演習：フィジカル・コンピューティング

11/27 ３限

演習１｜ブレッドボードに親しむ

演習２｜センサーの状態を「電圧計」から読み取る

11/27 ４限

演習３｜Arduino（センサー）

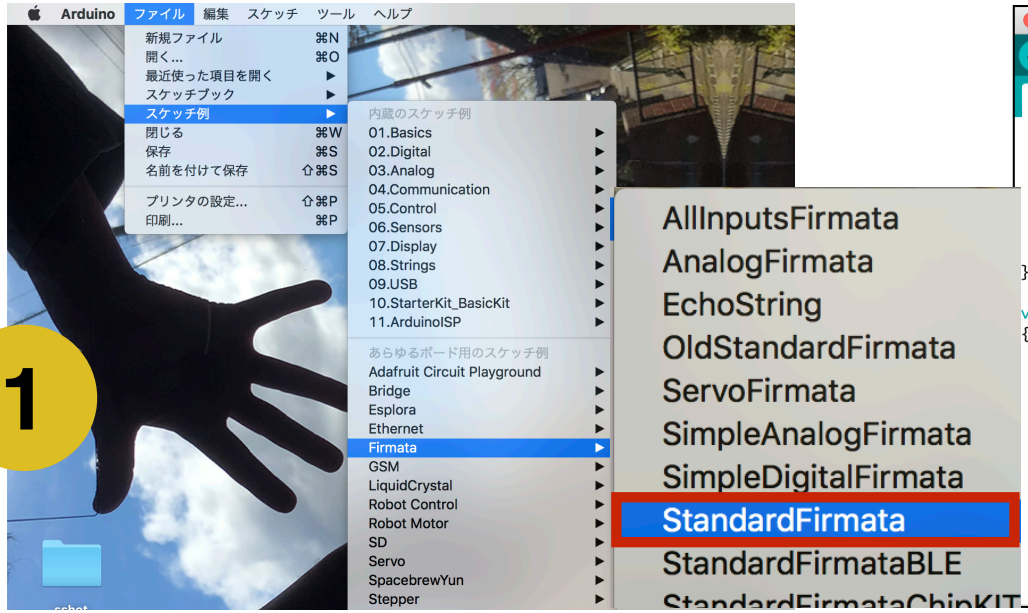
12/04 ３〜４限

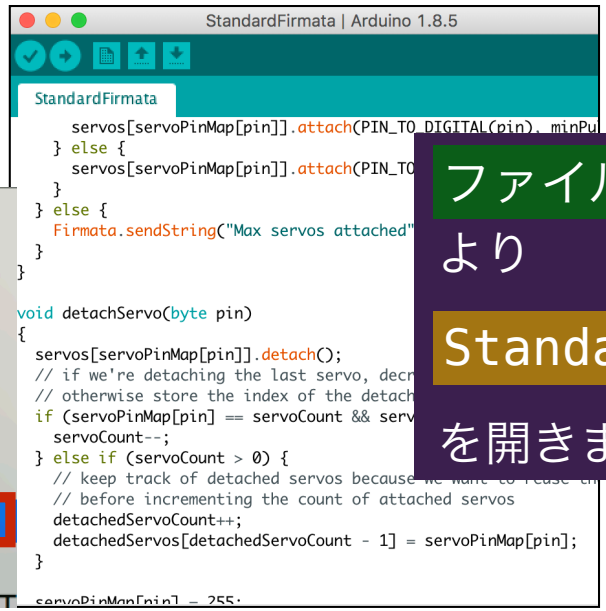
演習４A｜Arduino - Processing

演習４B｜Unity - Processing
（アプリケーション間通信）

担当：小鷹

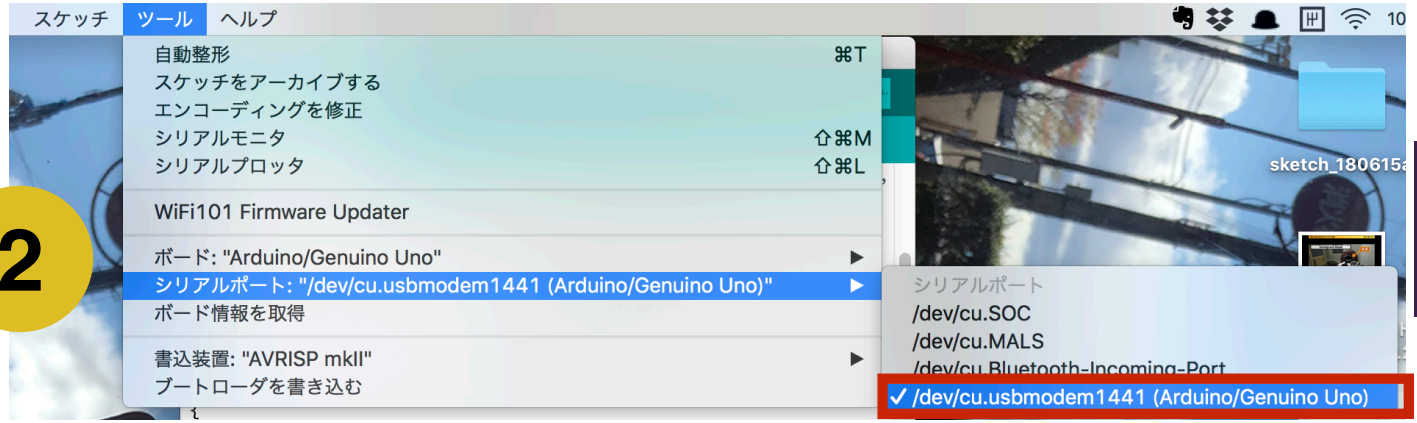
Firmataの導入（Arduino側）





1

ファイル／スケッチ例
より
StandardFirmata
を開きます.



2

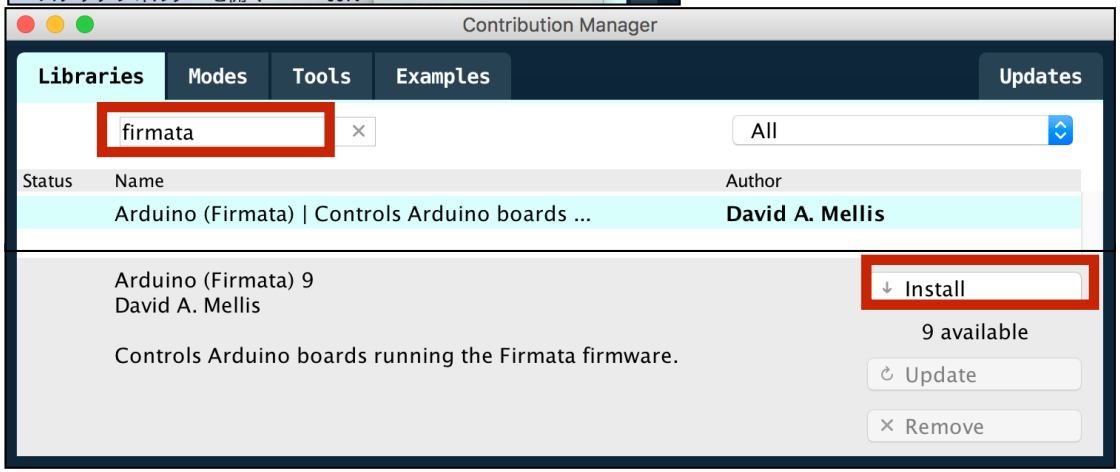
適切なシリアルポートを選択するとともに，ポート名を覚えておいてください.

