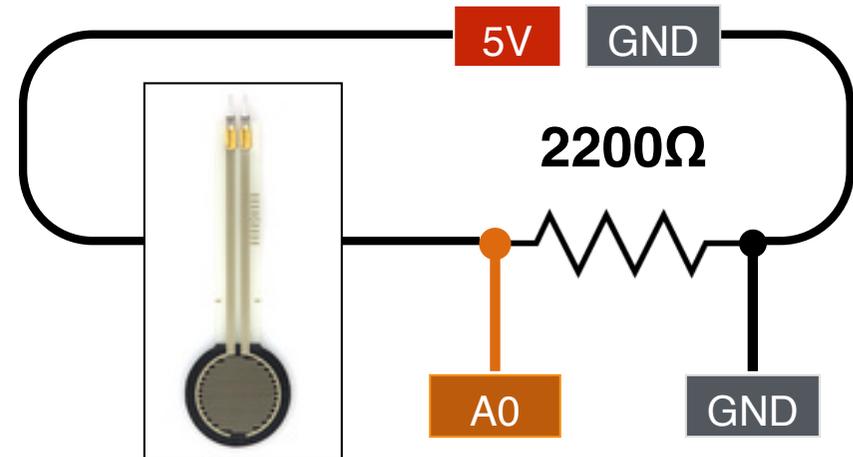
psketch\_Touch.pde

```
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 }
```

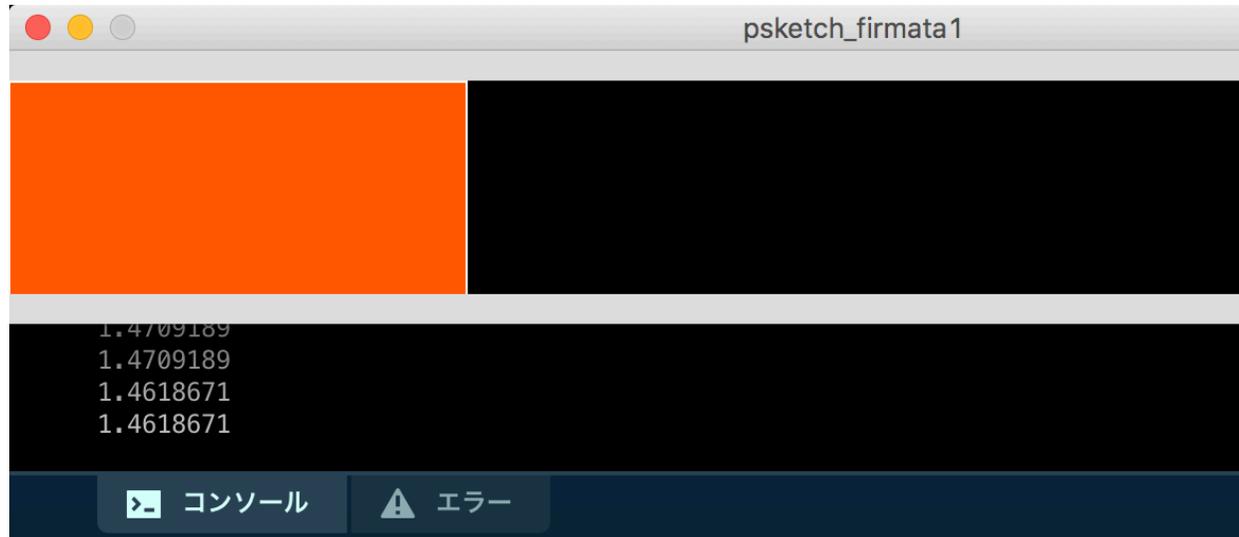
ArduinoのANピンは、Processingでは整数型のNに対応します。

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。

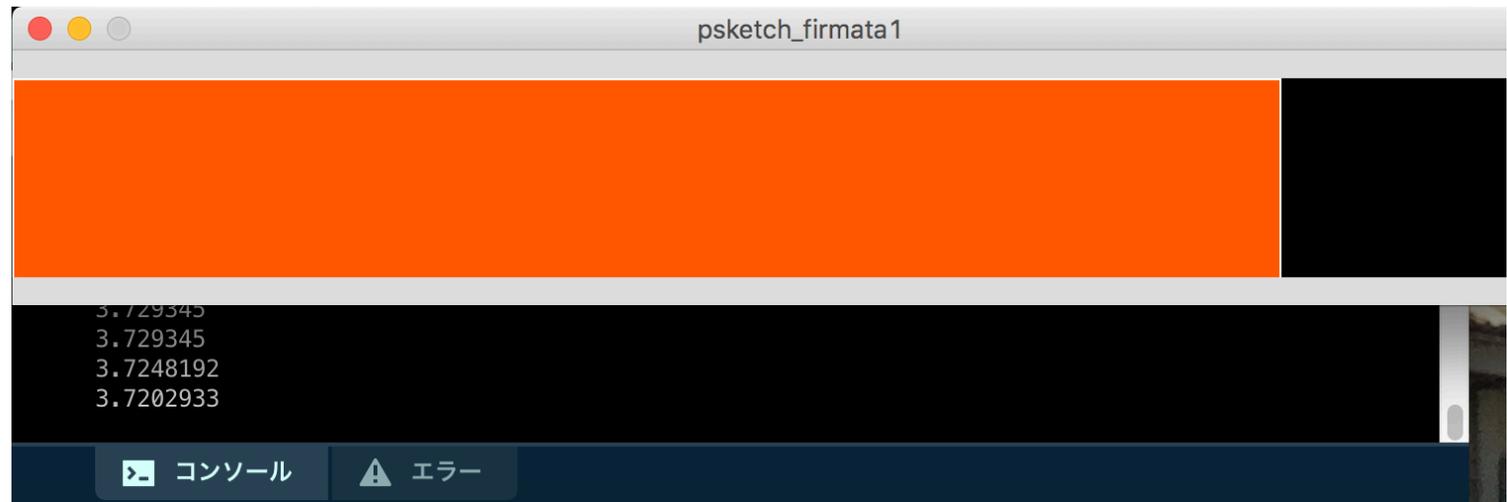


# 圧力センサの値をビジュアライズする



実行結果

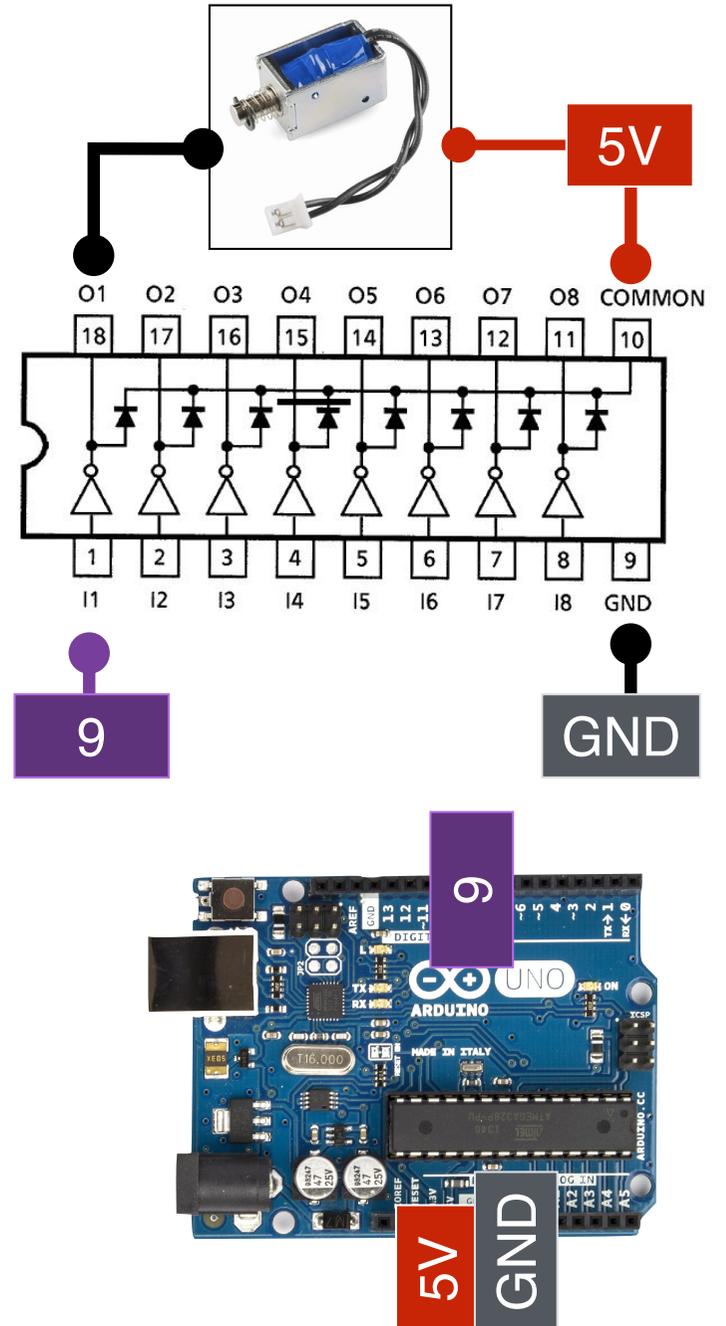
圧力センサを押すと、バーの長さが伸び縮みします。



# ソレノイドをGUIで動かす

psketch\_sorenoid.pde (1/2)

```
psketch_sorenoid
1 import processing.serial.*;
2 import cc.arduino.*;
3 import org.firmata.*;
4
5 Arduino arduino;
6 int sorPin = 9;
7 int val = 255; //ソレノイドの出力値
8 float cx, cy; //中点座標
9
10 float t_last = 0; //ソレノイドの最近の出力時刻 (ms)
11 float space = 200; //ソレノイドの休符時間長 (ms)
12
13 void setup(){
14   arduino = new Arduino(this, Arduino.list()[3], 57600);
15   arduino.pinMode(sorPin, Arduino.OUTPUT);
16
17   size(600,600); background(0);
18   cx = 0.5 * width; cy = 0.5 * height;
19 }
```



前回ソレノイドの値を変えた時刻からの経過時間 t をモニタします。

```
21 void draw(){
```

```
22     float t = millis() - t_last; //経過時間
```

```
23     //経過時間が休符時間長を超えた場合
```

```
24     if(t>space){
```

```
25         if(val==255){
```

```
26             val = 0;
```

```
27         }else{
```

```
28             val = 255;
```

```
29         }
```

```
30         arduino.analogWrite(sorPin, val);
```

```
31         t_last = millis(); //出力時刻の更新
```

```
32     }
```

```
33     if(mousePressed){
```

```
34         float rad = dist(mouseX, mouseY, cx, cy);
```

```
35         background(0); fill(255,100,0); noStroke();
```

```
36         ellipse(cx,cy,2*rad,2*rad);
```

```
37         stroke(255,255,0);
```

```
38         line(cx,0,cx,height); line(0,cy,width,cy);
```

```
39         space = rad;
```

```
40     }
```

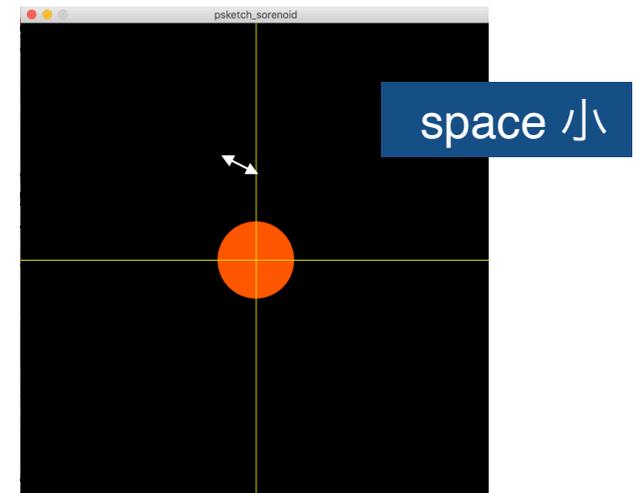
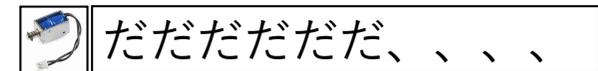
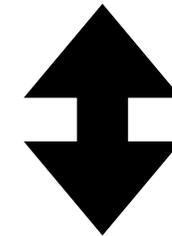
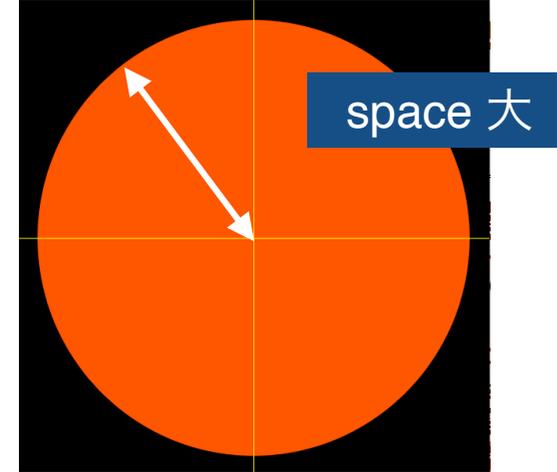
```
41 }
```

draw()

経過時間が space を超えた場合、ソレノイドの値を反転し、t\_lastを現在の時間(ms)に更新します。

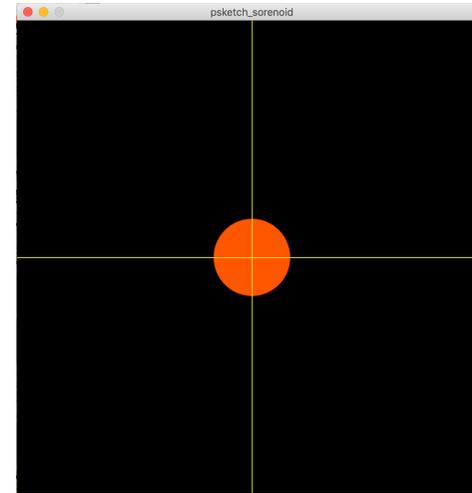
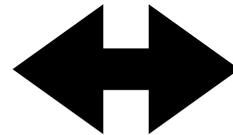
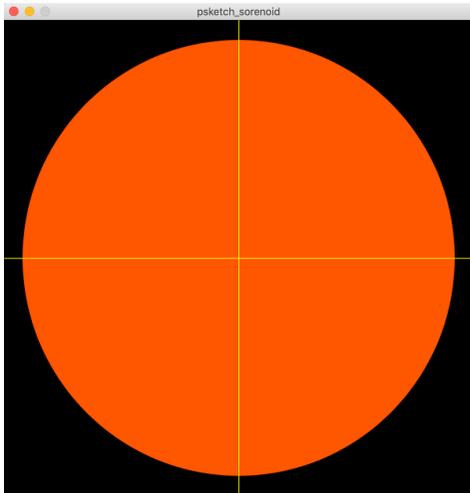
中心からマウスまでの距離に応じて、円の大きさと space を変えます。

psketch\_sorenoid.pde (2/2)

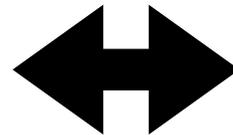


# 練習 1

圧力センサを押すと、GUI上の円が収縮し、それに伴い、ソレノイドの打撃のスピードが高まるようなプログラムを作成してください。



押さない



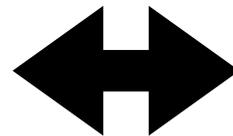
強く押す

圧力センサ

ソレノイド



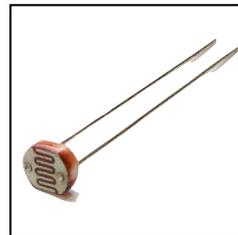
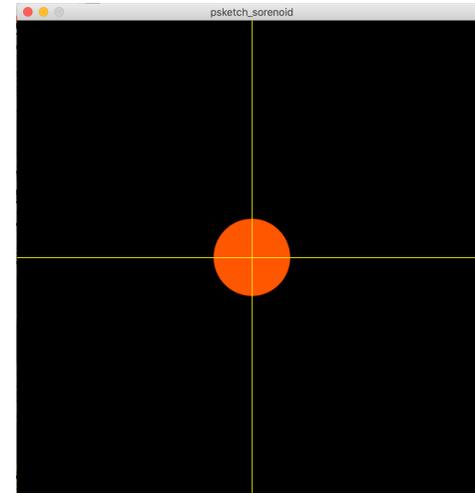
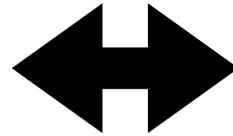
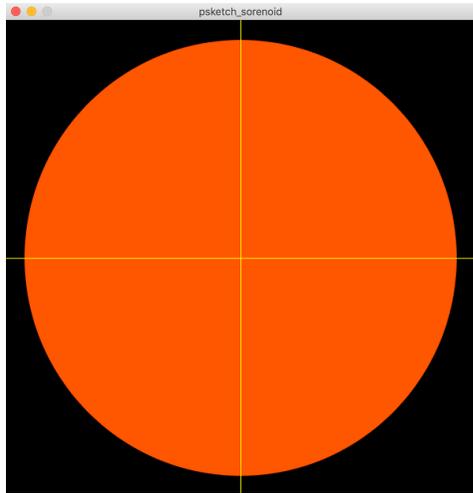
だっ、だっ、だっ、、