3

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）

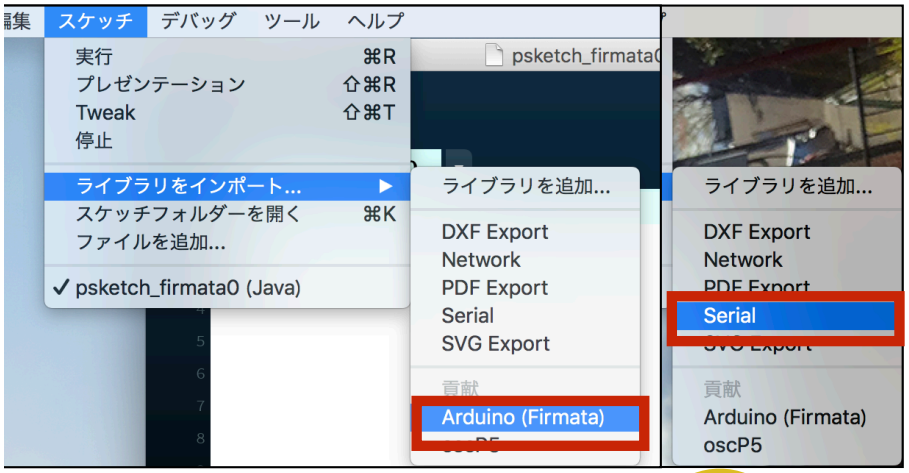
スケッチ／ライブラリをインポート／ライブラリを追加

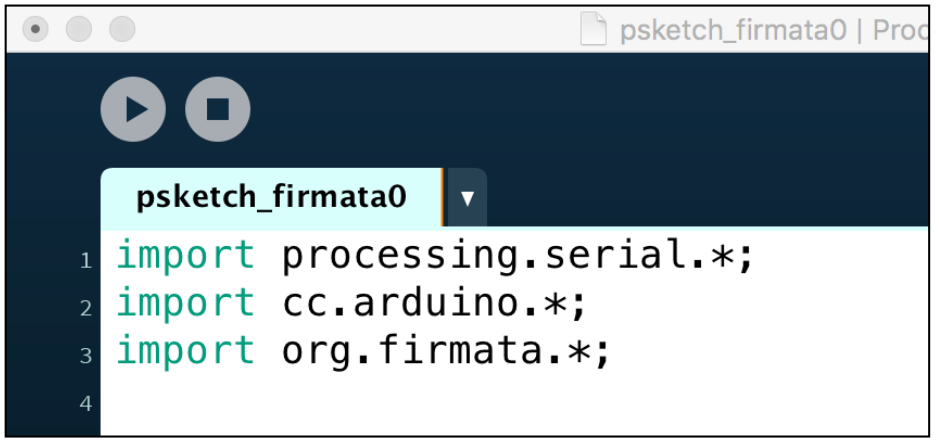


1

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）





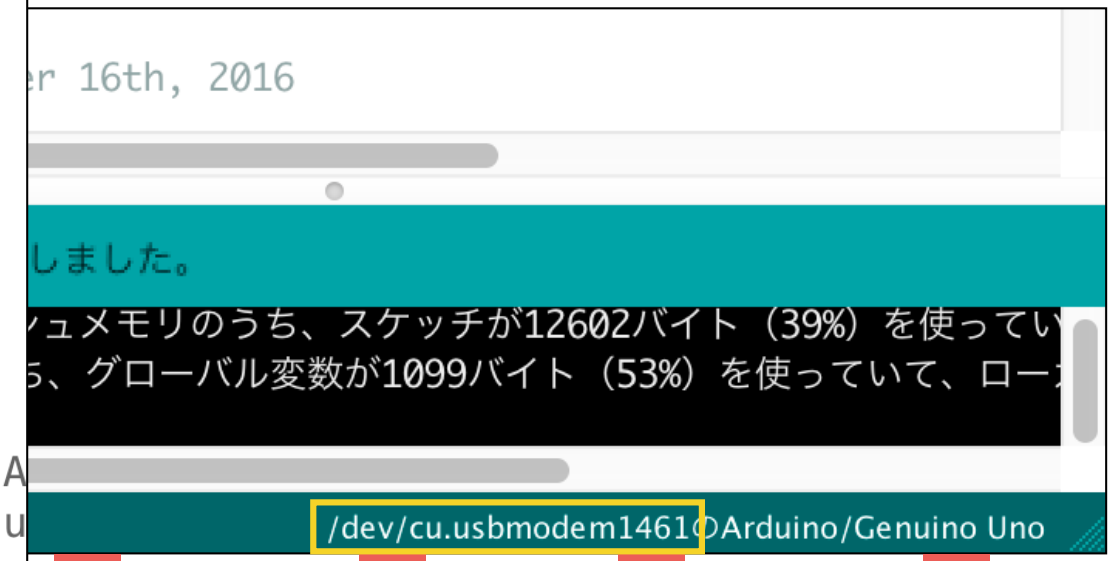
2

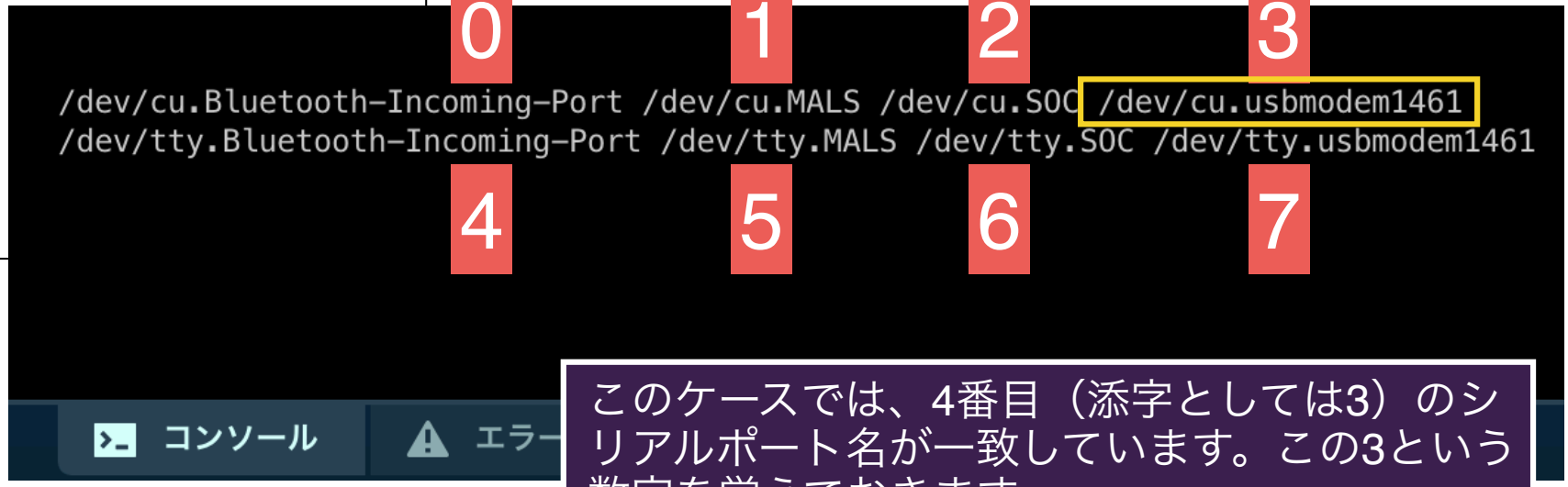
Arduino（Firmata）とSerialの二つのライブラリをインポートしてください.

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.





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）

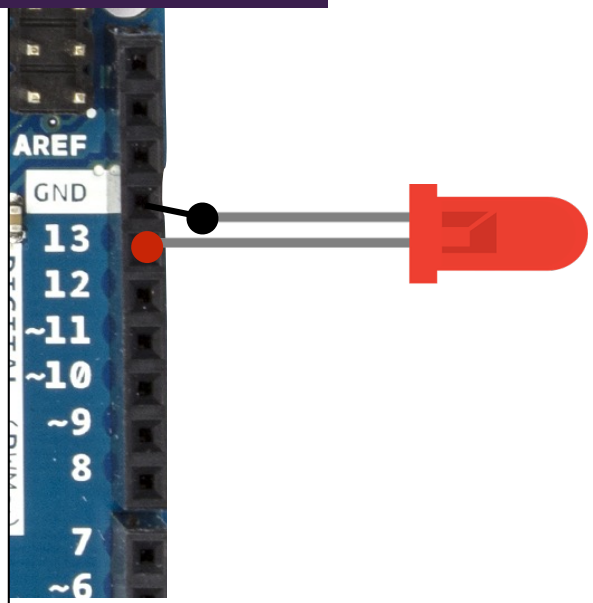
マウスを押す／離すによって，LEDのON・OFFが切り替わり，同時に画面の背景も黒から赤に変わります．なお，実行の際には，Arduinoは終了しておきます（USBをProcessingのために解放するためです）．

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

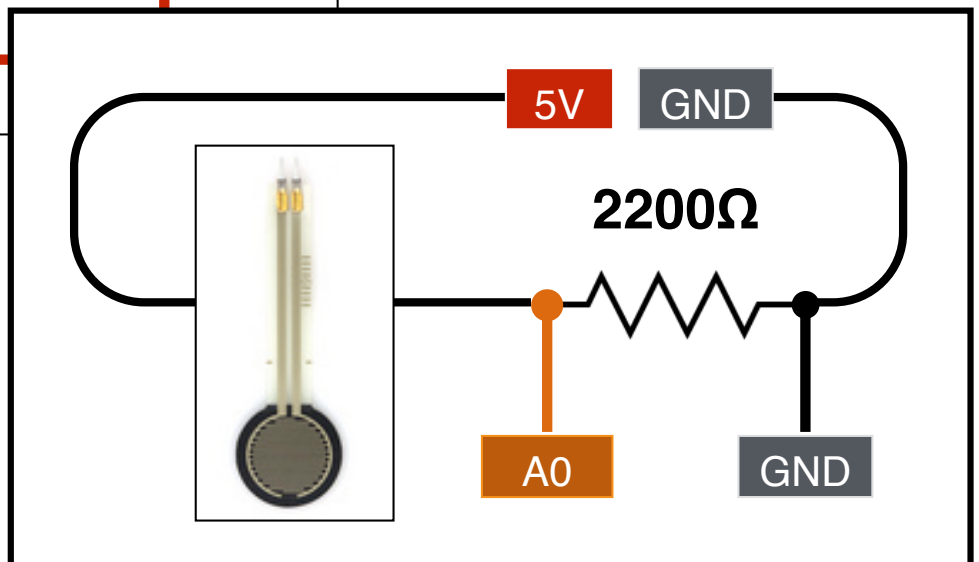
psketch_Touch.pde

```
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 }
```

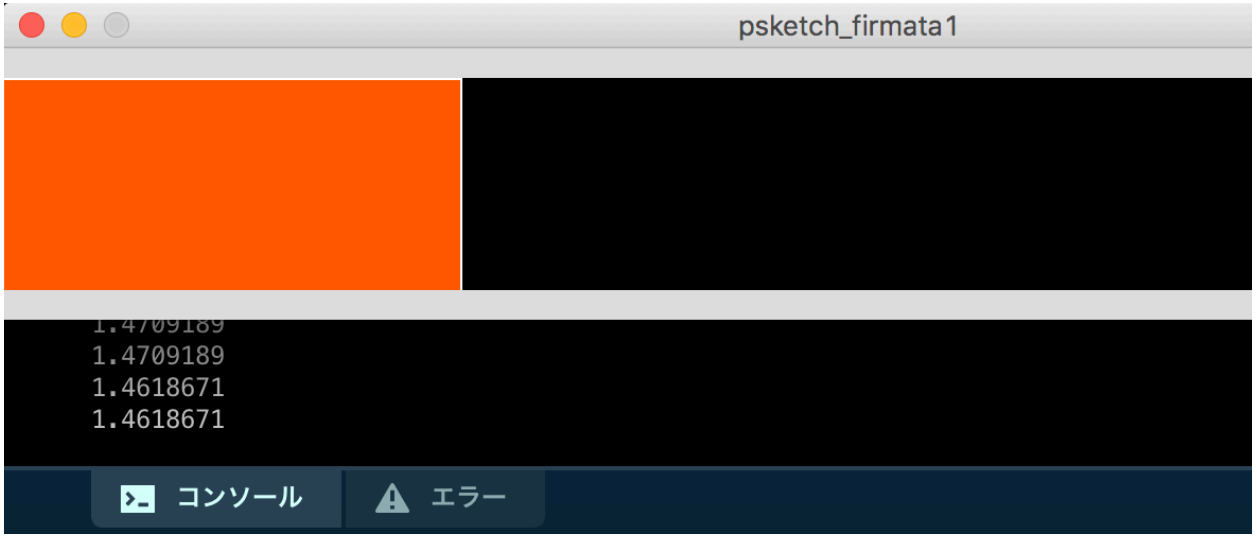
ArduinoのANピンは, Processingでは整数型のNに対応します.

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.

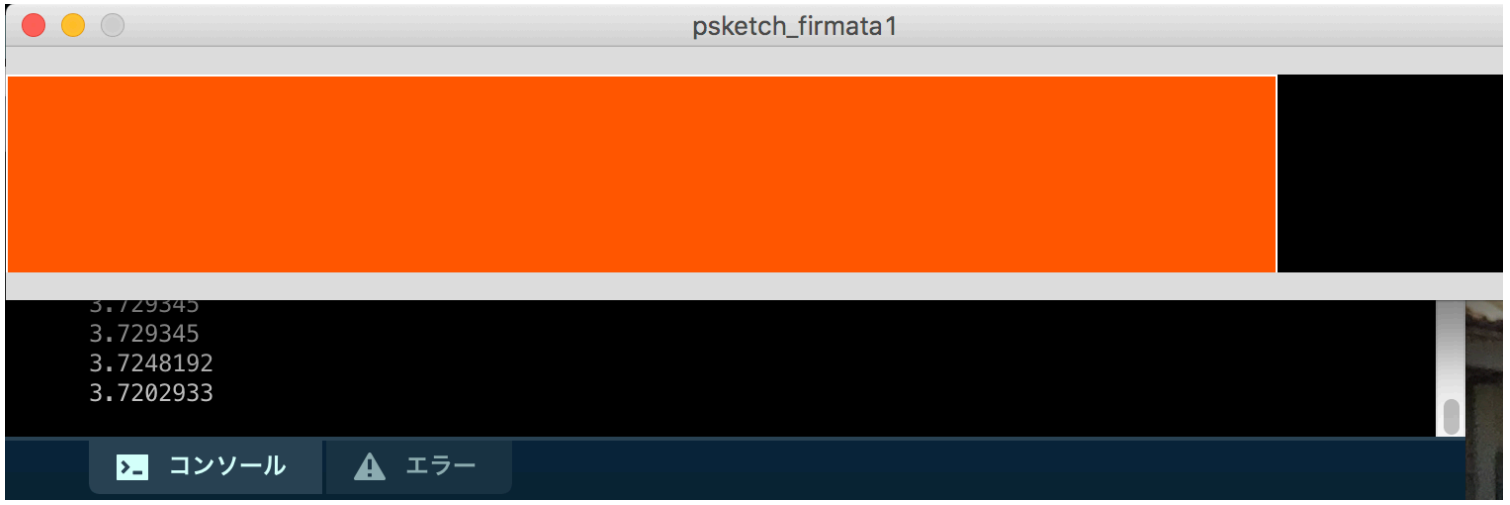


圧力センサの値をビジュアライズする



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.