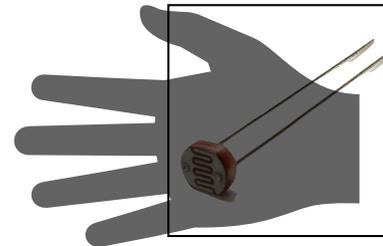
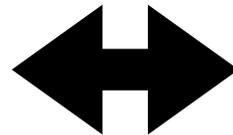
だだだだだだだっ、、

# 練習 2

照度センサを手で隠すと、GUI上の円が収縮し、それに伴い、ソレノイドの打撃のスピードが高まるようなプログラムを作成してください。



隠さない



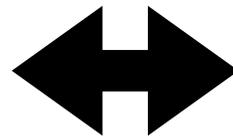
手で隠す

照度センサ

ソレノイド



だっ、だっ、だっ、、



だだだだだだだっ、、

# 演習：フィジカル・コンピューティング

12/07

3限

演習 1 | ブレッドボードに親しむ

12/07

4限

演習 2 | センサーの状態を「電圧計」から読み取る

12/07

5限

演習 3 | Arduino (センサー)

12/21

4限・5限

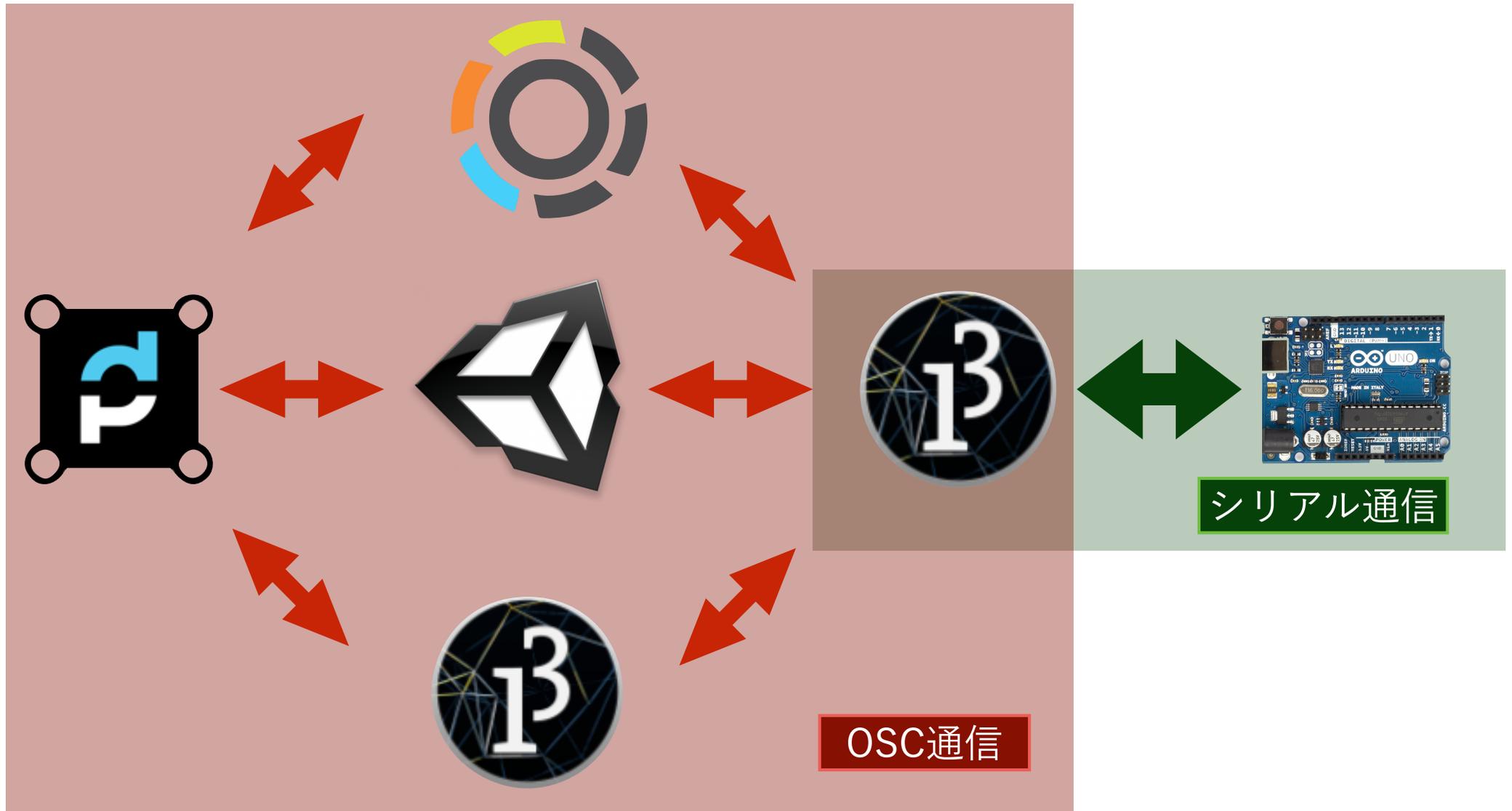
演習 4 A | Arduino - Processing

演習 4 B | Unity - Processing

(アプリケーション間通信)

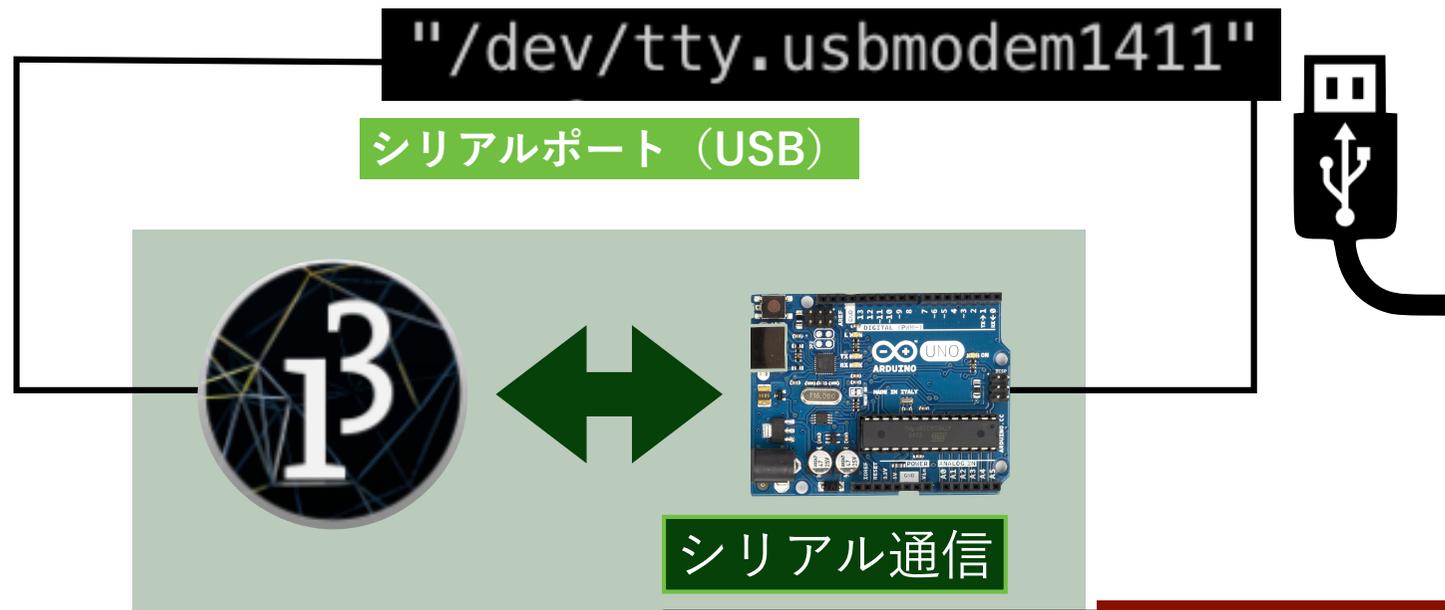
# アプリケーション間通信の全体像

ArduinoとProcessingはシリアル通信ベースで、  
ProcessingとPC内のアプリケーションはOSC通信で、  
相互に情報をやりとりすることができます。



# シリアル通信

シリアル通信は、同一のシリアルポートを介して通信を行います。Processing - Arduino間の通信の場合は、USBがシリアルポートの役割を果たします。一度に送ることのできるデータ量は、1バイト（0-255）と制限されることに注意が必要です。



```
import processing.serial.*;

Serial myport;

void setup(){
  myport = new Serial(this, Serial.list()[7], 115200);
  size(255, 255);
}
```

print(Serial.list())で、対応するUSBポートを探ることができます。

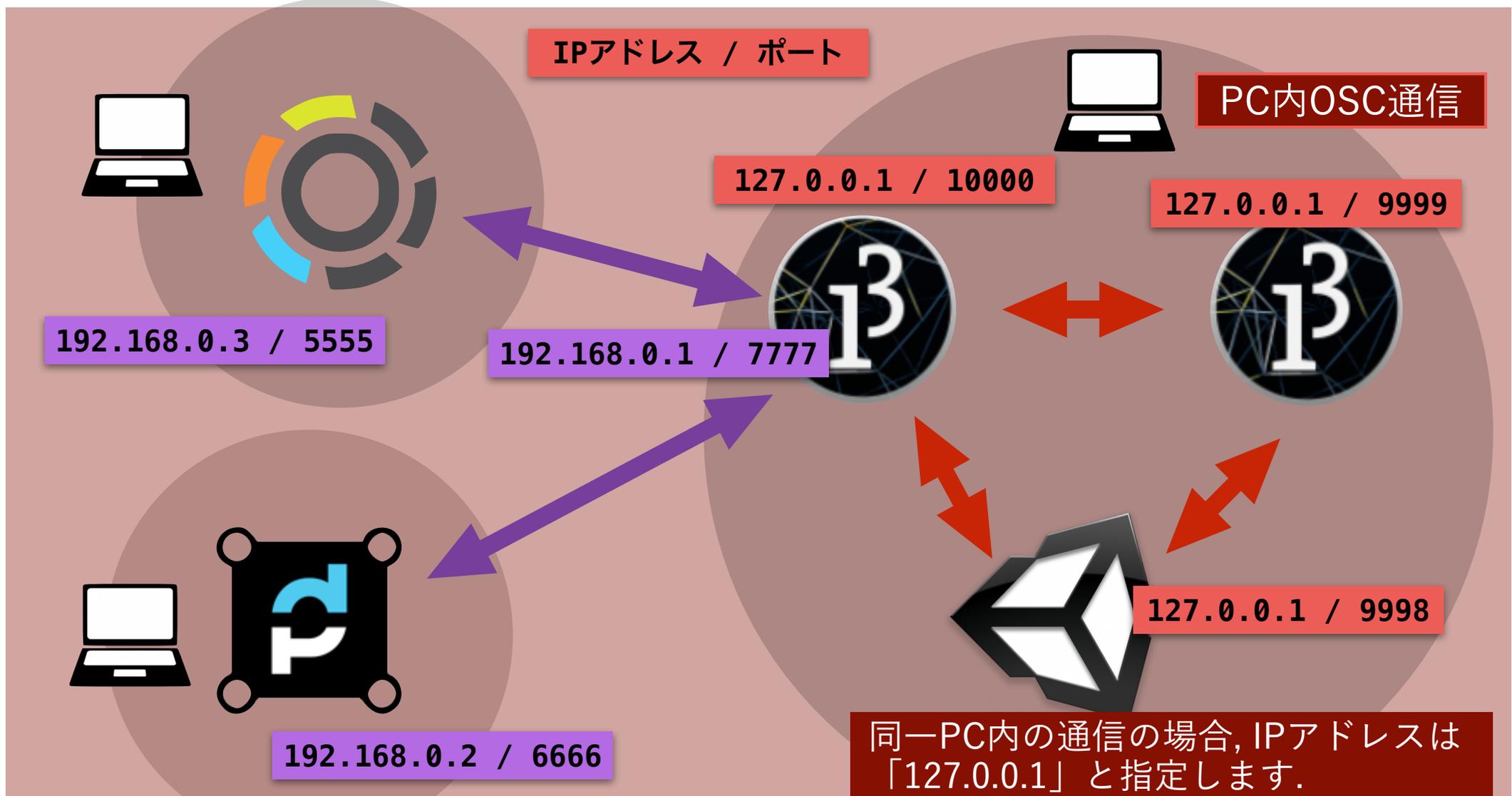
ツール	ヘルプ
自動整形	
スケッチをアーカイブする	
エンコーディングを修正	
シリアルモニタ	
シリアルプロッタ	
WiFi101 Firmware Updater	
ボード: "Arduino/Genuino Uno"	
シリアルポート: "/dev/cu.usbmodem1411"	Arduino/Genuino Uno)
ボード情報を取得	

ArduinoとPCをUSBで接続し、ツールで適当なシリアルポート/  
dev/  
XX.usbmodemXXXXを選択します。

# OSC (Open Sound Control) 通信

## IPアドレスとポート

PC間のOSC通信は、宛先としてIPアドレス（端末の区別）とポート（端末内のアプリケーションの区別）を指定することで、相手を見つけることができます。ポートの数字（通常4桁・5桁）は好きに割り当てます。



# OSC (Open Sound Control) 通信

## OSC Message

OSC通信は、送信するデータをパッケージにして送ります。パッケージの中身として、(1) どのような形式のデータが入っているのか (**OSC Arguments**)、そして (2) パッケージの宛名 (**OSC Address**) を指定する必要があります。

### OSC Message

OSC Address

OSC Arguments (複数可)

/mouse

x(int), y(int)

/mousey

y(int)

/message

info(string)

/volume

amplitude(float)

/mouse 34 25

/mouse 34 28

/mousey 40

/message "hello"

/mouse "good-bye"

/volume 2.453

/volume 1.537

/volume 0.003

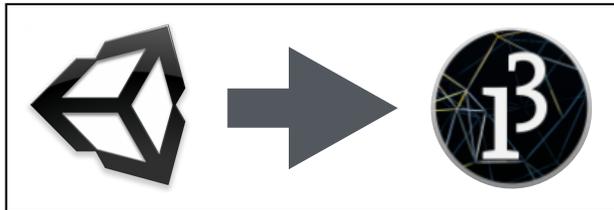
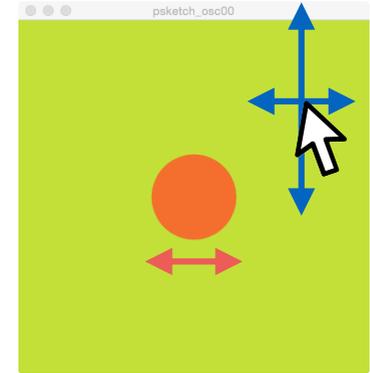
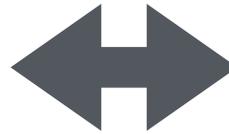
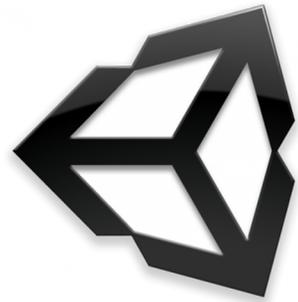
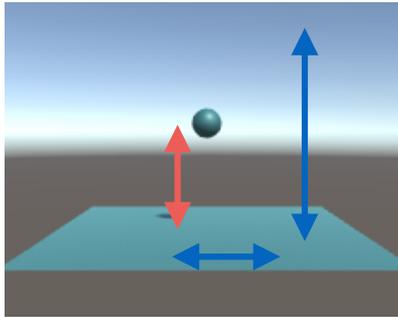
# 実装練習 (UNITY 対 Processing の通信)

127.0.0.1 / 6666

127.0.0.1 / 5555

SERVER

Client



`/pos/floor, /pos/ball`

床とボールのXYZ座標 (float, float, float)