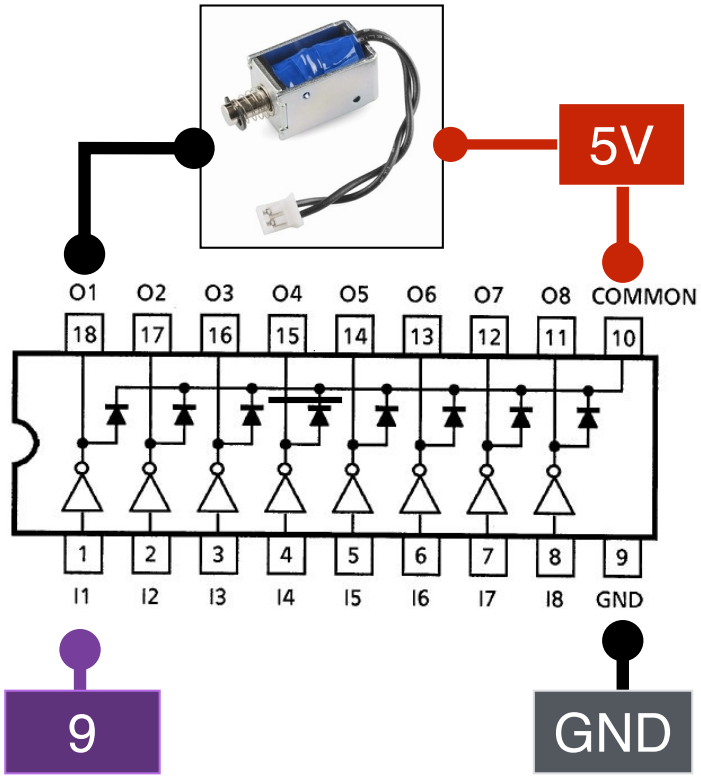


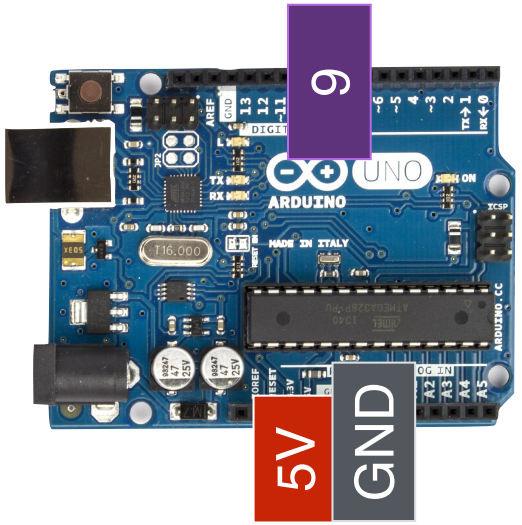
ソレノイドをGUIで動かす

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.

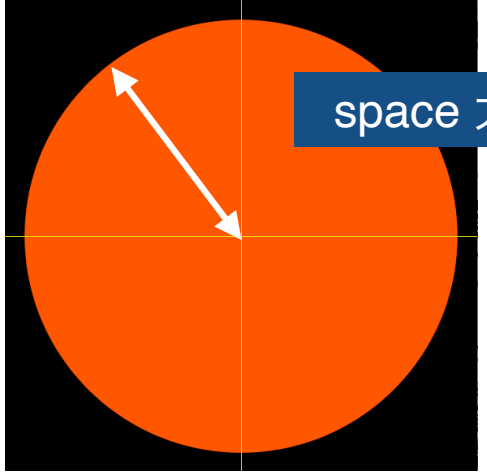
```
21 void draw(){
22
23   float t = millis() - t_last; //経過時間
24
25   //経過時間が休符時間長を超えた場合
26   if(t>space){
27
28     if(val==255){
29       val = 0;
30     }else{
31       val = 255;
32     }
33     arduino.analogWrite(sorPin, val);
34
35     t_last = millis(); //出力時刻の更新
36
37   }
38
39   if(mousePressed){
40     float rad = dist(mouseX, mouseY, cx, cy);
41
42     background(0);  fill(255,100,0);  noStroke();
43     ellipse(cx,cy,2*rad,2*rad);
44
45     stroke(255,255,0);
46     line(cx,0,cx,height); line(0,cy,width,cy);
47
48     space = rad;
49   }
50
51 }
```

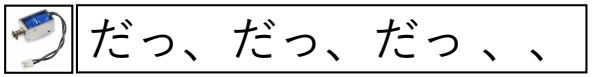
draw()

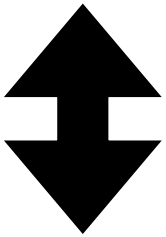
経過時間が space を超えた場合, ソレノイドの値を反転し, t_lastを現在の時間(ms)に更新します.

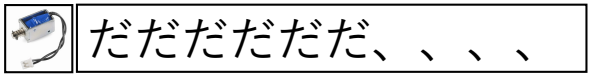
中心からマウスまでの距離に応じて, 円の大きさと space を変えます.

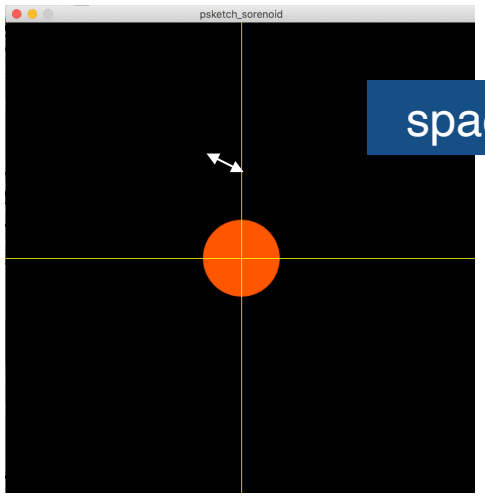
psketch_sorenoid.pde (2/2)





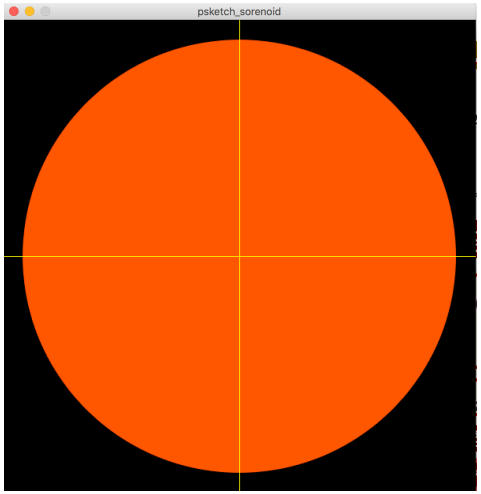


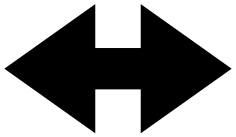


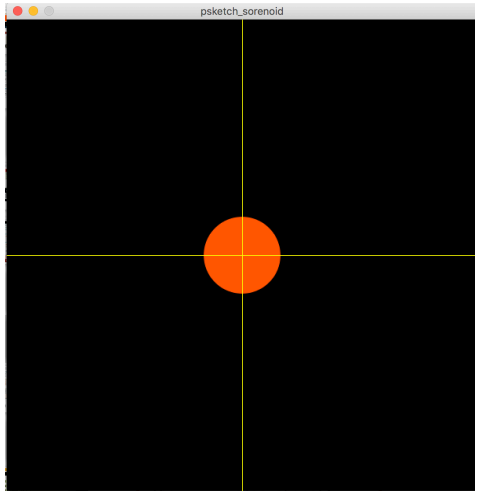


練習 1

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.

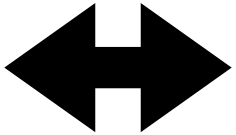






圧力センサ

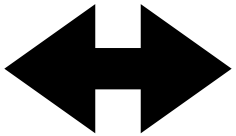
押さない



強く押す

ソレノイド

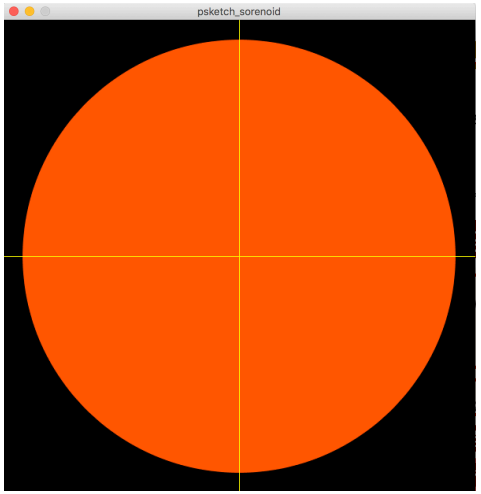
だっ、だっ、だっ、、

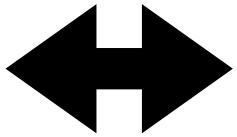


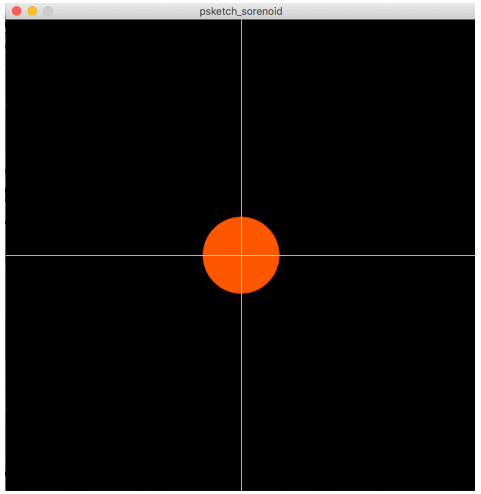
だだだだだだだっ、、

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.