`/hit`

球と床の距離を円の直径と連動させます。

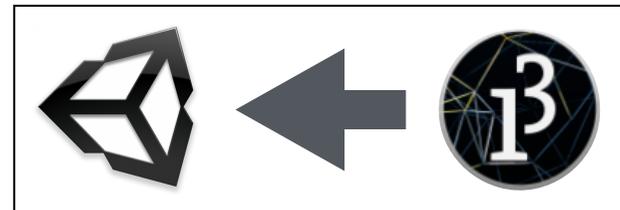
衝突回数 (int)

衝突に合わせて、背景色を変化。

`/mousex, /mousey`

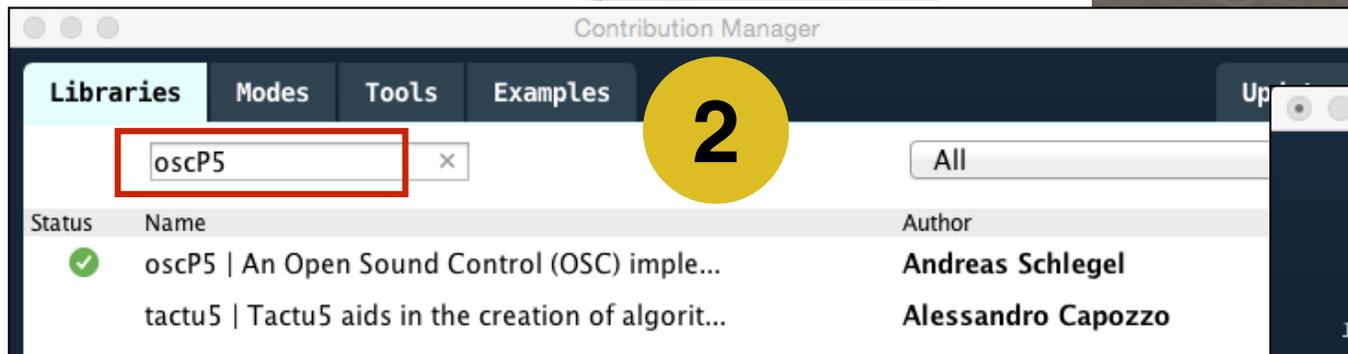
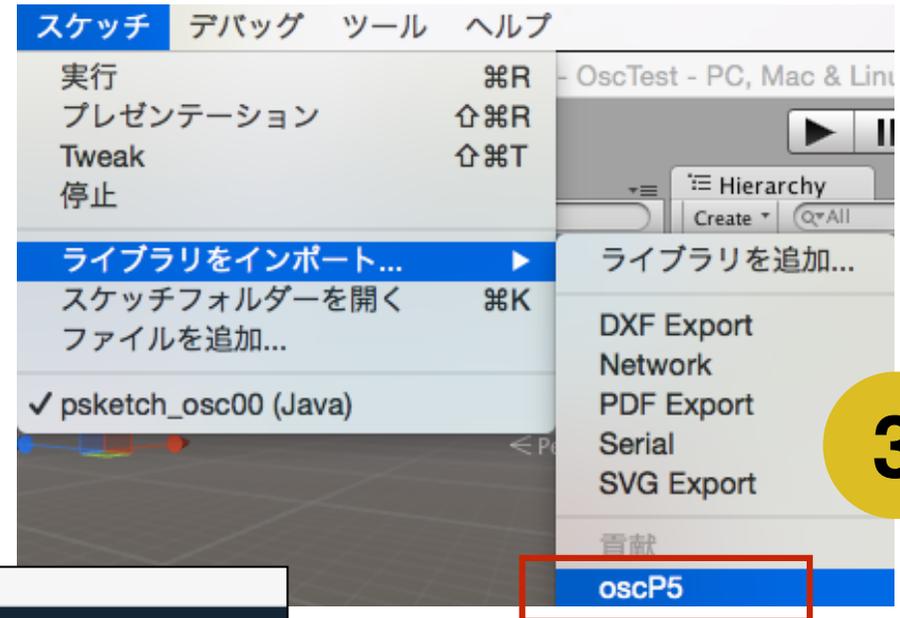
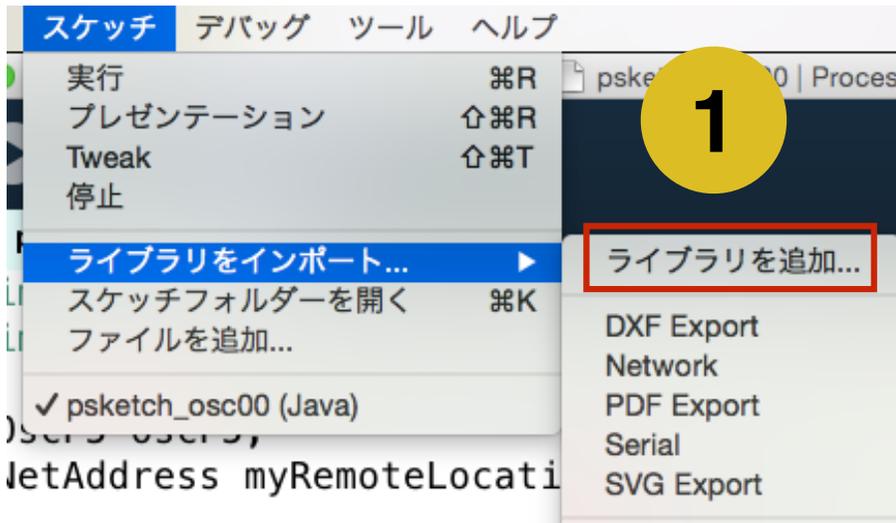
X座標 (int) , Y座標 (int)

マウスのXY座標を、床の高さと連動させます。



# Processing側の準備

## ライブラリ (oscP5) の追加



oscP5と入力すると、候補としてoscP5が表示されるので、選択してInstallしてください。



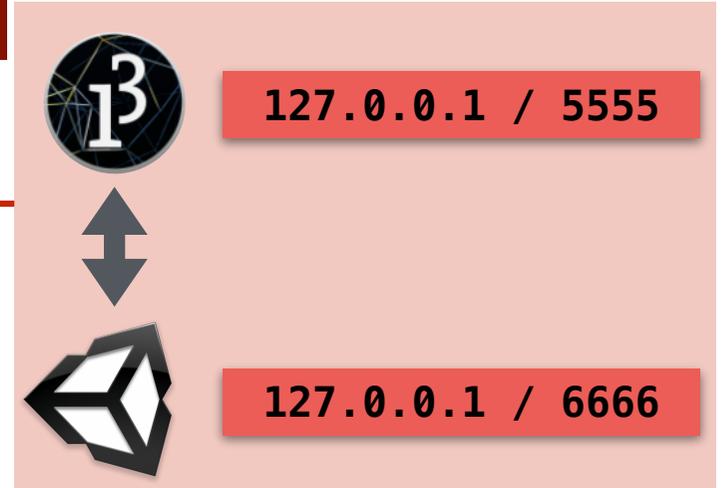
インストール後、Contribution Libraryとして、oscP5が表示されるので選択すると、自動的に関連のimportがimportされます。

# Processing側の準備

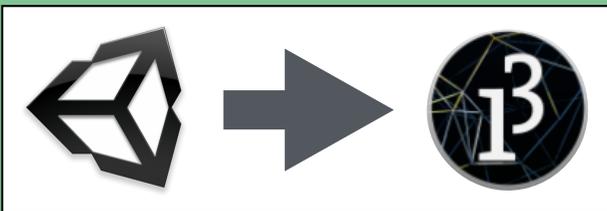
## Processingのコード (1/3)

```
1 import netP5.*;
2 import oscP5.*;
3
4 OscP5 oscP5;
5 NetAddress myRemoteLocation;
6
7 float y_floor = 0; float y_ball = 0; //床とボールのY座標
8 float dif = 0; //ボールと床の距離
9 color col; //背景色 (衝突のたびにランダムに変わる)
10
11 void setup(){
12     size(400,400); frameRate(10); background(0);
13
14     oscP5 = new OscP5(this,5555);
15     myRemoteLocation = new NetAddress("127.0.0.1",6666);
16
17     oscP5.plug(this,"getFloorPos","/pos/floor");
18     oscP5.plug(this,"getBallPos","/pos/ball");
19     oscP5.plug(this,"getHit","/hit");
20 }
```

アドレスとポートを指定します. 自分自身 (Processing) のアドレスは指定する必要ありません.



oscアドレス「/pos/floor」を受け取ると, 自作関数「getFloorPos」をコールアップします. (他の2つも同様)



/pos/floor, /pos/ball

床とボールのXYZ座標 (float, float, float)

/hit

球と床の距離を円の直径と連動させます.

衝突回数 (int)

衝突に合わせて, 背景色を变化.

# Processing側の準備

## Processingのコード (2/3)

```
22 void draw(){
23
24     dif = y_ball - y_floor; //床とボールの距離を計算
25     background(col); noStroke(); fill(255,100,0);
26     ellipse(width/2.,height/2.,50.*dif,50.*dif);
27
28     if(mousePressed){
29         OscMessage myMessage_x = new OscMessage("/mouseX");
30         myMessage_x.add(mouseX);
31         oscP5.send(myMessage_x, myRemoteLocation);
32
33         OscMessage myMessage_y = new OscMessage("/mouseY");
34         myMessage_y.add(mouseY);
35         oscP5.send(myMessage_y, myRemoteLocation);
36     }
37 }
```

マウスが押されている場合、マウスのXY座標を、OSCメッセージとして、UNITYに送ります。

`background(col);`

背景の色 (col) は、getHit関数を受け取るとランダムに変わります。

円の中心を、windowの中心位置に、直径を50\*difで描画します。

`ellipse(width/2.,height/2.,50.*dif,50.*dif);`

`/mouseX`

X座標 (int)

`/mouseY`

Y座標 (int)



# Processing側の準備

## Processingのコード (3/3)

コールアップ関数の処理を実際に定義します。

```
16  
17 oscP5.plug(this,"getFloorPos","/pos/floor");  
18 oscP5.plug(this,"getBallPos","/pos/ball");  
19 oscP5.plug(this,"getHit","/hit");
```

```
39 void getFloorPos(float x, float y, float z){  
40     y_floor = y;  
41 }
```

/pos/floorを受け取ったら、2つ目の引数 (床のY座標) を参照して、y\_floorの値を更新

```
42  
43 void getBallPos(float x, float y, float z){  
44     y_ball = y;  
45 }
```

/pos/ballを受け取ったら、2つ目の引数 (床のY座標) を参照して、y\_ballの値を更新

```
46  
47 void getHit(int count){  
48     println(count);  
49     col = color(random(255),random(255),random(255));  
50 }
```

/hitを受け取ったら、引数 (衝突数) を出力して、背景色をランダムに変更 (実質的にcountは描画に使いません)



/pos/floor, /pos/ball

床とボールのXYZ座標 (float, float, float)

/hit

球と床の距離を円の直径と連動させます。

衝突回数 (int)

衝突に合わせて、背景色を変化。

# Processing側の準備

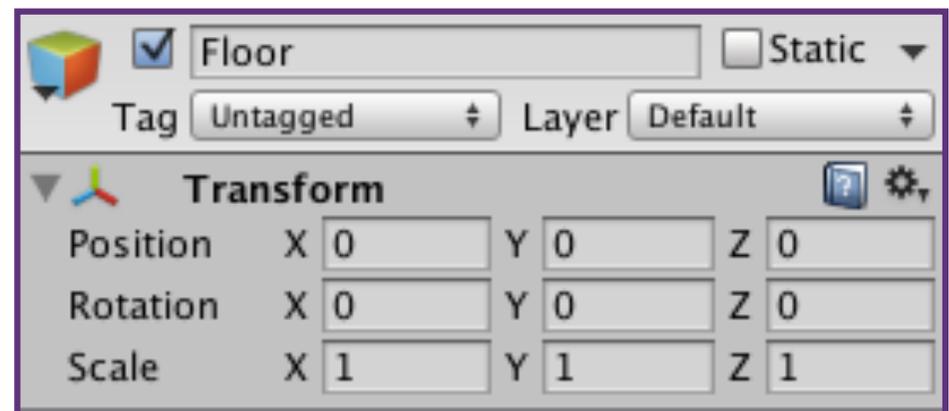
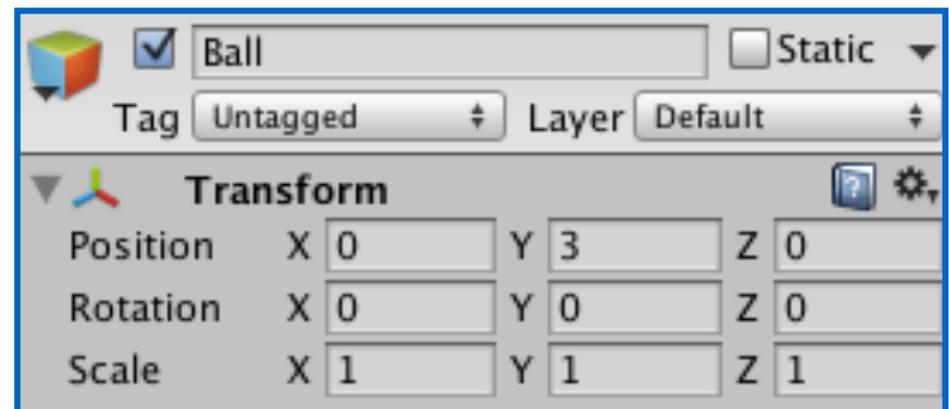
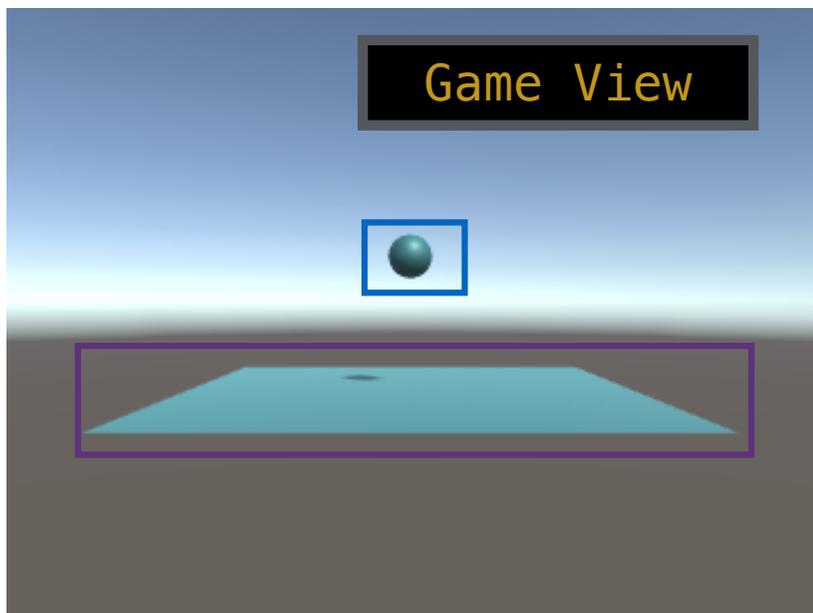
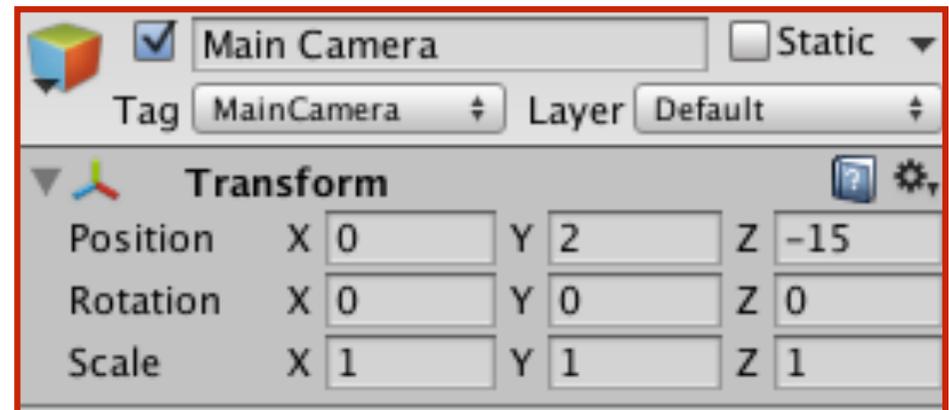
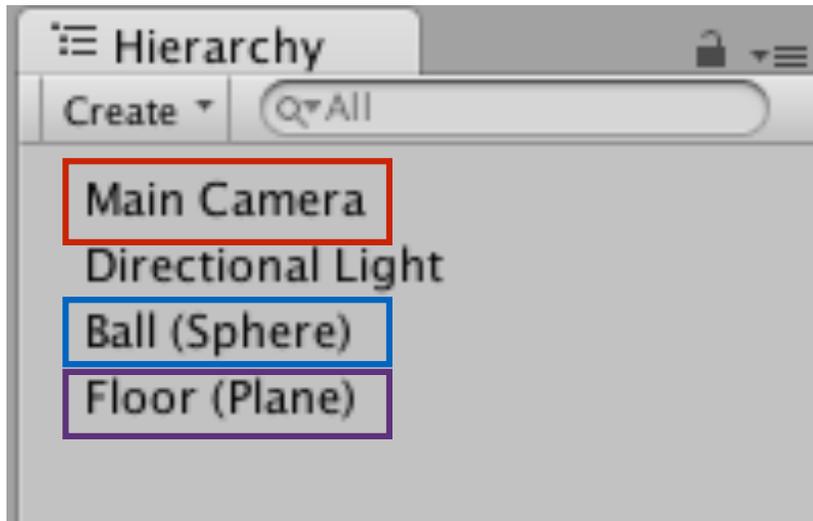
## Processingのコード (全体)

```
1 import netP5.*;
2 import oscP5.*;
3
4 OscP5 oscP5;
5 NetAddress myRemoteLocation;
6
7 float y_floor = 0; float y_ball = 0; //床とボールのY座標
8 float dif = 0; //ボールと床の距離
9 color col; //背景色 (衝突のたびにランダムに変わる)
10
11 void setup(){
12     size(400,400); frameRate(10); background(0);
13
14     oscP5 = new OscP5(this,5555);
15     myRemoteLocation = new NetAddress("127.0.0.1",6666);
16
17     oscP5.plug(this,"getFloorPos","/pos/floor");
18     oscP5.plug(this,"getBallPos","/pos/ball");
19     oscP5.plug(this,"getHit","/hit");
20 }
21
22 void draw(){
23
24     dif = y_ball - y_floor; //床とボールの距離を計算
25     background(col); noStroke(); fill(255,100,0);
26     ellipse(width/2.,height/2.,50.*dif,50.*dif);
```

```
28 if(mousePressed){
29     OscMessage myMessage_x = new OscMessage("/mouseX");
30     myMessage_x.add(mouseX);
31     oscP5.send(myMessage_x, myRemoteLocation);
32
33     OscMessage myMessage_y = new OscMessage("/mouseY");
34     myMessage_y.add(mouseY);
35     oscP5.send(myMessage_y, myRemoteLocation);
36 }
37 }
38
39 void getFloorPos(float x, float y, float z){
40     y_floor = y;
41 }
42
43 void getBallPos(float x, float y, float z){
44     y_ball = y;
45 }
46
47 void getHit(int count){
48     println(count);
49     col = color(random(255),random(255),random(255));
50 }
```

# UNITY側の準備

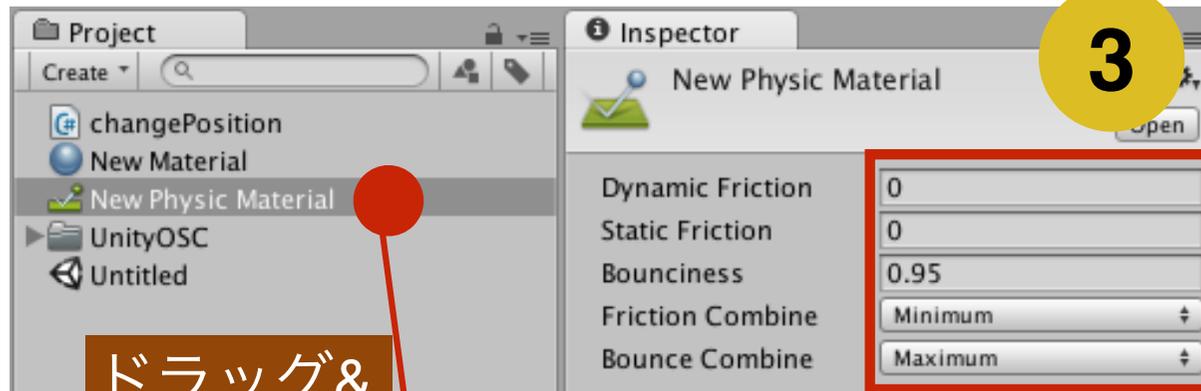
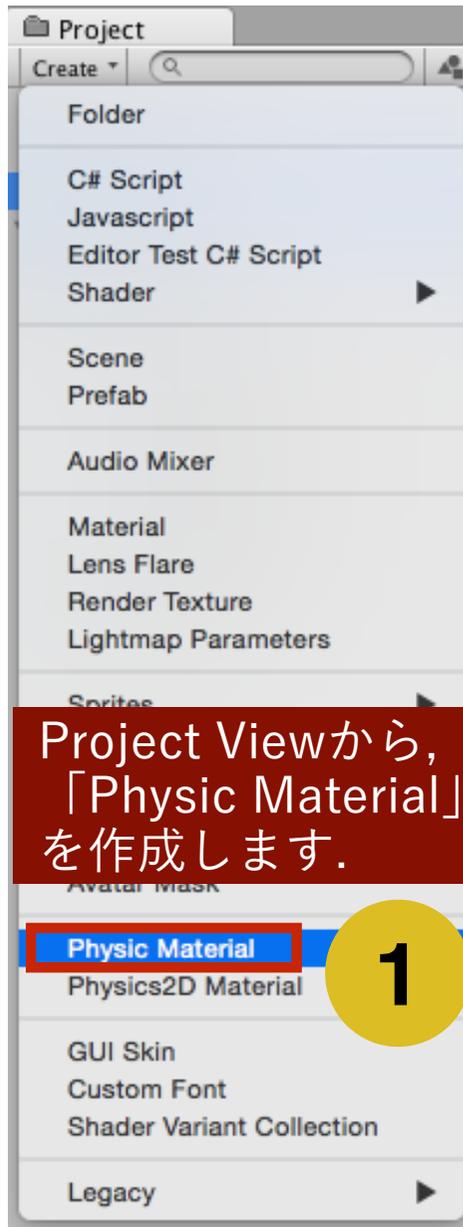
## 基本環境の構築（ゲームオブジェクトの追加）



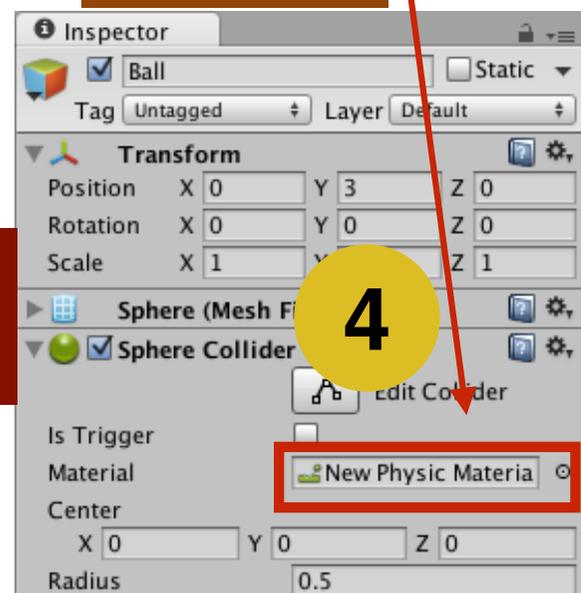
必要に応じて、Materialを追加して色をつけてください。

# UNITY側の準備

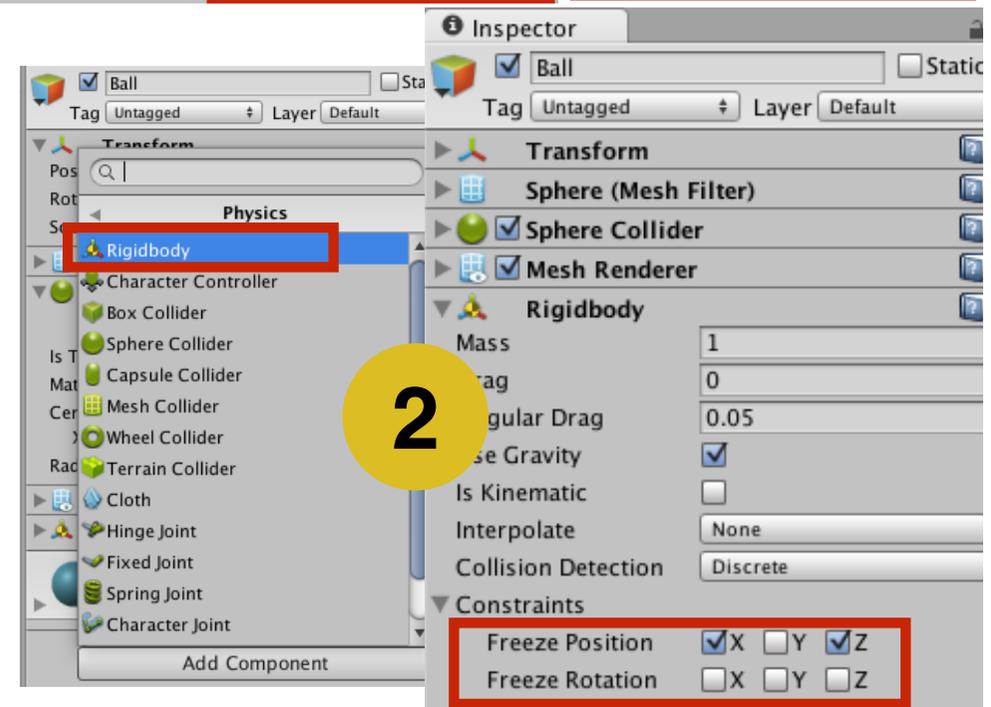
## 基本環境の構築（ボールがバウンドするようにする）



Bouncinessは反発係数ですので、これを増やすと、ボールが床から跳ね返るようになります。



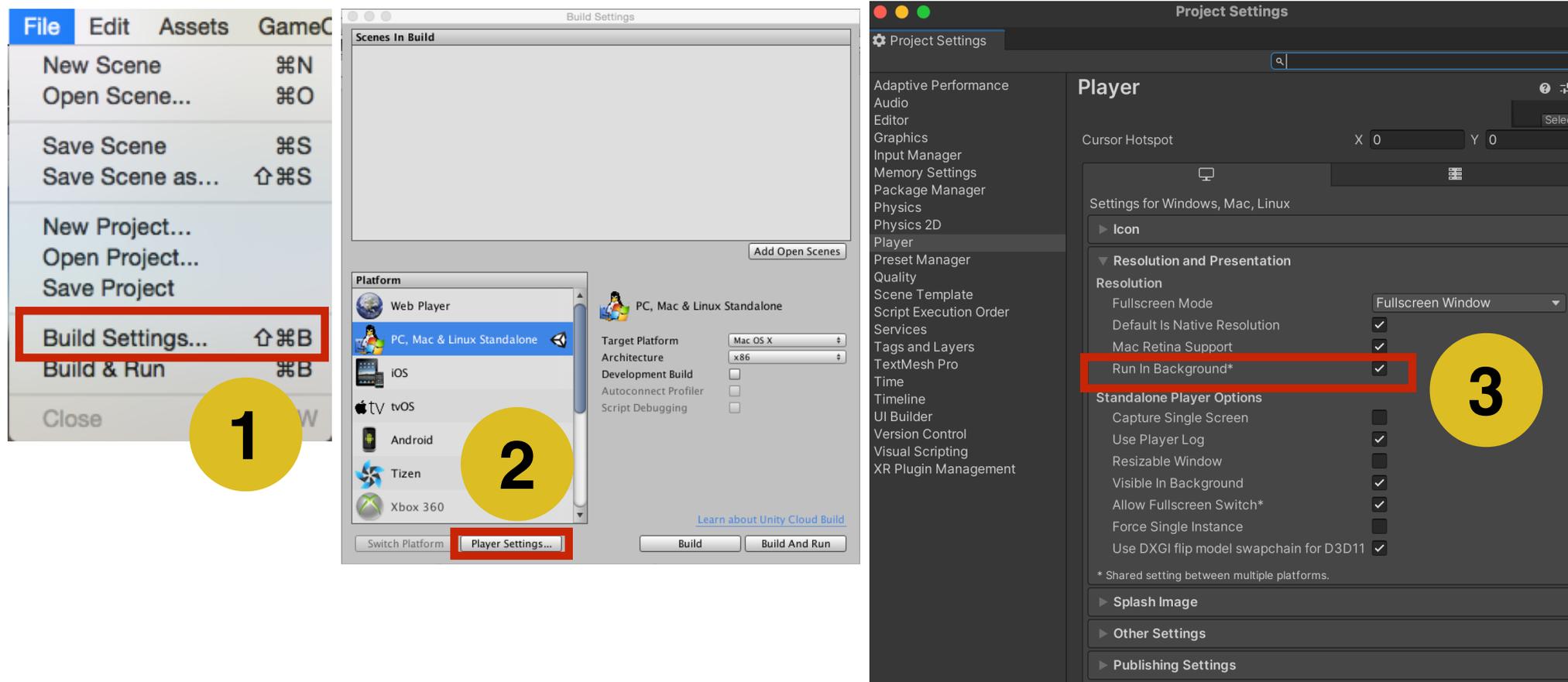
作成した物理特性をBallと関連付けます。



コンポーネントとしてRigidbodyを追加し、Y座標（上下）のみを動くようにします。

# UNITY側の準備

## 基本環境の構築 (バックグラウンドの処理を許可)



Processingの実行アプリケーションを立ち上げている間にも、UNITYの描画処理をキャンセルしないための措置です。

(現バージョンではデフォルトでチェックが入っているはずです。)

# UNITY側の準備

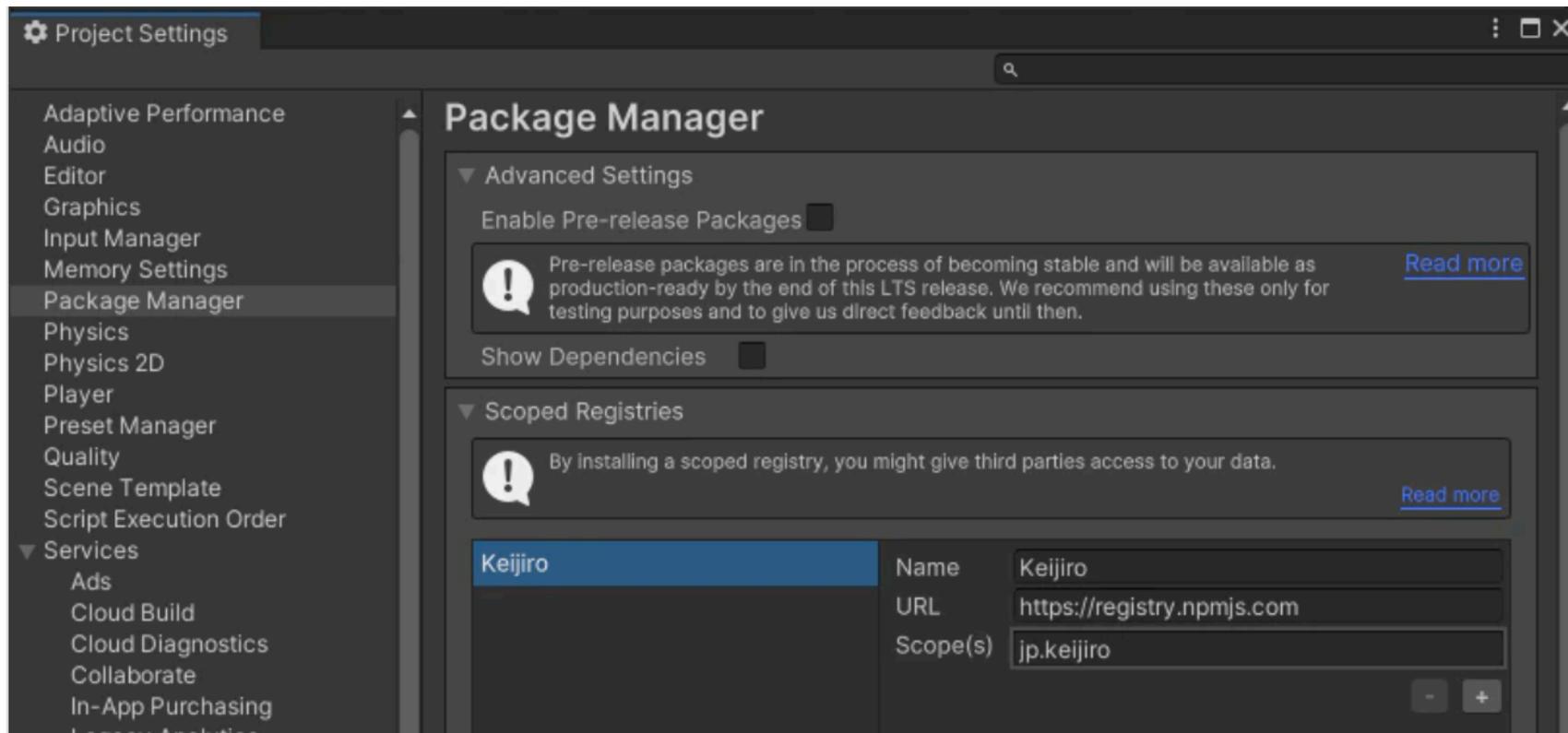
## OscJackのインポート (1/2)