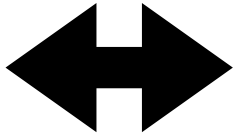


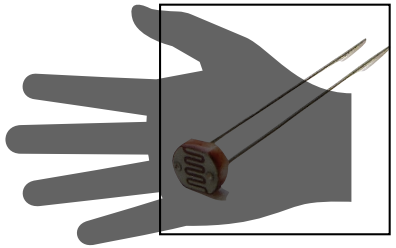




照度センサ

隠さない

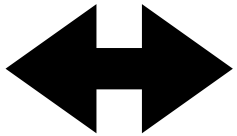




手で隠す

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

11/27 ３限

演習１｜ブレッドボードに親しむ

演習２｜センサーの状態を「電圧計」から読み取る

11/27 ４限

演習３｜Arduino（センサー）

12/04 ３〜４限

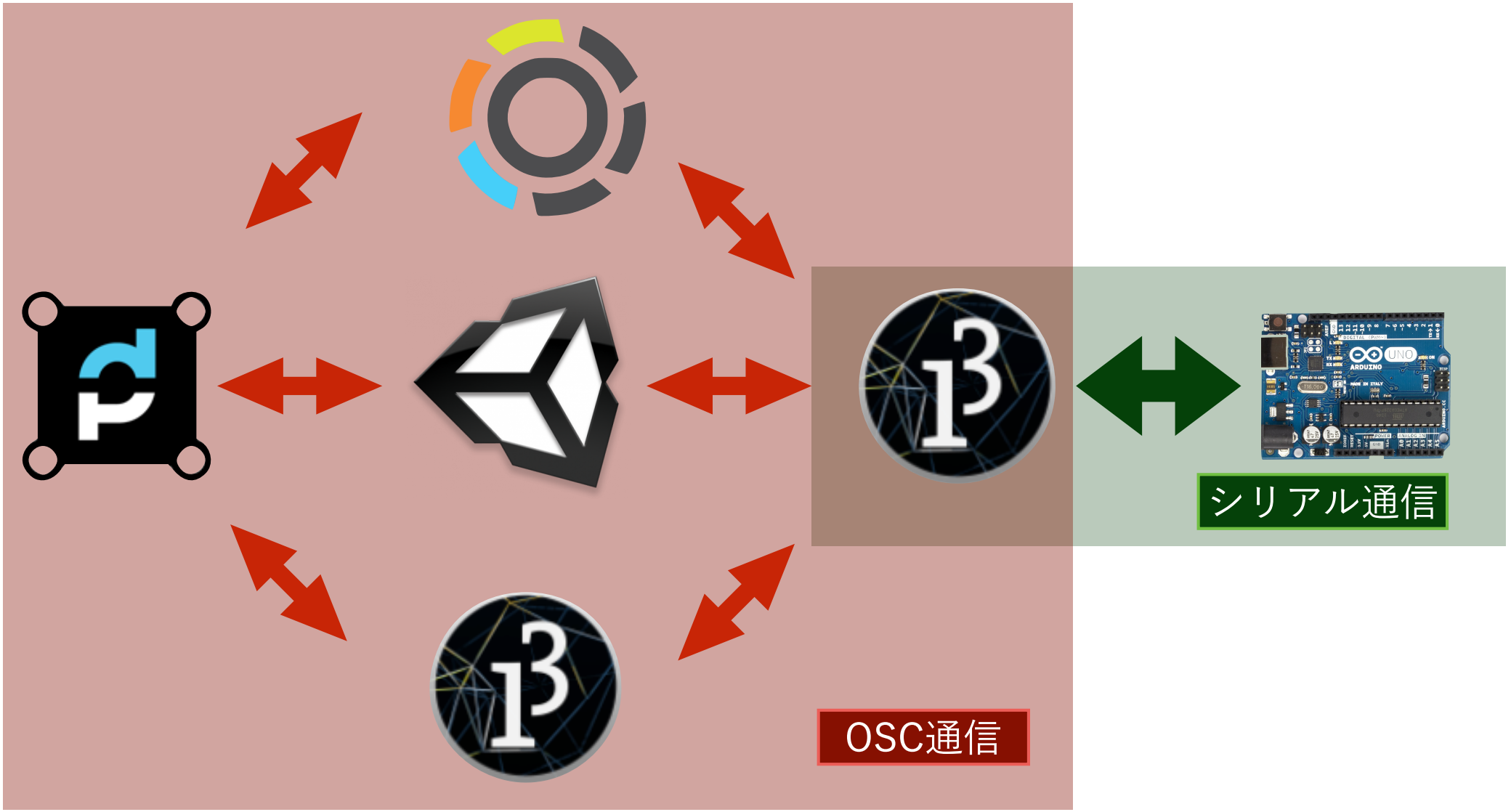
演習４A｜Arduino - Processing

演習４B｜Unity - Processing

（アプリケーション間通信）

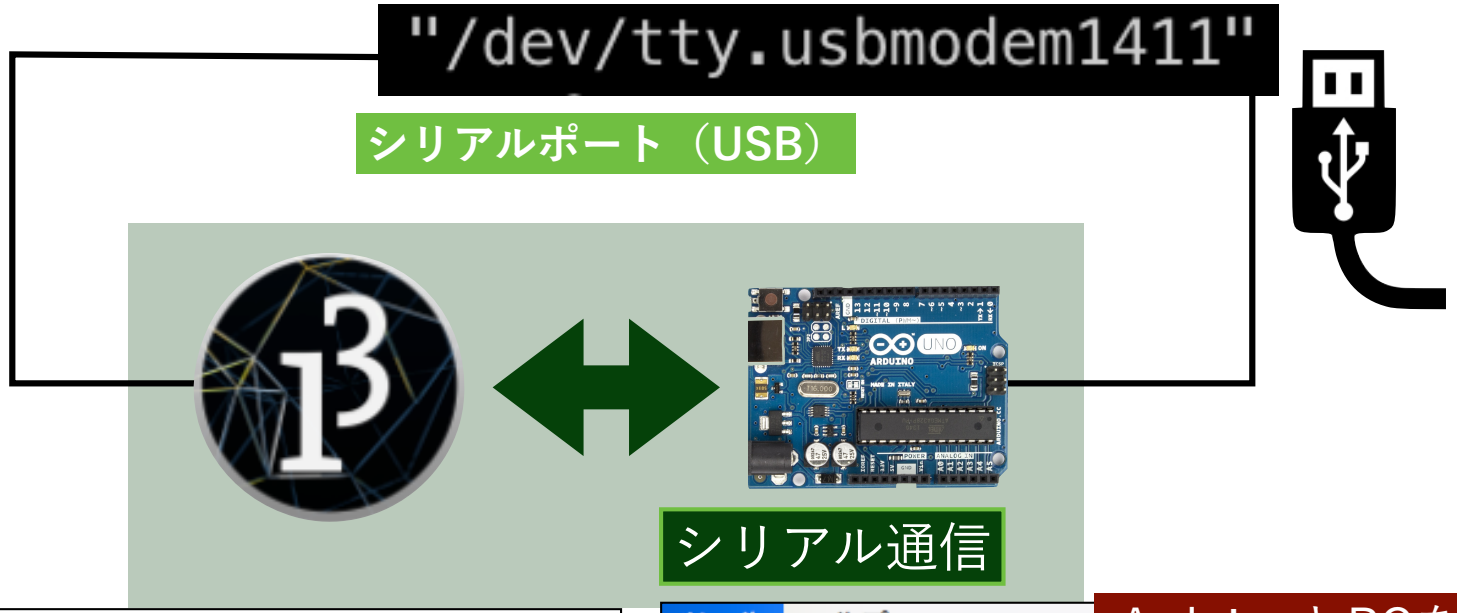
アプリケーション間通信の全体像

**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．**



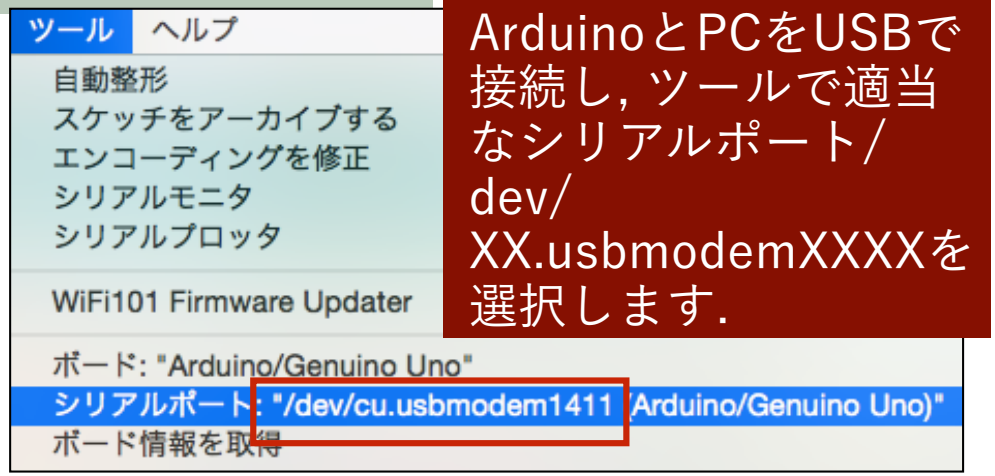
シリアル通信

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ポートを探すことができます.

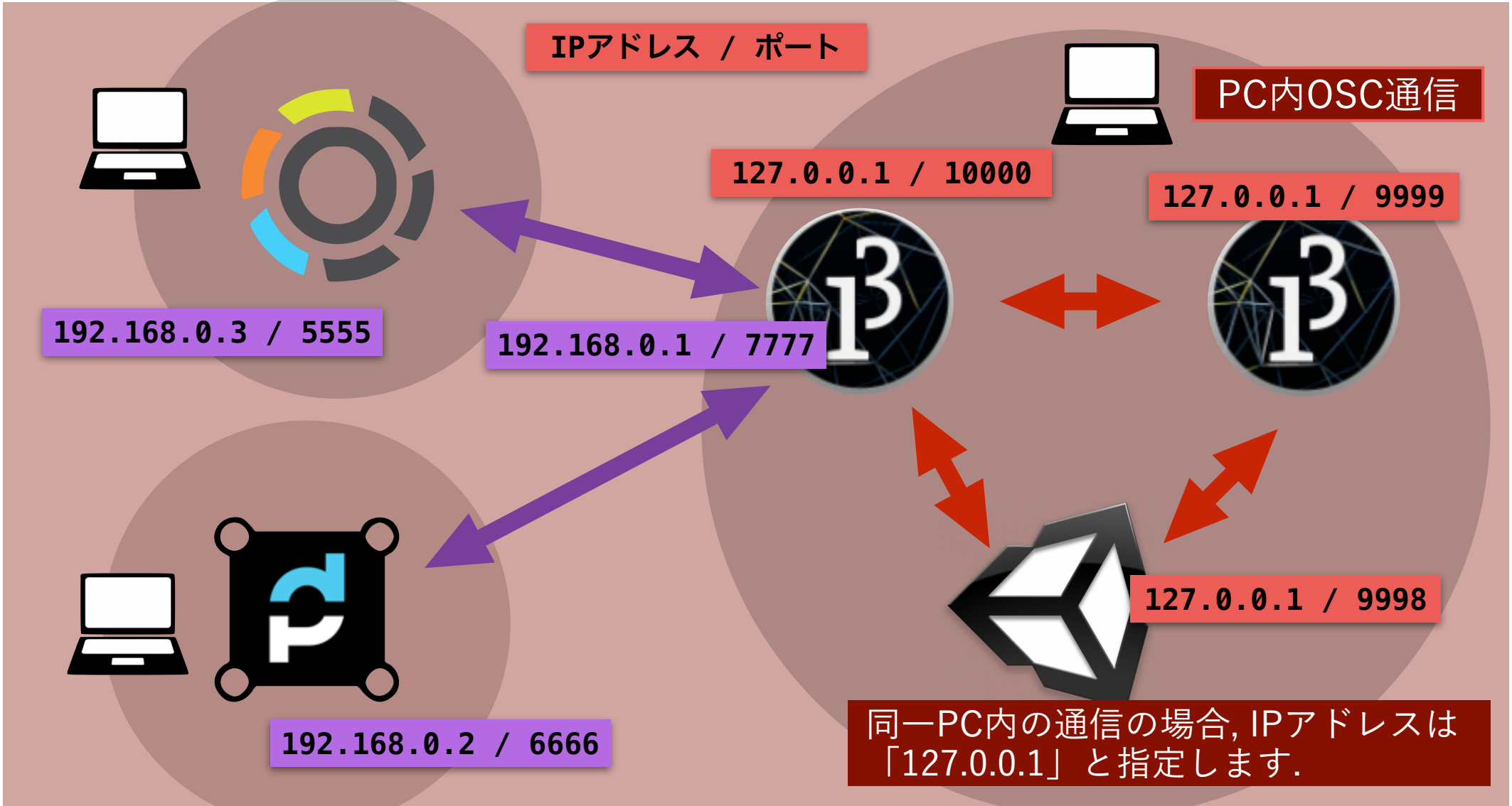


ArduinoとPCをUSBで接続し, ツールで適当なシリアルポート/dev/XX.usbmodemXXXXを選択します.