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>

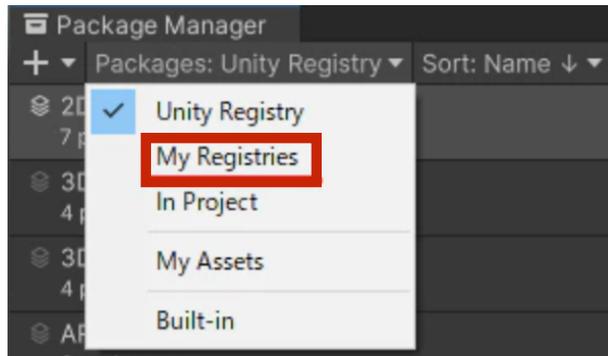
Scope(s) : jp.keijiro



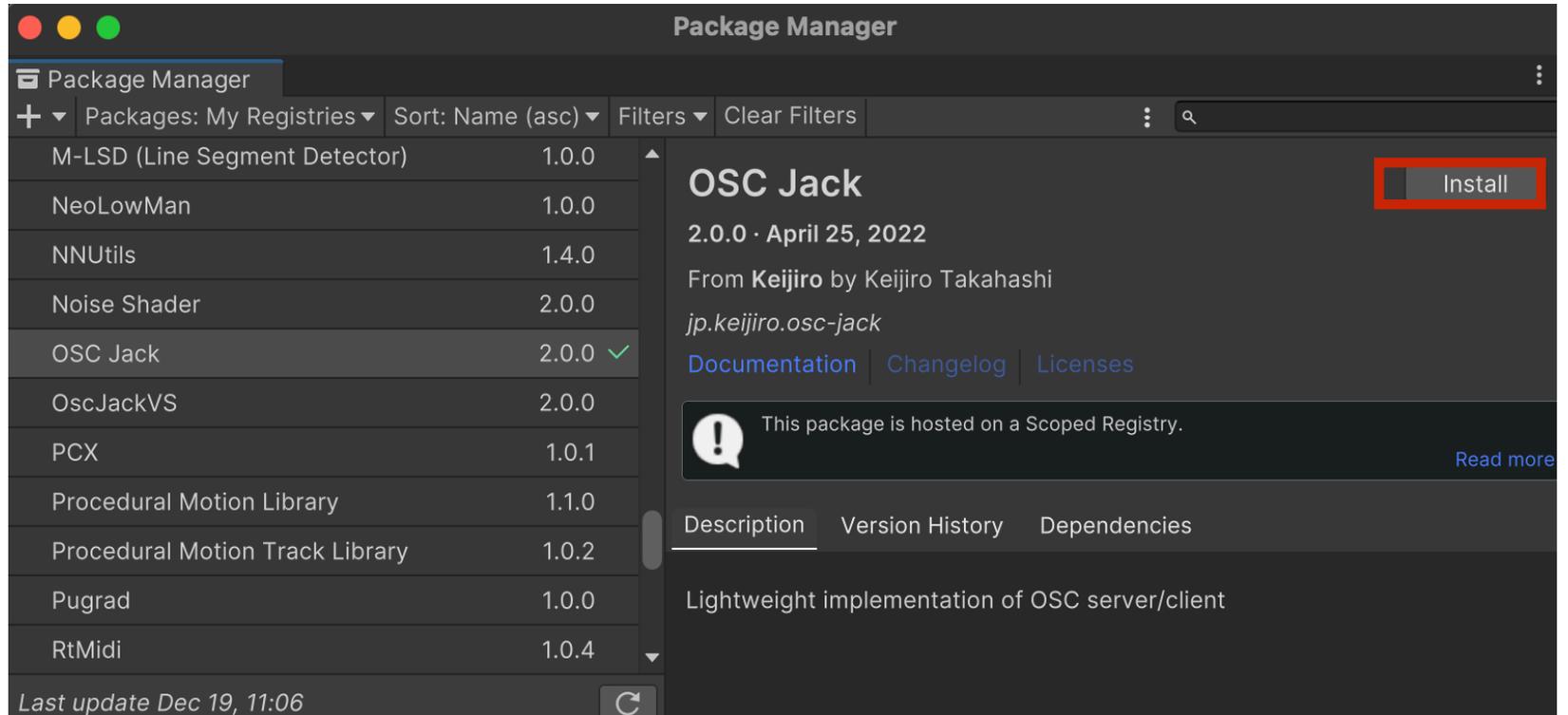
# UNITY側の準備

## 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。



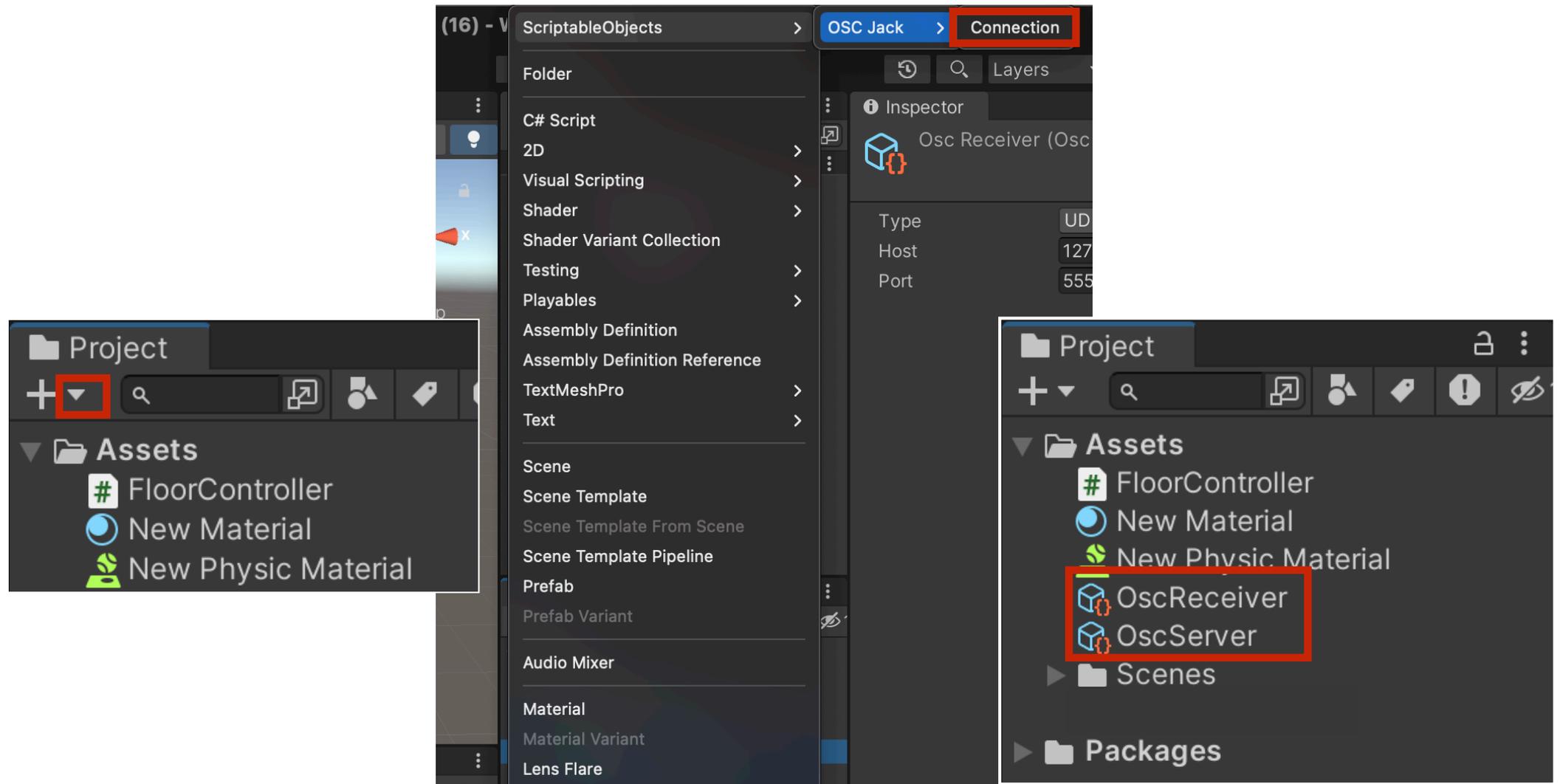
リストから OSCJackを選択し、Installボタンを押す。これでインポートは完了。



# 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

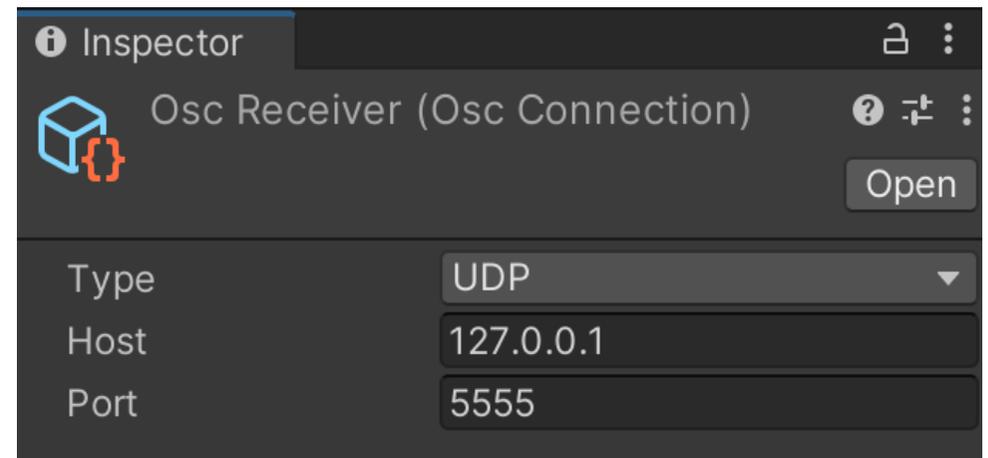
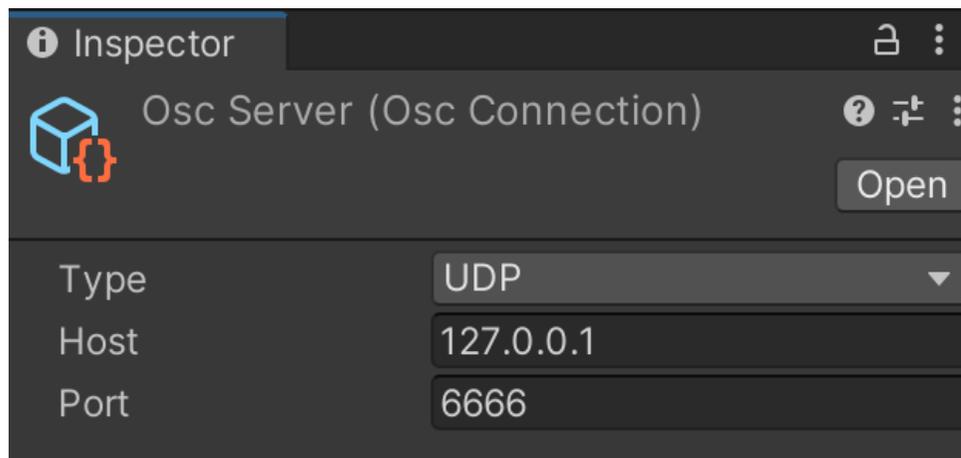
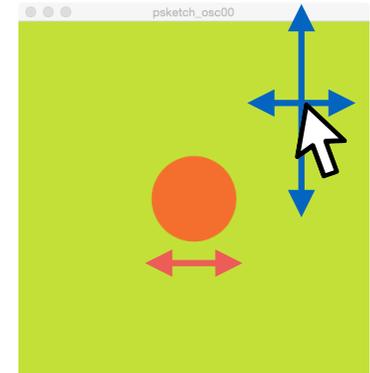
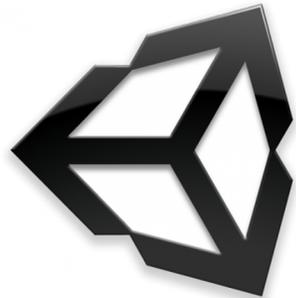
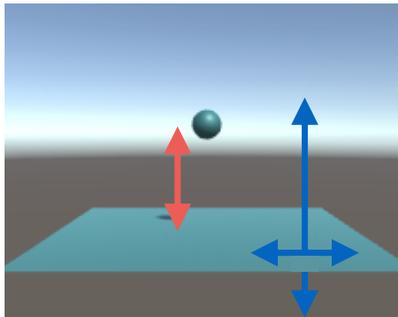
## ポート情報を管理するConnectionの設定

127.0.0.1 / 6666

127.0.0.1 / 5555

SERVER

Client



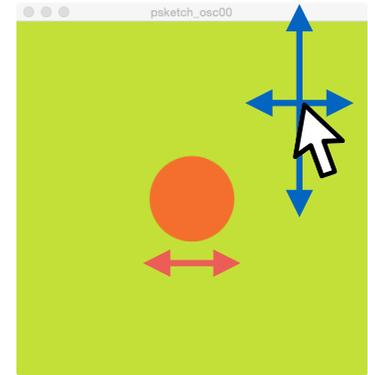
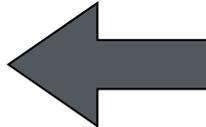
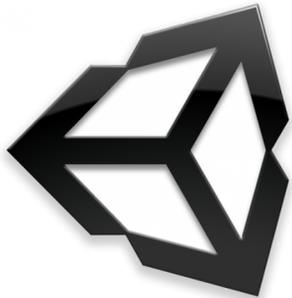
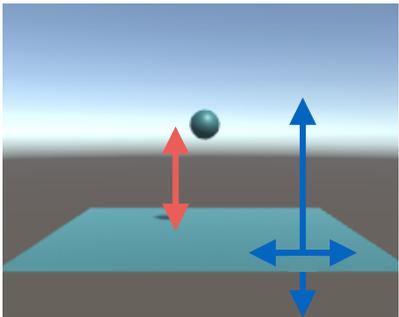
それぞれのオブジェクトのインスペクタに、適切なIPアドレス及びポート番号を登録します。TypeはUDPのままでOKです（OSC通信は全てUDPです）。

127.0.0.1 / 6666

127.0.0.1 / 5555

SERVER

Client



Inspector

Osc Server (Osc Connection)

Type: UDP

Host: 127.0.0.1

Port: 6666

Open

Inspector

Osc Receiver (Osc Connection)

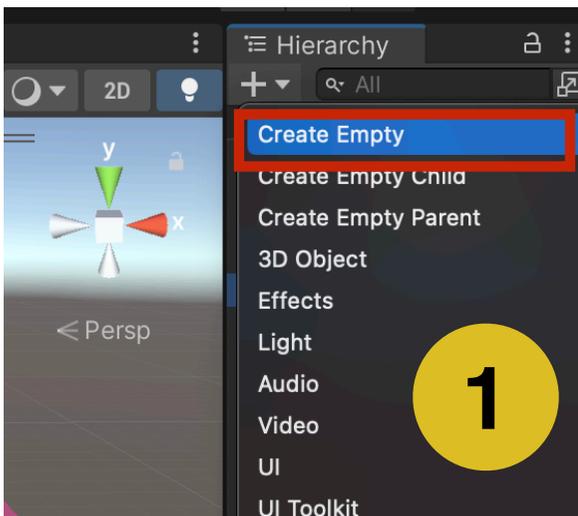
Type: UDP

Host: 127.0.0.1

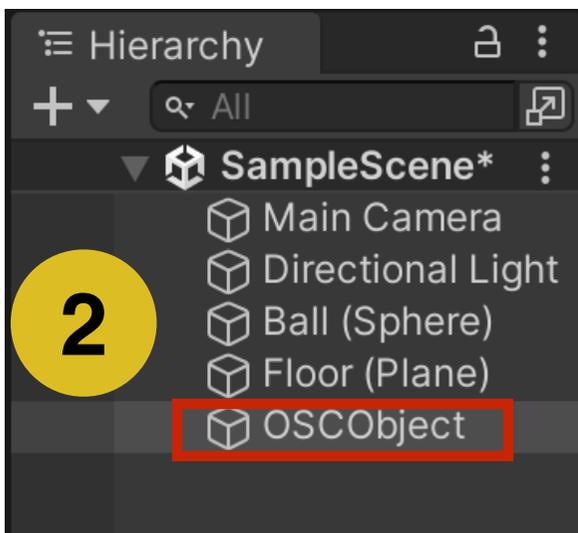
Port: 5555

Open

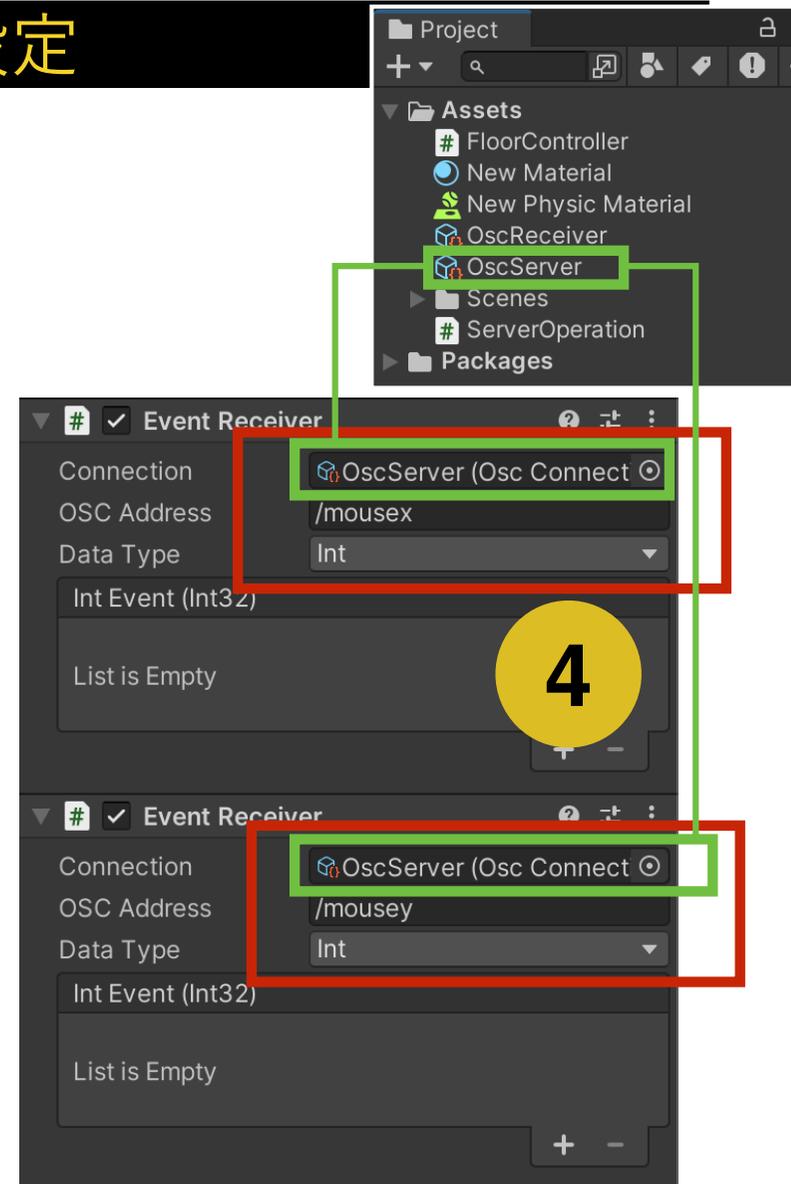
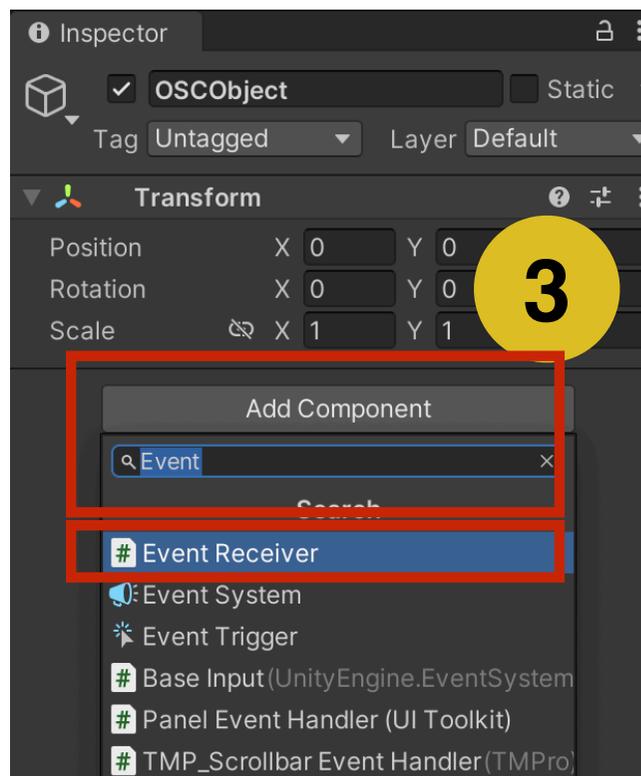
## EventReceiverの設定



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。

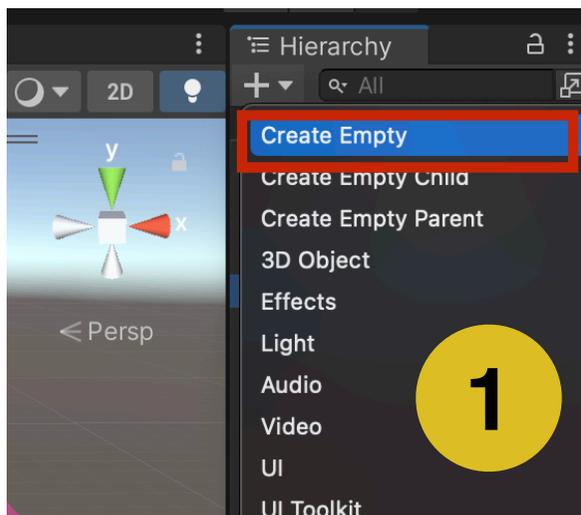


空のゲームオブジェクトを作成し、OSCObjectとリネームする。

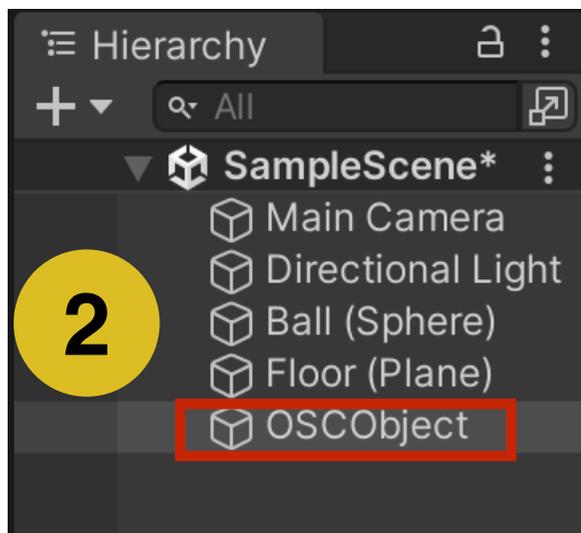


Connectionにすでに作成した「OscServer」をアタッチし、OSCAddressに/mousex、/mouseyを、Data Typeをそれぞれ「Int」に設定しておきます。

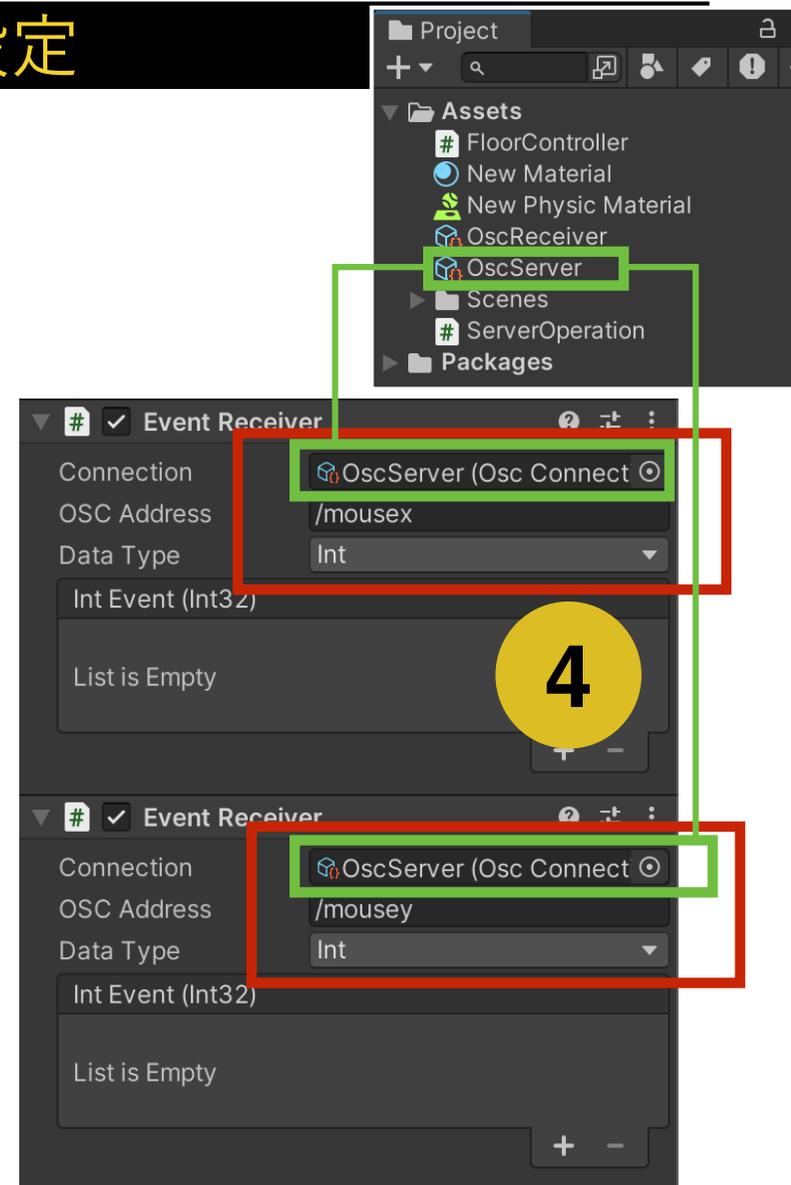
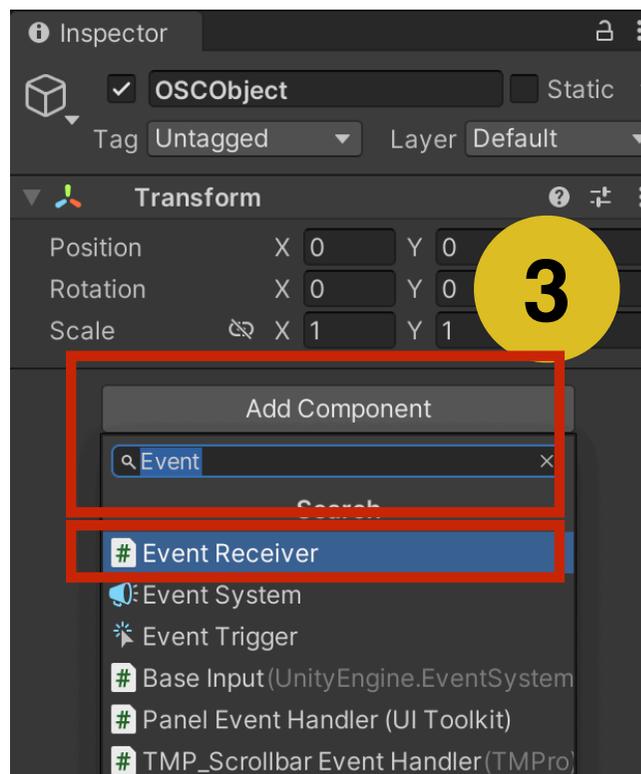
## EventReceiverの設定



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/mousex、/mouseyを、Data Typeをそれぞれ「Int」に設定しておきます。

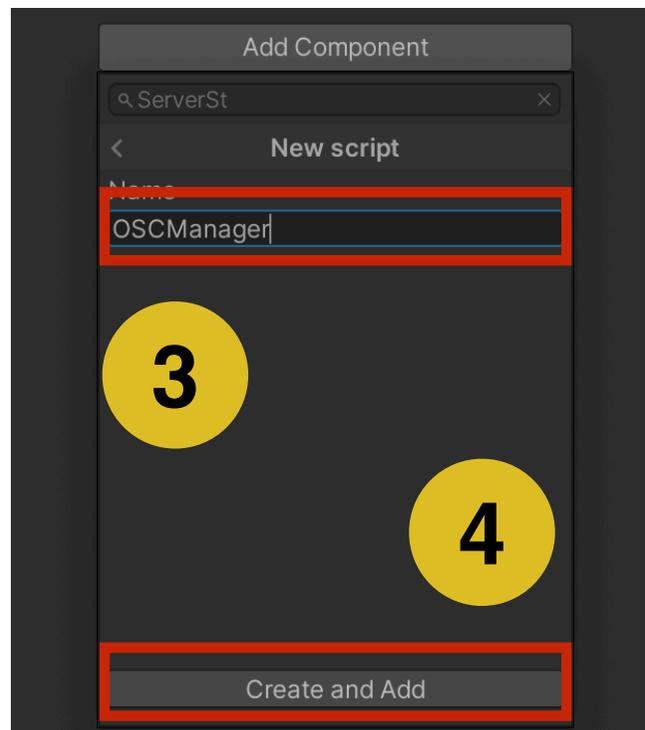
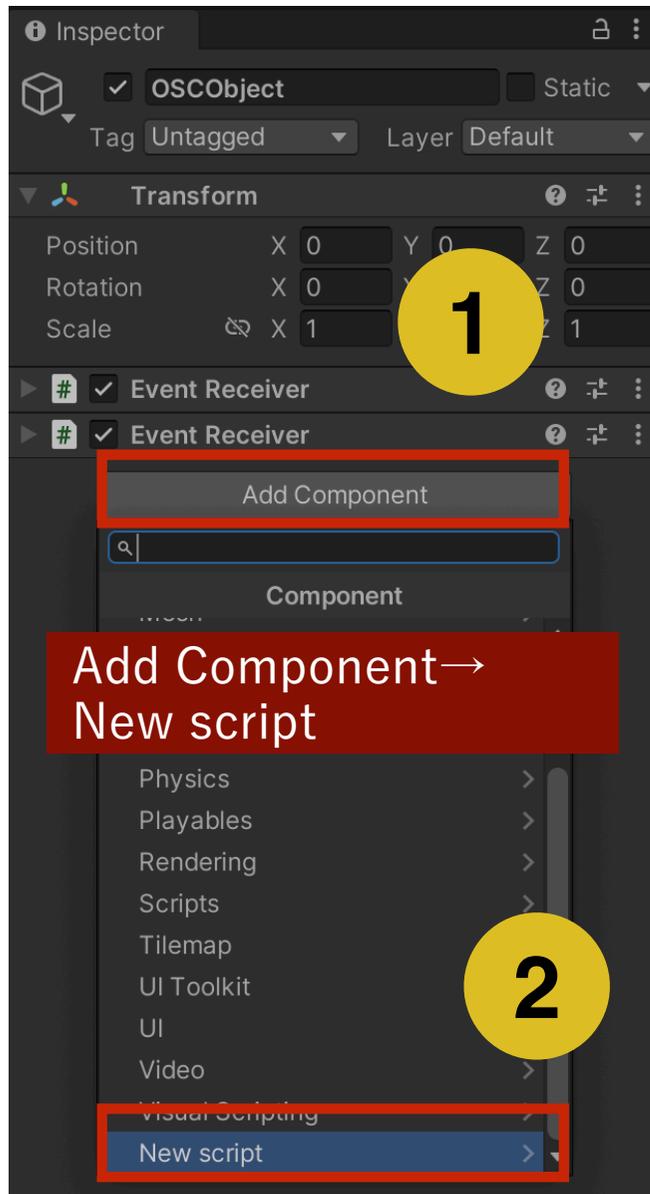
## OSC Monitorによる受信状況の確認

この状態で、Window→OSC Monitorを開いてUnityを走らせ、Processing側の実行ウィンドウをマウスで押すと、Processing側から2種類のマウス座標 (/mousex、 /mousey) が届いていることが確認できます。

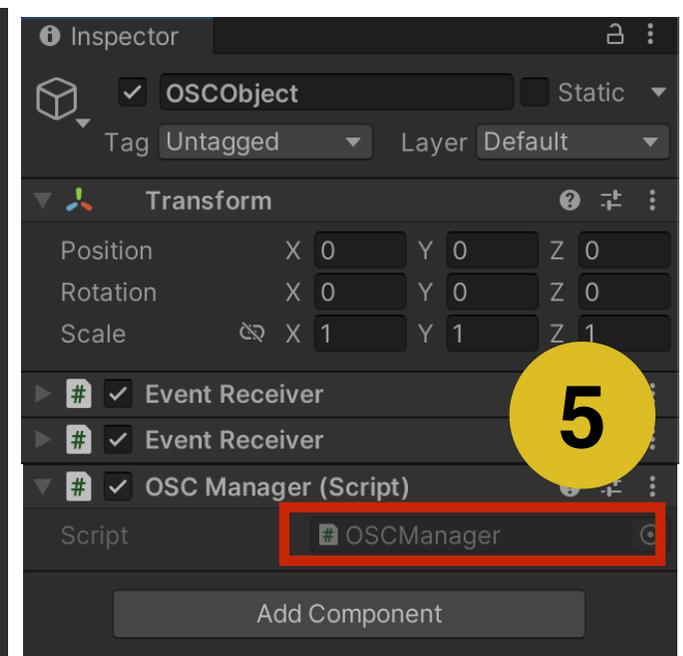
The screenshot displays the Unity 2022.3.14f1 interface. The 'Window' menu is open, and 'OSC Monitor' is selected. The OSC Monitor window shows a list of received OSC messages: /mouse: 89, /mouse: 191, /mouse: 88, /mouse: 192, /mouse: 71, /mouse: 194, /mouse: 58, /mouse: 182, /mouse: 55, /mouse: 122, /mouse: 55, /mouse: 114, /mouse: 55, /mouse: 114, /mouse: 55, /mouse: 114, /mouse: 55, /mouse: 114, /mouse: 47, /mouse: 132, /mouse: 45, /mouse: 147, /mouse: 45, /mouse: 147. A purple callout box labeled 'OSC Monitor' points to this list. The Processing execution window, titled 'sketch\_231218c', is also visible, showing a dark background with three white mouse cursor icons. A red callout box with white text says 'マウスでクリックしながら動かしてみる。' (Click and move with the mouse). A purple callout box labeled 'PROCCSSINGの実行ウィンドウ' (Processing execution window) points to the Processing window. The Unity Inspector shows the OSCObject component with OSC Address /mousex and Data Type Int, and another Event Receiver component with OSC Address /mousey and Data Type Int.