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）**

OSC Address	OSC Arguments
/mouse	x(int), y(int)
/mousey	y(int)
/message	info(string)
/volume	amplitude(float)

/mouse 34 25
/mouse 34 28
/mousey 40

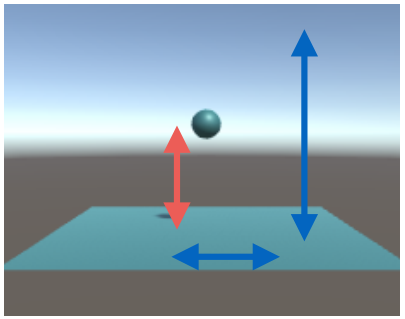
/message “hello”
/mouse “good-bye”

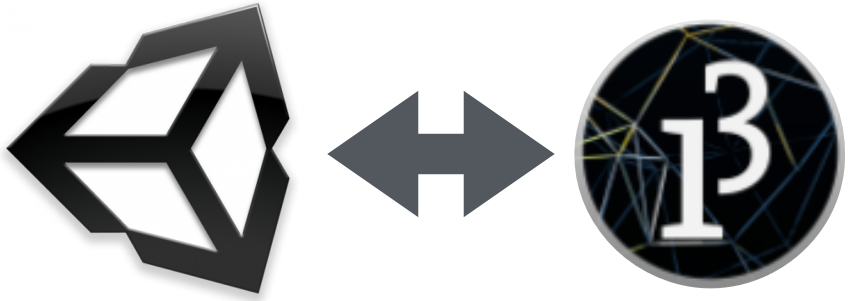
/volume 2.453
/volume 1.537
/volume 0.003

実装練習（UNITY 対 Processing の通信）

127.0.0.1 / 6666

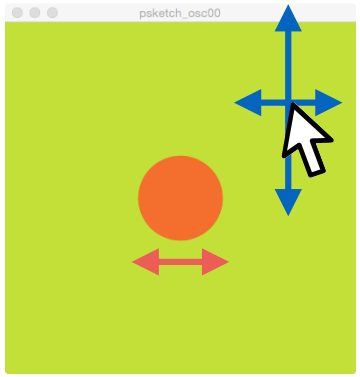
SERVER

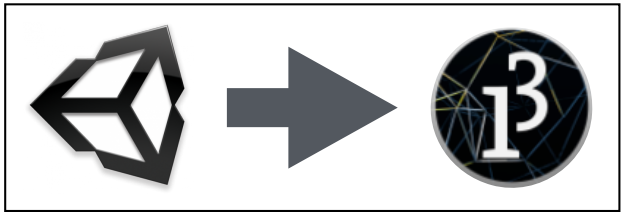




127.0.0.1 / 5555

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.

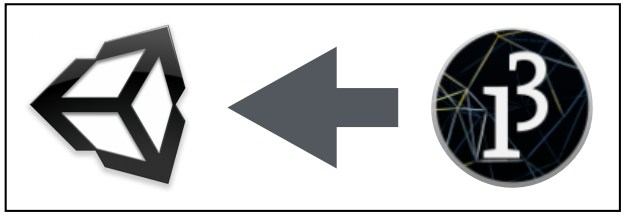
衝突回数（int）

衝突に合わせて, 背景色を変化.

/mousex, /mousey

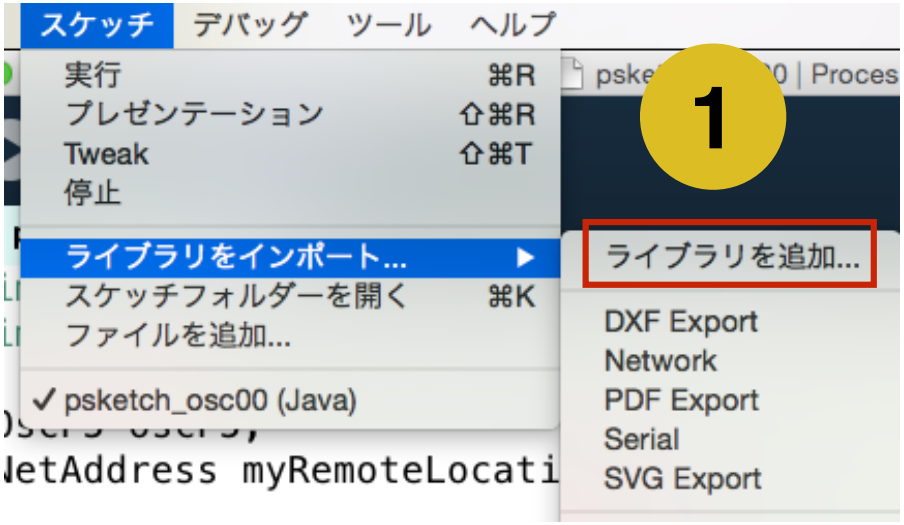
X座標（int）, Y座標（int）

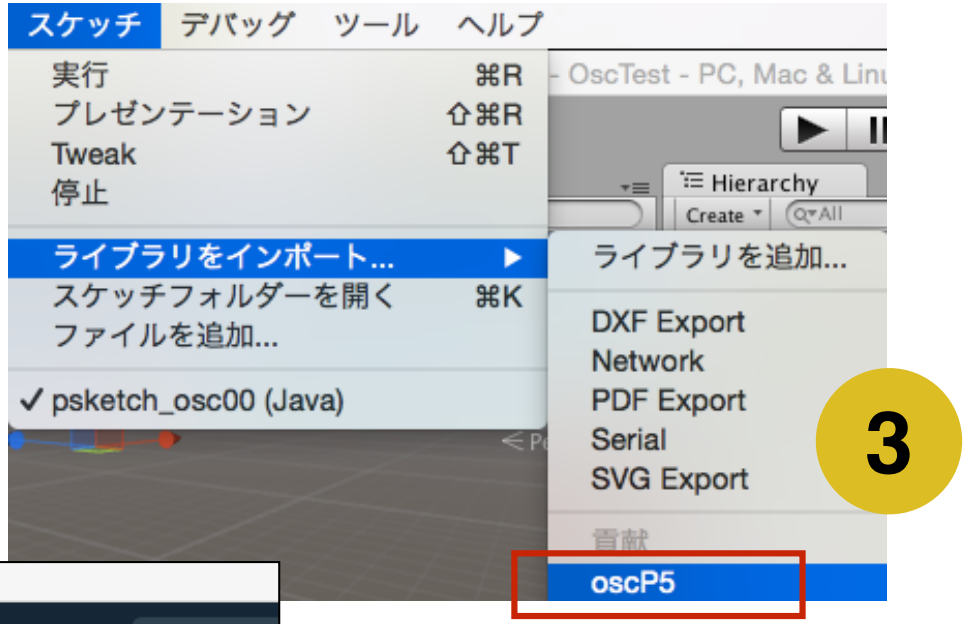
マウスのXY座標を, 床の高さと連動させます.

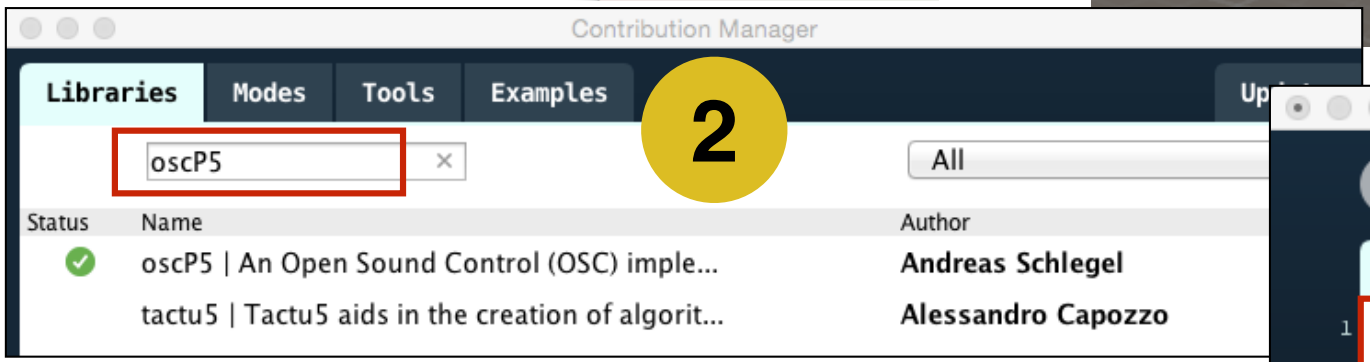


Processing側の準備

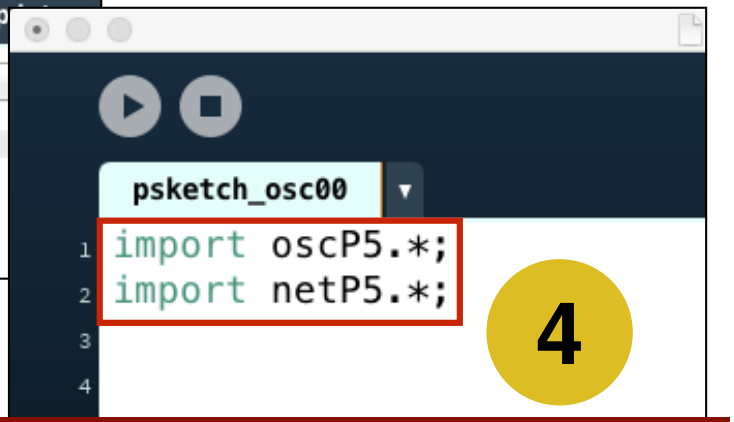
ライブラリ（oscP5）の追加







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．



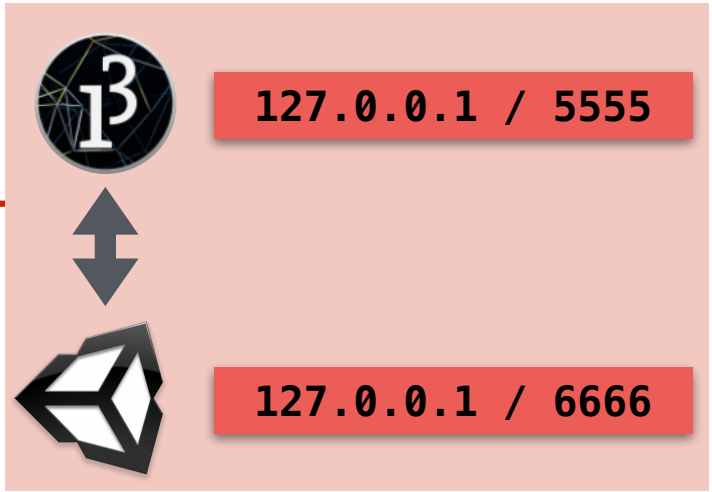
インストール後，Contribution Libraryとして，oscP5が表示されるので選択すると，自動的に関連のimportがimportされます．

Processing側の準備

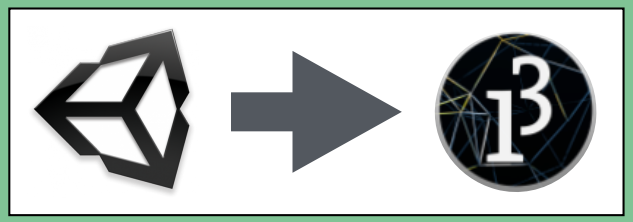
Processingのコード（1/3）

```
import netP5.*;
import oscP5.*;

OscP5 oscP5;
NetAddress myRemoteLocation;

float y_floor = 0; float y_ball = 0; //床とボールのY座標
float dif = 0; //ボールと床の距離
color col;      //背景色（衝突のたびにランダムに変わる）

void setup(){
  size(400,400);  frameRate(10);  background(0);

  oscP5 = new OscP5(this,5555);
  myRemoteLocation = new NetAddress("127.0.0.1",6666);

  oscP5.plug(this,"getFloorPos","/pos/floor");
  oscP5.plug(this,"getBallPos","/pos/ball");
  oscP5.plug(this,"getHit","/hit");
 }
```

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, 背景色を変化.