## 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



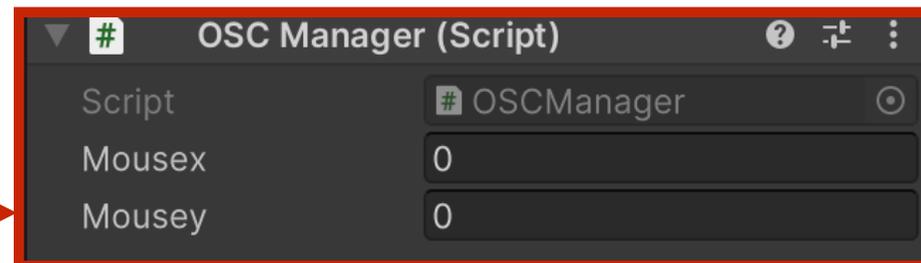
## コールバック関数の作成と関連付け (2/3)

デフォルトのstart関数  
とupdate関数は消去し  
てください。

OSCManager.cs ×

Assets &gt; OSCManager.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class OSCManager : MonoBehaviour
6 {
7     public int mousex = 0;
8     public int mousey = 0;
9
10    public void getMouseX(int x){
11        mousex = x;
12    }
13
14    public void getMouseY(int y){
15        mousey = y;
16    }
17 }
18
```

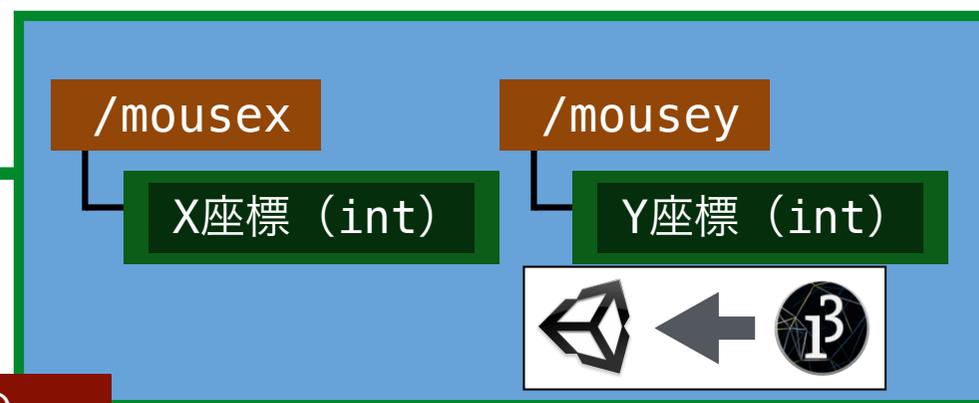


```
public int mousex = 0;
public int mousey = 0;
```

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。

```
public void getMouseX(int x){
    mousex = x;
}

public void getMouseY(int y){
    mousey = y;
}
```



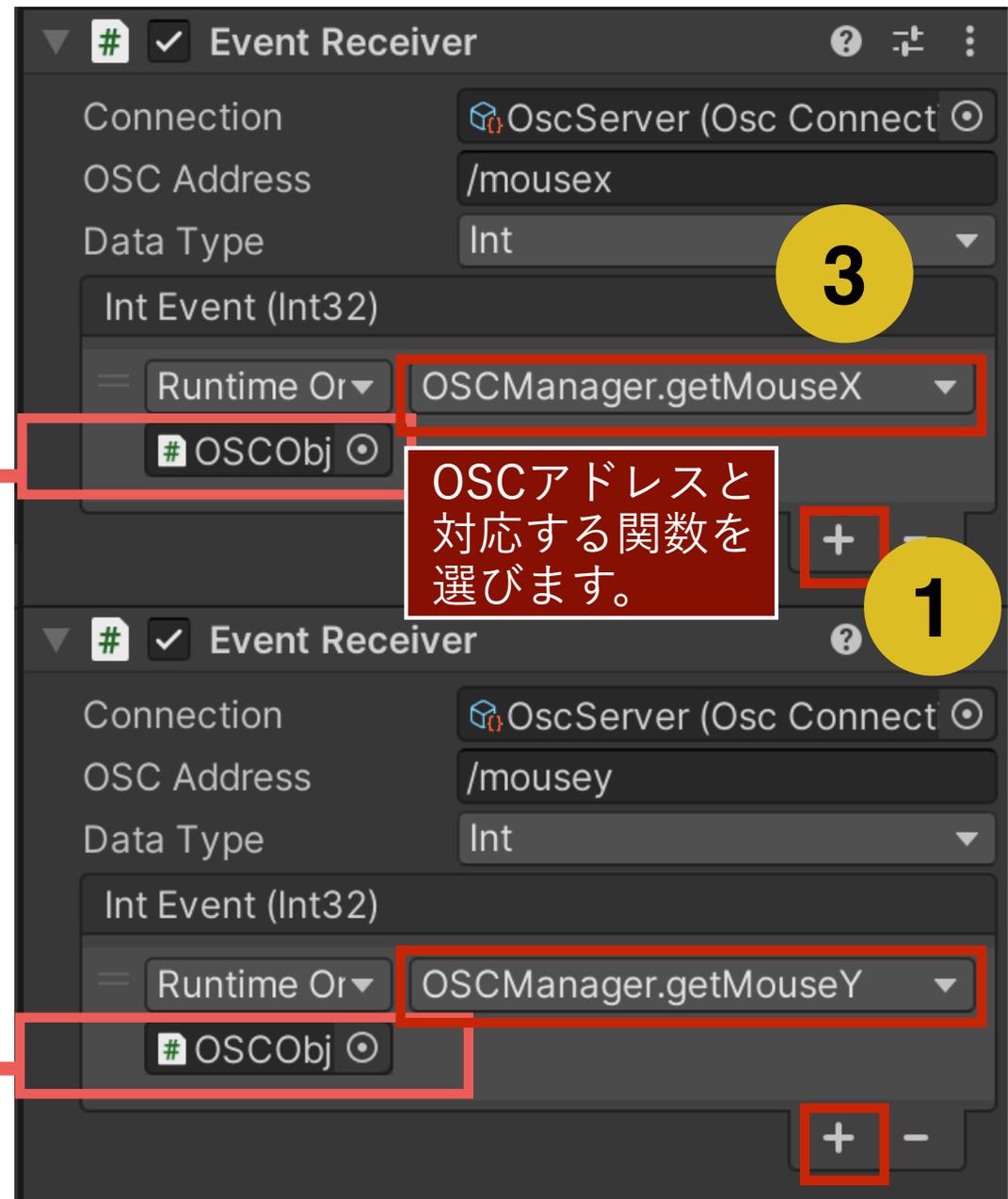
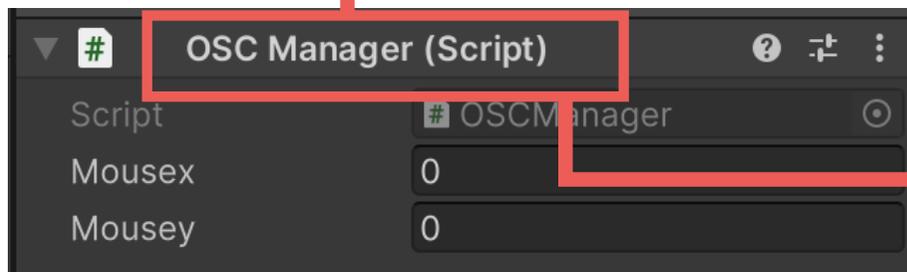
/mousex、/mouseyを受け取るための  
コールアップ関数を作成しておきます。それ  
ぞれ、引数がint型であることに注意。  
(おそらく複数の引数はとれない模様)

## コールバック関数の作成と関連付け (3/3)

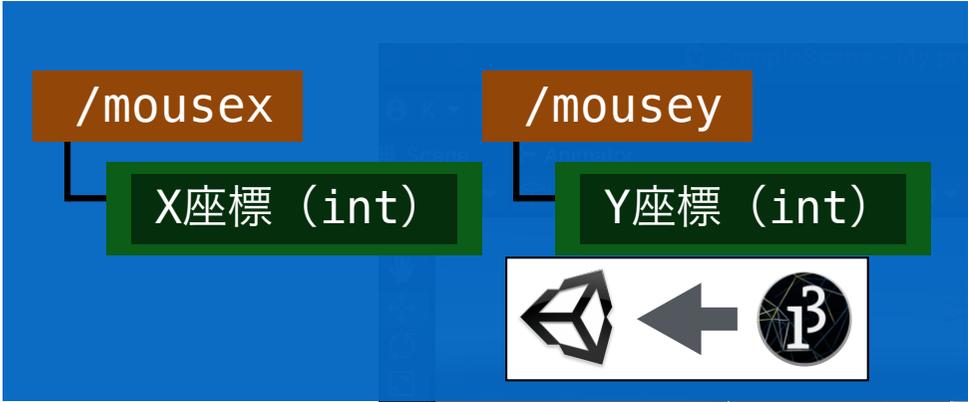
個別のOSCアドレスごとに、すでに作成済みのコールバック関数を対応させていきます。

2

先ほど作成したコールバック関数の入っている「OSCManager」をアタッチします。



## コールバック関数の動作確認



Inspector Panel Configuration:

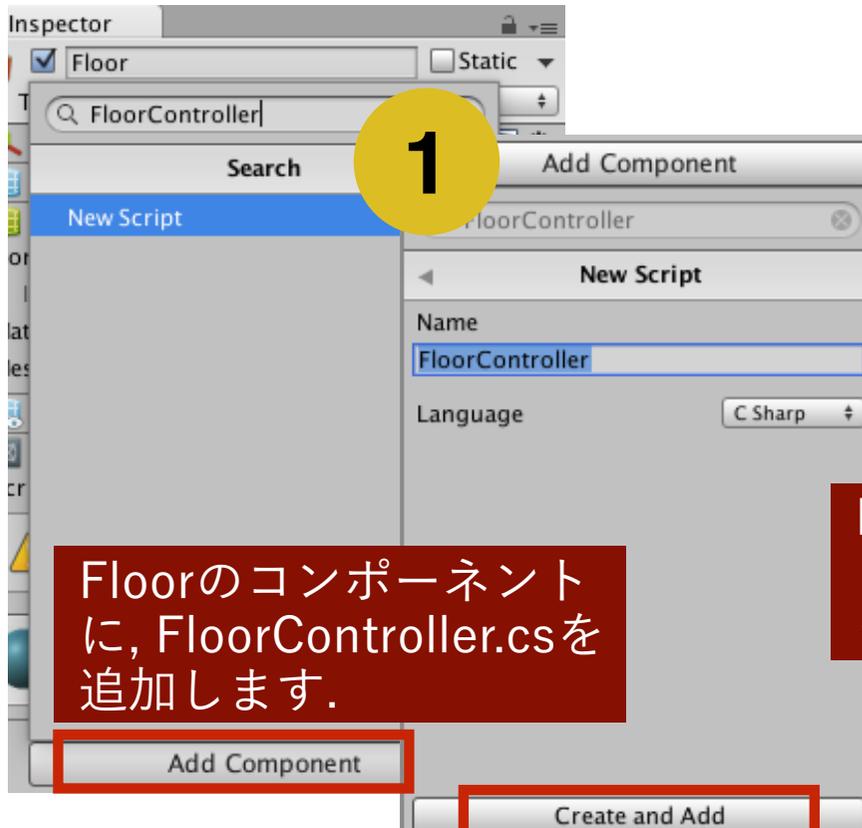
- OSCObject**
  - Tag: Untagged
  - Layer: Default
  - Transform: Position (X: 0, Y: 0, Z: 0), Rotation (X: 0, Y: 0, Z: 0), Scale (X: 1, Y: 1, Z: 1)
- Event Receiver**
  - Connection: OscServer (Osc Connect)
  - OSC Address: /mouseX
  - Data Type: Int
  - Int Event (Int32): Runtime Or OSCManager.getMouseX
- Event Receiver**
  - Connection: OscServer (Osc Connect)
  - OSC Address: /mouseY
  - Data Type: Int
  - Int Event (Int32): Runtime Or OSCManager.getMouseY
- OSC Manager (Script)**
  - Script: OSCManager
  - MouseX: 274
  - MouseY: 240

PROCSSINGの実行ウィンドウ

ここでProcessing側のマウス座標が、リアルタイムでOSCManagerの2つのパブリック変数の値に反映されていることを確認してください。



## FloorController.csコンポーネントの作成



1

ドラッグ&ドロップ

Projectファイルで作成したものを直接にアタッチしてもOKです。

2

Floorのコンポーネントに、FloorController.csを追加します。

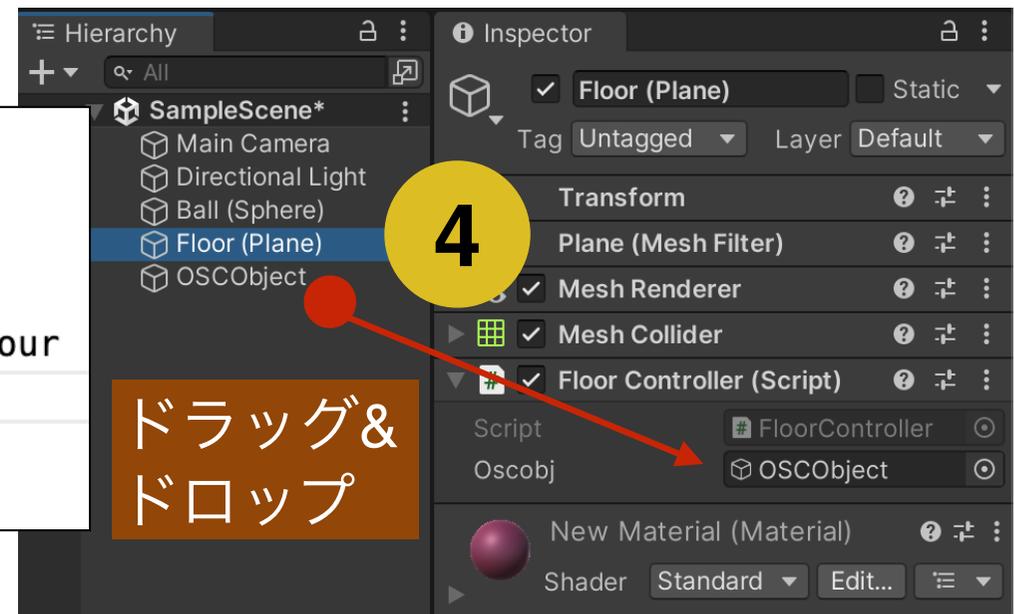
Add Component

Create and Add

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class FloorController : MonoBehaviour
6 {
7     public GameObject oscobj = null;
8 }
```

3

スクリプト上で、ゲームオブジェクトをpublic変数として追加



4

ドラッグ&ドロップ

## FloorController.csを編集

FloorController.cs

Assets &gt; FloorController.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class FloorController : MonoBehaviour
6 {
7     public GameObject oscobj = null;
8     private OSCManager oscmng;
9
10
11 void Start()
12 {
13     oscmng = (OSCManager)oscobj.GetComponent("OSCManager");
14 }
15
```

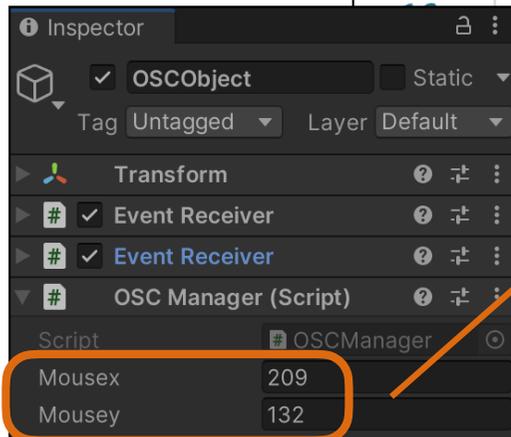
OscObjectから, OSCManagerコンポーネントを取り出しています。(おまじないと思ってください)

```
void Start()
```

```
{
|
|
}
```

```
    oscmng = (OSCManager)oscobj.GetComponent("OSCManager");
```

初期化処理



```
void Update()
```

```
{
|
|
}
```

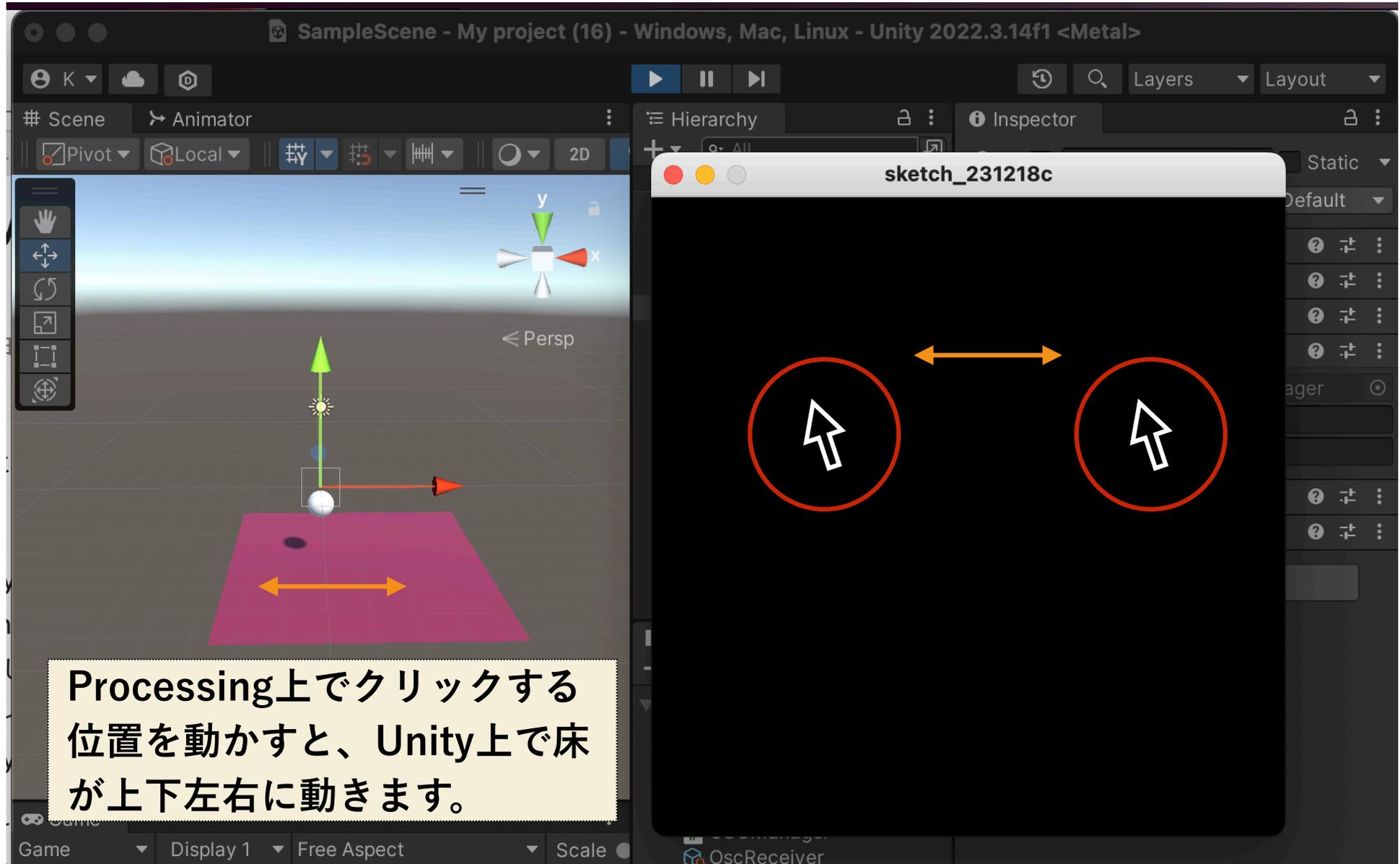
```
    float x = (oscmng.mousex - 200) * 0.01f;
    float y = 0f - oscmng.mousey * 0.01f;
    this.transform.position = new Vector3(x,y,0f);
```

Processing上でクリックしたXY座標を、床のXY座標にマッピングしています。

OSCObjectの2つのpublic変数にアクセスしています。

毎フレーム行われる処理を記述します。

## FloorController.csを編集

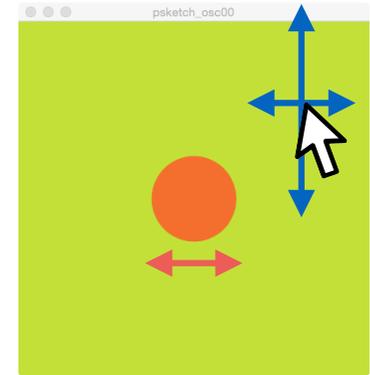
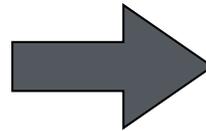
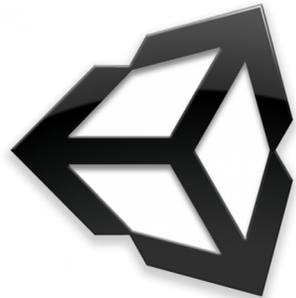
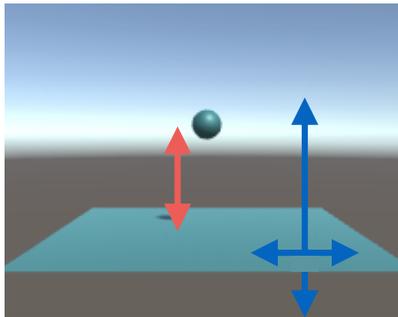


127.0.0.1 / 6666

127.0.0.1 / 5555

SERVER

Client



Inspector

Osc Server (Osc Connection)

Type: UDP

Host: 127.0.0.1

Port: 6666

Open

Inspector

Osc Receiver (Osc Connection)

Type: UDP

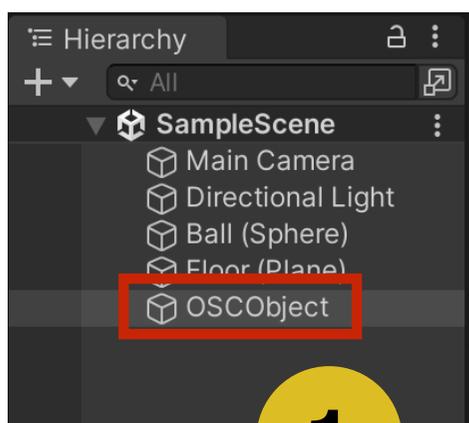
Host: 127.0.0.1

Port: 5555

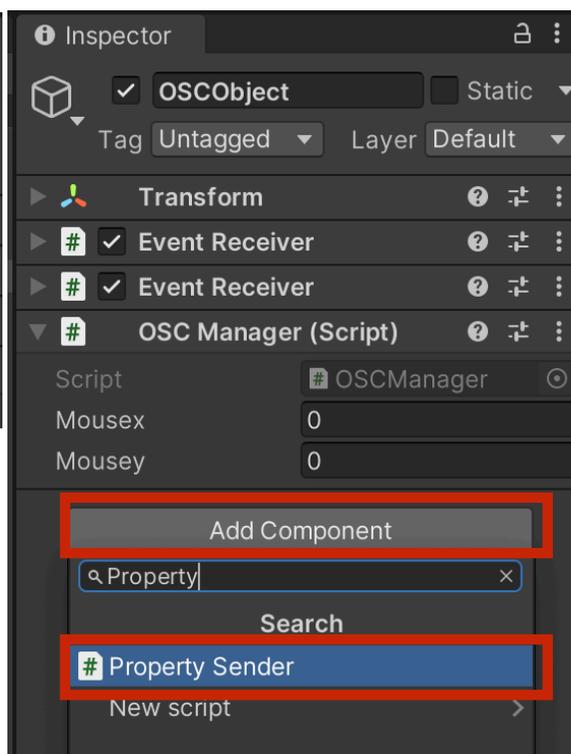
Open

## GameObjectの三次元位置 (XYZ) を送信する (1/2)

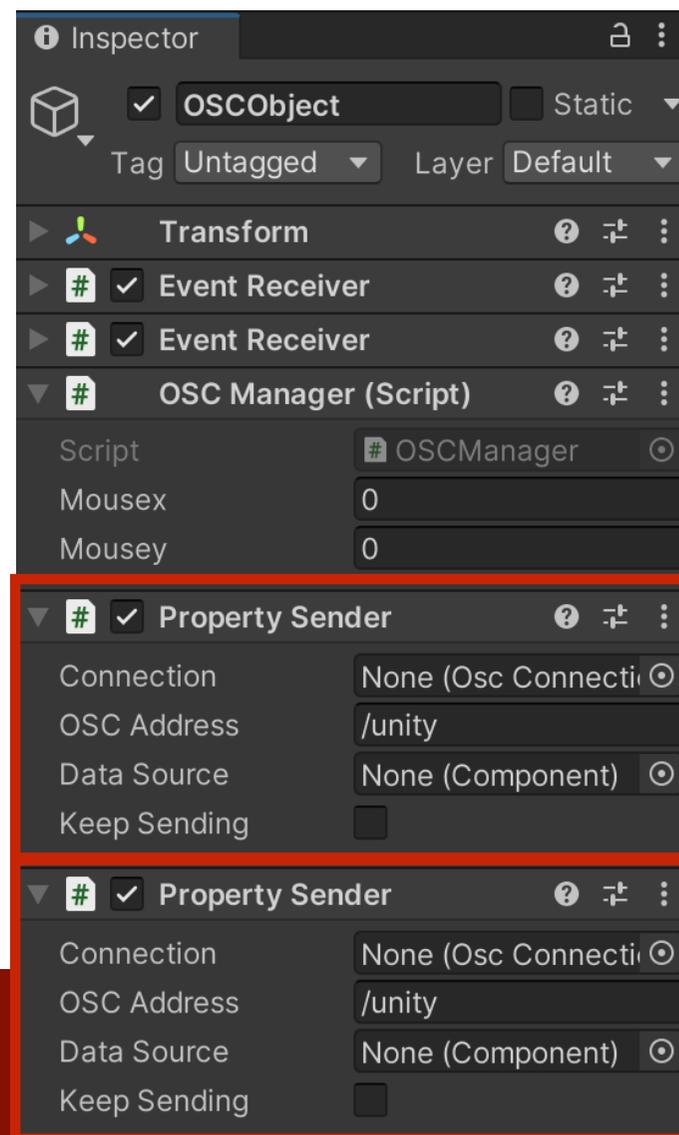
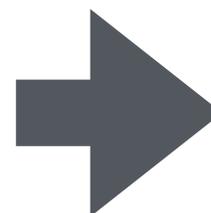
スクリプトを使わずに、OSC通信を行うためには、コンポーネントであるProperty Senderを使います。ここでは、特定のゲームオブジェクトの位置情報 (3つのfloat値) を送信する方法を解説します。



OSCObjectを選択した状態で、



Add ComponentでProperty Senderを追加します。



2つ追加してください。初期状態では右のようになります。

## 床と球の三次元位置 (XYZ) を送信する (2/2)

The image shows two side-by-side screenshots of the Unity Inspector's Property Sender window. The left window is configured for the 'Ball (Sphere)' object, with the OSC Address set to '/pos/ball'. The right window is configured for the 'Floor (Plane)' object, with the OSC Address set to '/pos/floor'. Both windows have 'Keep Sending' checked and 'Value changes only' unchecked. A red box highlights the 'OscReceiver (Osc)' component in both. A yellow box highlights the 'Ball (Sphere) (Transform)' and 'Floor (Plane) (Transform)' data sources. A red callout box with the text '三次元位置を送信する' (Send 3D position) points to the 'position' property dropdown in the right window. Red lines connect the 'OscReceiver' component in the Asset Hierarchy to the Property Sender window in both screenshots.

`/pos/floor, /pos/ball`

床とボールのXYZ座標 (float, float, float)

球と床の距離を円の直径と連動させます。

## 衝突情報を衝突のタイミングで送信する

スクリプトからOSC通信の送信を行う方法です。赤枠部分は、FloorController.csの中で新たに追記した部分です。

```

FloorController.cs ×
Assets > FloorController.cs
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using OscJack;   OscJackのライブラリを使います。
5
6 public class FloorController : MonoBehaviour
7 {
8     public GameObject oscobj = null;   送信先のOSC情報を
9     private OSCManager oscmng;       まとめたものです。
10
11     [SerializeField] string ipAddress = "127.0.0.1";
12     [SerializeField] int port = 5555;
13     OscClient client;
14
15     int count = 0;
16
17     void Start()
18     {
19         oscmng = (OSCManager)oscobj.GetComponent("OSCManager");
20         client = new OscClient(ipAddress, port);
21
22     }

```

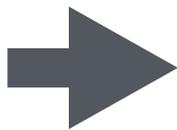
Processing側の端末をオブジェクト化したものです。

```

24 void Update()
25 {
26     float x = (oscmng.mousex - 200) * 0.01f;
27     float y = 0f - oscmng.mousey * 0.01f;
28     this.transform.position = new Vector3(x,y,0f);
29 }
30
31 void OnDisable()
32 {
33     client.Dispose();   ネットワークが壊れた時の
34 }                       処理？おまじないと
35                               思って書いて下さい。
36
37 void OnCollisionEnter(Collision collision){
38     count++;
39     Debug.Log("Collision!!" + count);
40     client.Send("/hit", count);
41 }

```

自分の配置されているゲームオブジェクト（この場合、床）が何かと衝突したタイミングで呼ばれるコールアップ関数の中に、OSC通信の処理を書き込みます。ここでは /hit のOSCアドレスに、衝突のcountを送っています。



/hit

衝突回数 (int)

衝突に合わせて、背景色を変化。

## 衝突情報を衝突のタイミングで送信する

OSC通信を行う本体のメソッドはSendです。送信できる引数の形式は以下の通り。

```
void OnCollisionEnter(Collision collision){  
    count++;  
    Debug.Log("Collision!!" + count);  
    client.Send("/hit", count);  
}
```

OscClient#Send のオーバーロードメソッドは次のようになっているように、送信できる形式は1～4個のint、1～4個のfloatまたは1つのstringです。

```
public void Send(string address);  
public void Send(string address, int data);  
public void Send(string address, int element1, int element2);  
public void Send(string address, int element1, int element2, int element3);  
public void Send(string address, float data);  
public void Send(string address, float element1, float element2);  
public void Send(string address, float element1, float element2, float element3);  
public void Send(string address, float element1, float element2, float element3, float element4);  
public void Send(string address, string data);
```

[https://qiita.com/aa\\_debdeb/items/34174f07909fa324e09c](https://qiita.com/aa_debdeb/items/34174f07909fa324e09c)