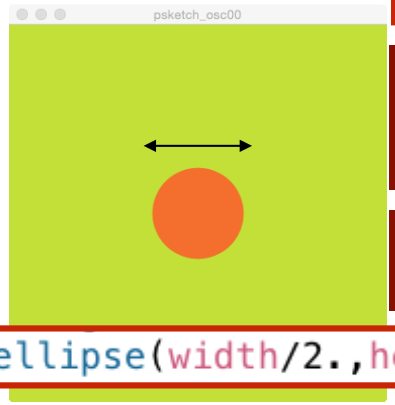
Processing側の準備

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．

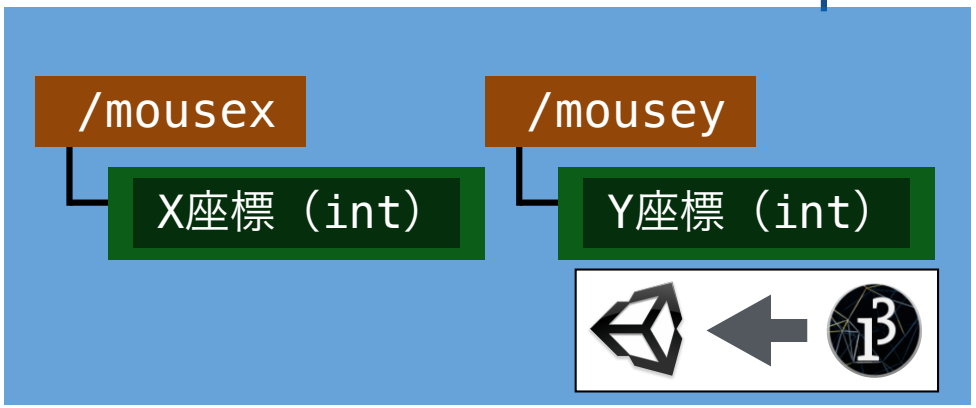
```
background(col);
```



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);
```



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。

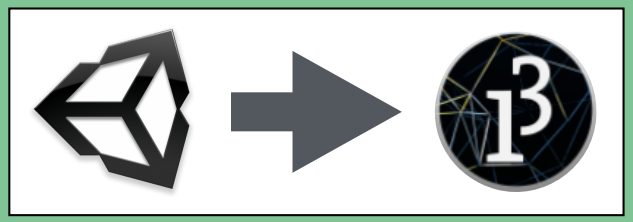
```
17 oscP5.plug(this,"getFloorPos","/pos/floor");
18 oscP5.plug(this,"getBallPos","/pos/ball");
19 oscP5.plug(this,"getHit","/hit");
```

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

/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

/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

/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
  - 衝突回数（int）

球と床の距離を円の直径と連動させます.

衝突に合わせて, 背景色を変化.

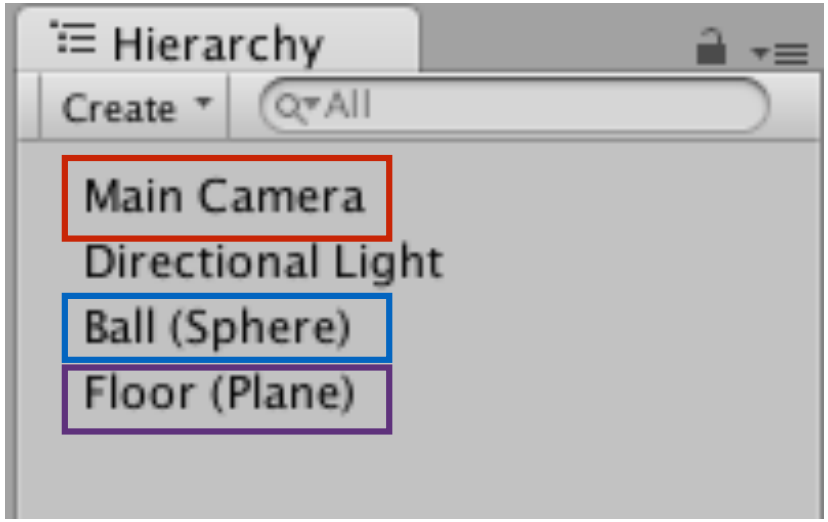
Processing側の準備

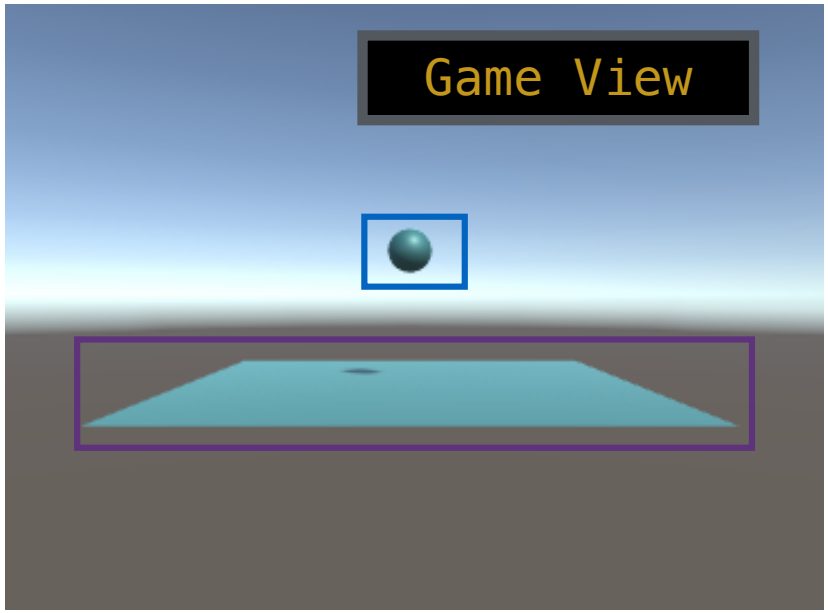
Processingのコード（全体）

```
import netP5.*;
import oscP5.*;

OscP5 oscP5;
NetAddress myRemoteLocation;

float y_floor = 0; float y_ball = 0; //床とボールのY座標
float dif = 0; //ボールと床の距離
color col;     //背景色（衝突のたびにランダムに変わる）

void setup(){
  size(400,400);  frameRate(10);  background(0);

  oscP5 = new OscP5(this,5555);
  myRemoteLocation = new NetAddress("127.0.0.1",6666);

  oscP5.plug(this,"getFloorPos","/pos/floor");
  oscP5.plug(this,"getBallPos","/pos/ball");
  oscP5.plug(this,"getHit","/hit");
 }

void draw(){

  dif = y_ball - y_floor; //床とボールの距離を計算
  background(col);  noStroke();  fill(255,100,0);
  ellipse(width/2.,height/2.,50.*dif,50.*dif);
  if(mousePressed){
    OscMessage myMessage_x = new OscMessage("/mousex");
    myMessage_x.add(mouseX);
    oscP5.send(myMessage_x, myRemoteLocation);

    OscMessage myMessage_y = new OscMessage("/mousey");
    myMessage_y.add(mouseY);
    oscP5.send(myMessage_y, myRemoteLocation);
  }
}

void getFloorPos(float x, float y, float z){
    y_floor = y;
}

void getBallPos(float x, float y, float z){
    y_ball = y;
}

void getHit(int count){
  println(count);
  col = color(random(255),random(255),random(255));
}
```

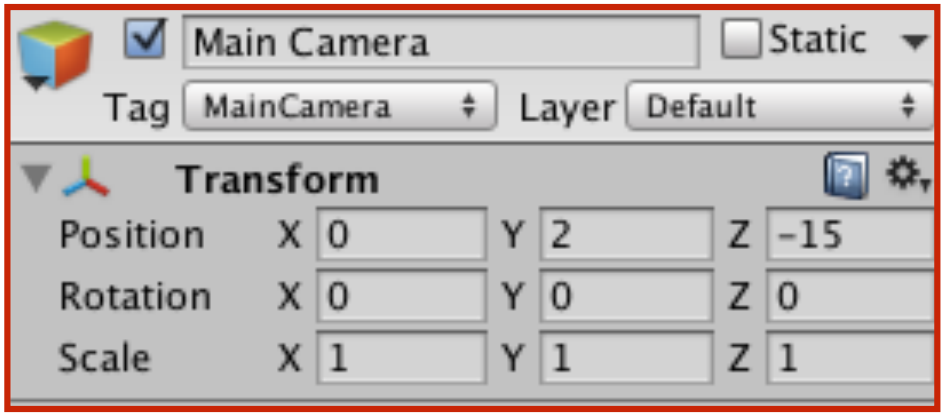
UNITY側の準備

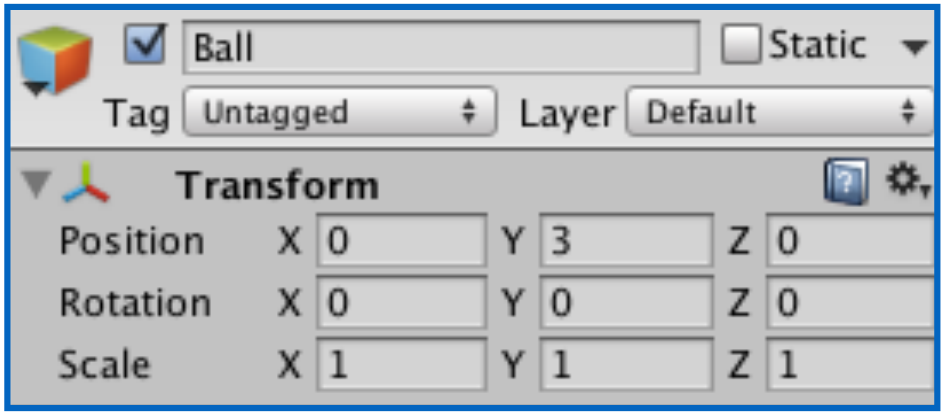
基本環境の構築（ゲームオブジェクトの追加）

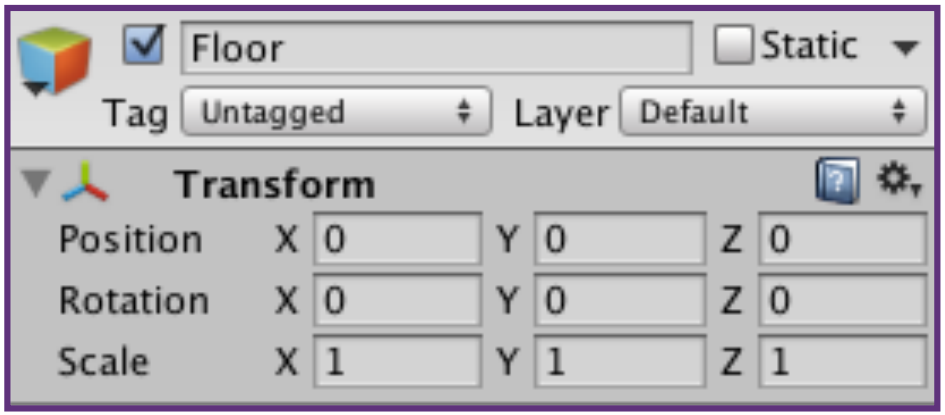




必要に応じて, Materialを追加して色をつけてください.

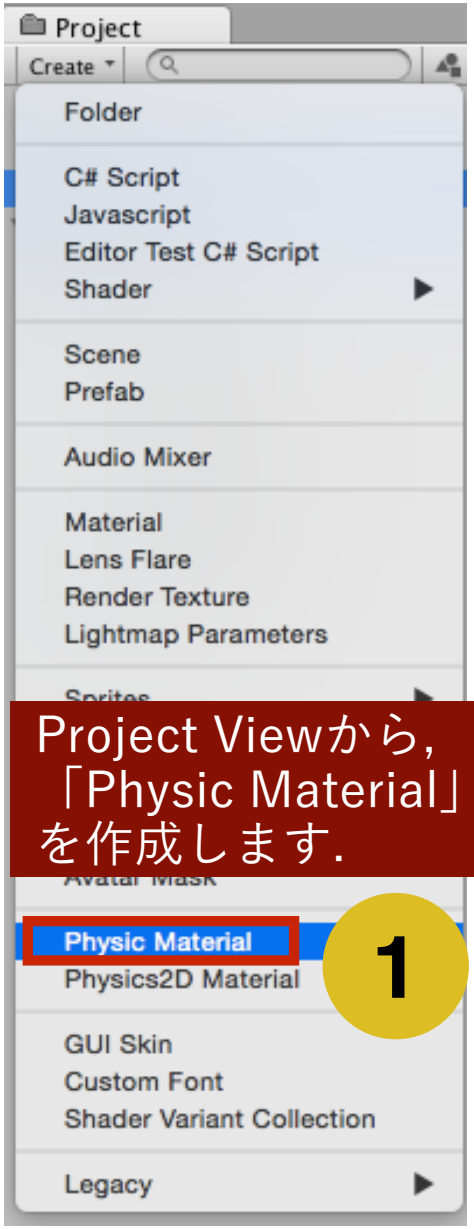






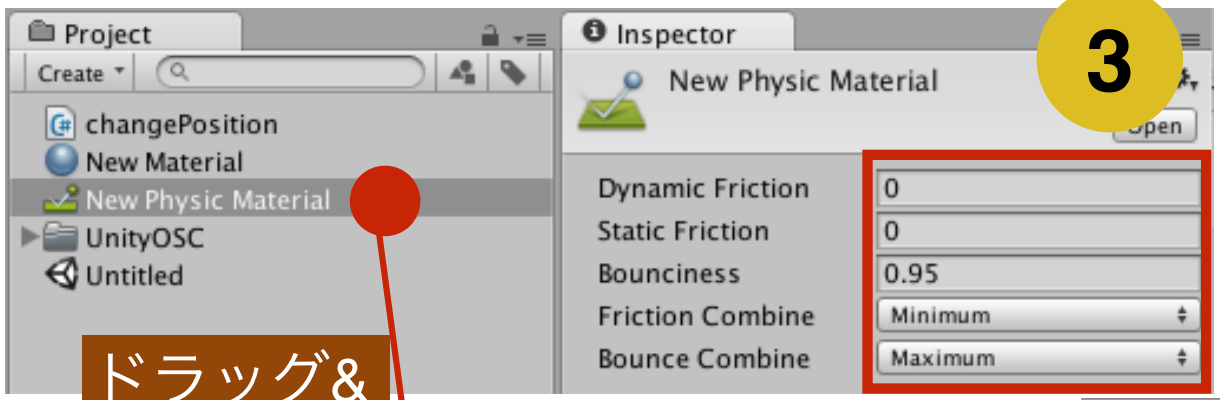
UNITY側の準備

基本環境の構築（ボールがバウンドするようにする）



Project Viewから，「Physic Material」を作成します．

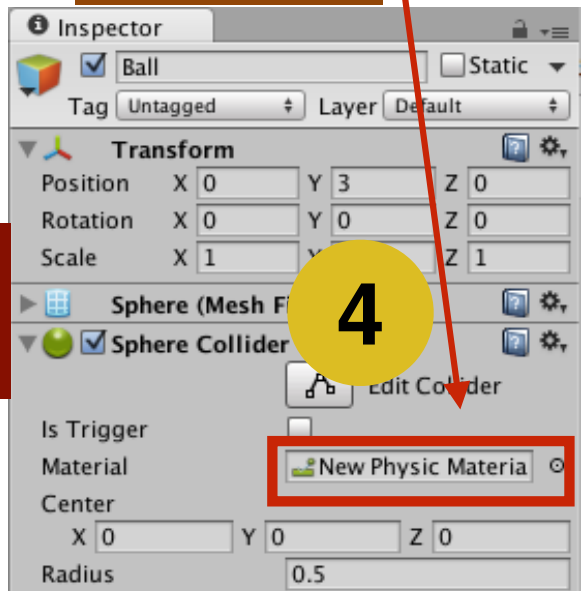
1



3

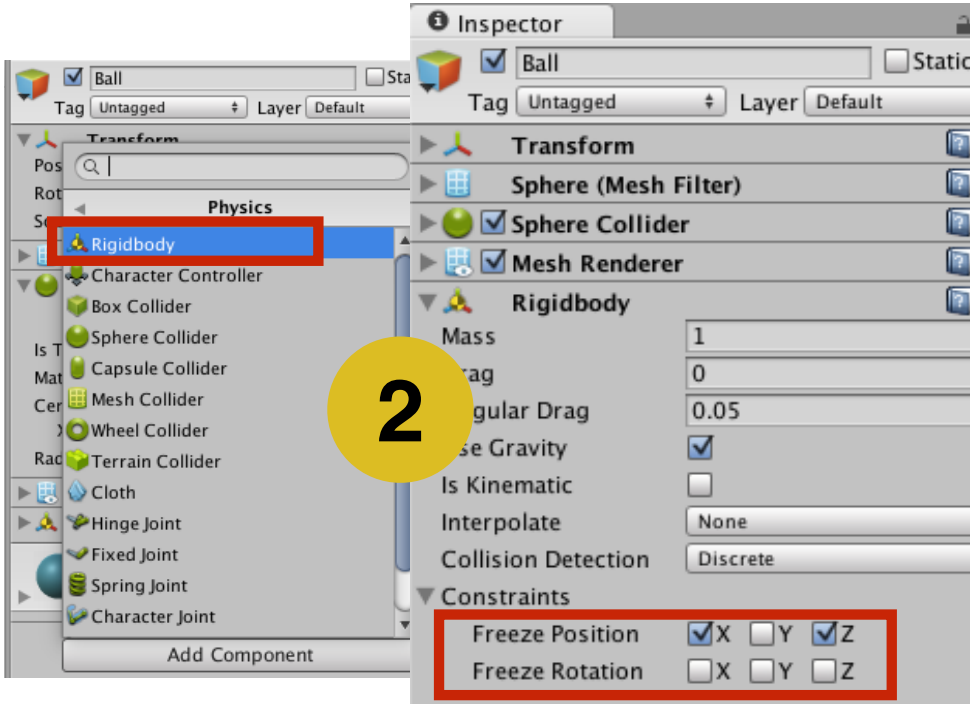
Bouncinessは反発係数ですので，これを増やすと，ボールが床から跳ね返るようになります．

ドラッグ&ドロップ



4

作成した物理特性をBallと関連付けます．



2

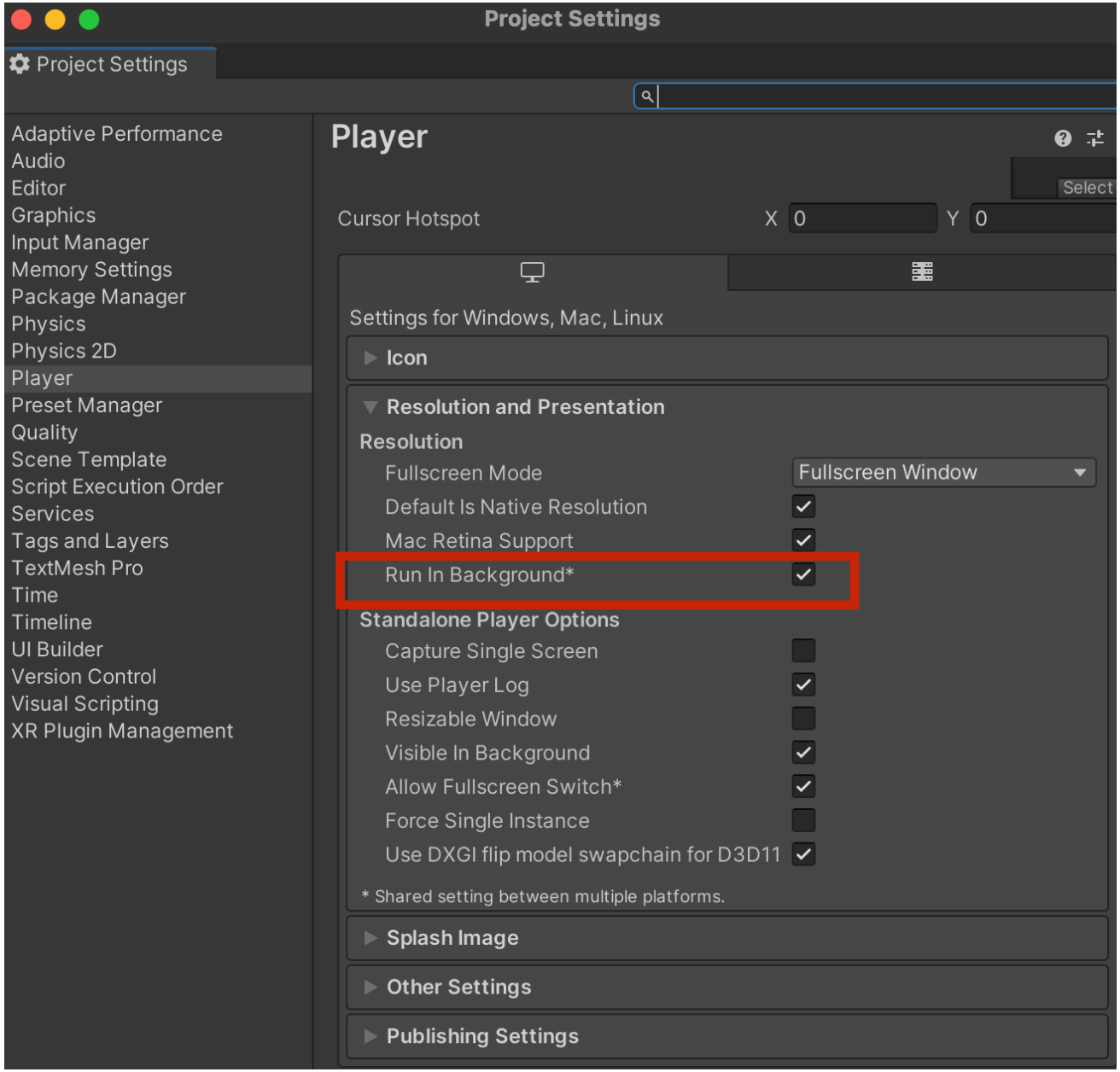
コンポーネントとしてRigidbodyを追加し，Y座標（上下）のみを動くようにします．

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）

Edit→Project SettingsからPlayerを選択し、Run in Background*にチェックをいれる。

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.



UNITY側の準備

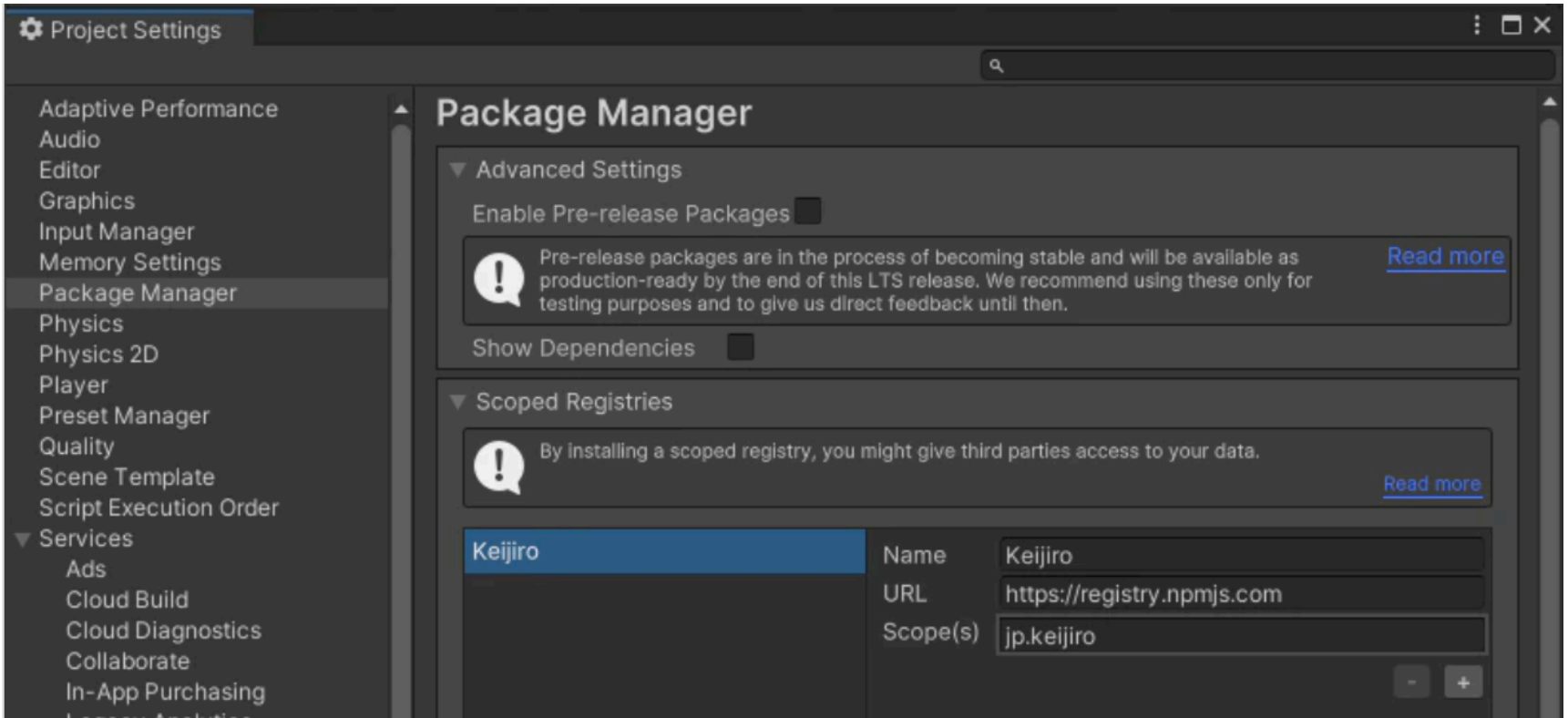
OscJackのインポート（1/2）

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

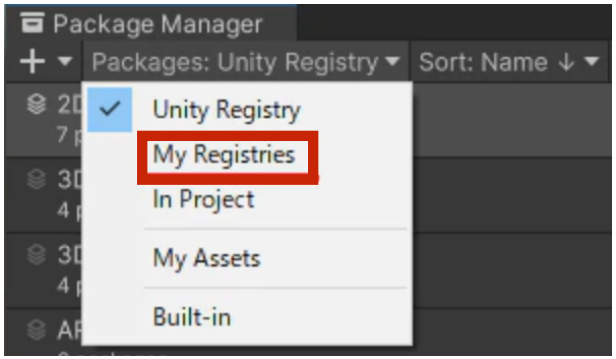
Scope(s) : jp.keijiro



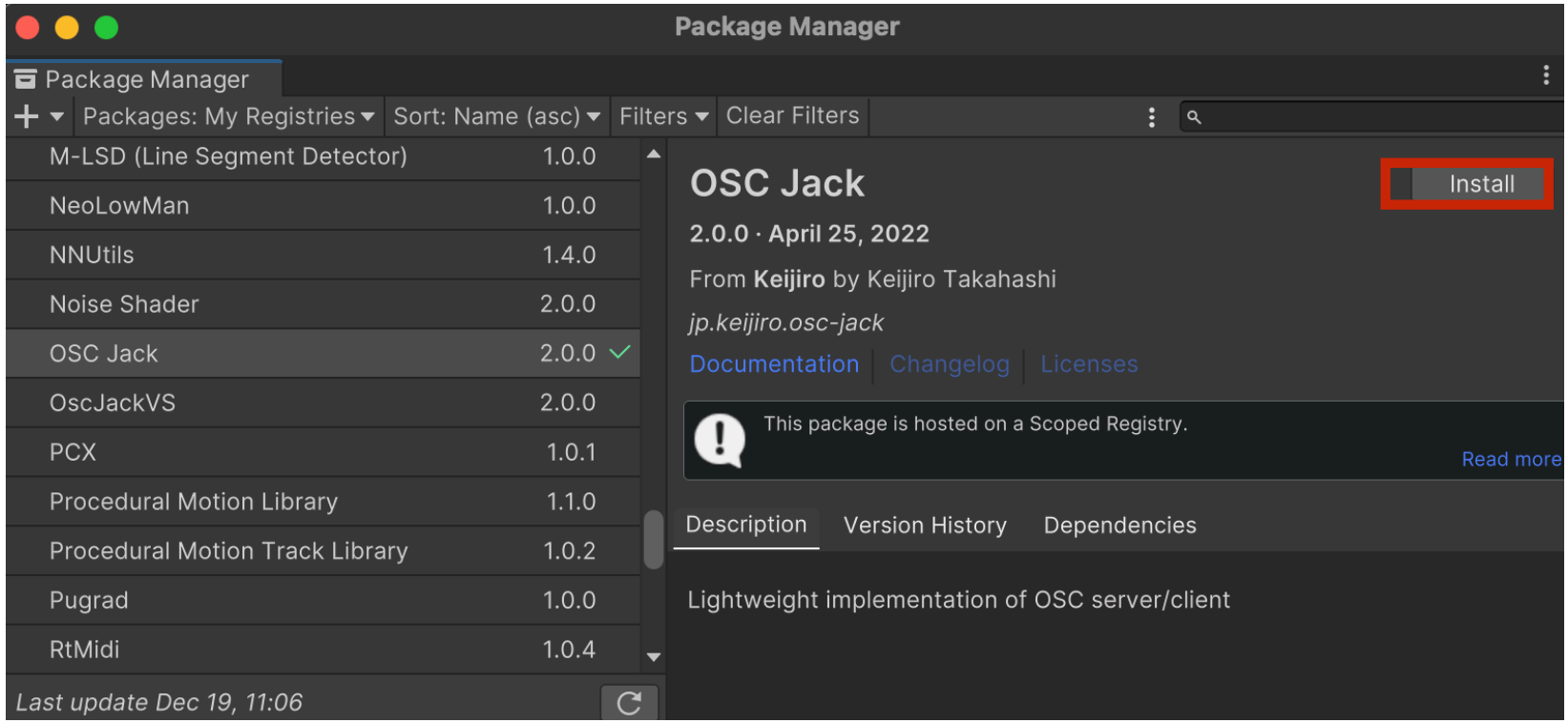
UNITY側の準備

OscJackのインポート（2/2）[Unity6 以前]

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.



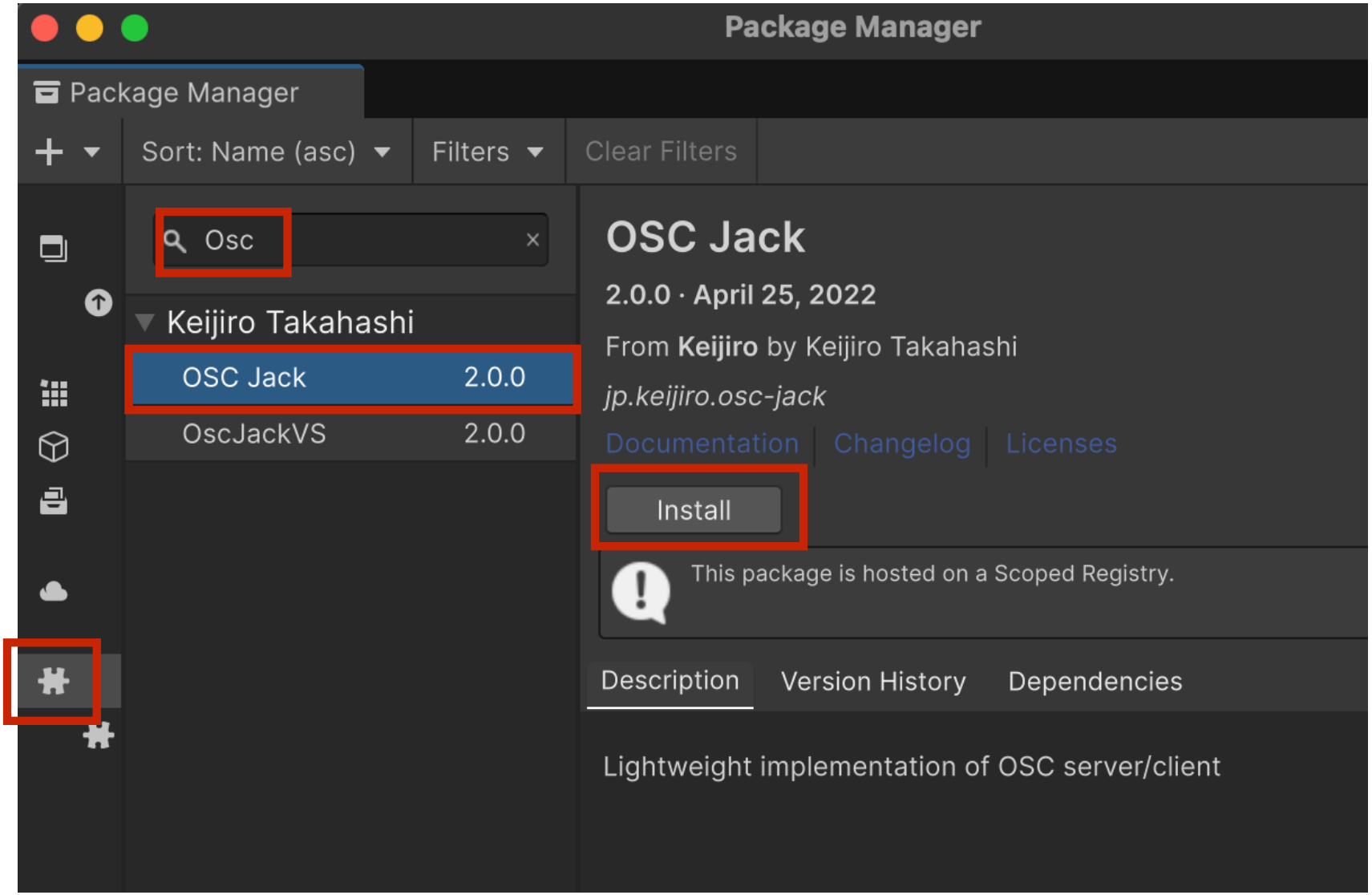
リストから OSCJackを選択し、Installボタンを押す. これでインポートは完了.



UNITY側の準備

OscJackのインポート（2/2）[Unity6の場合]

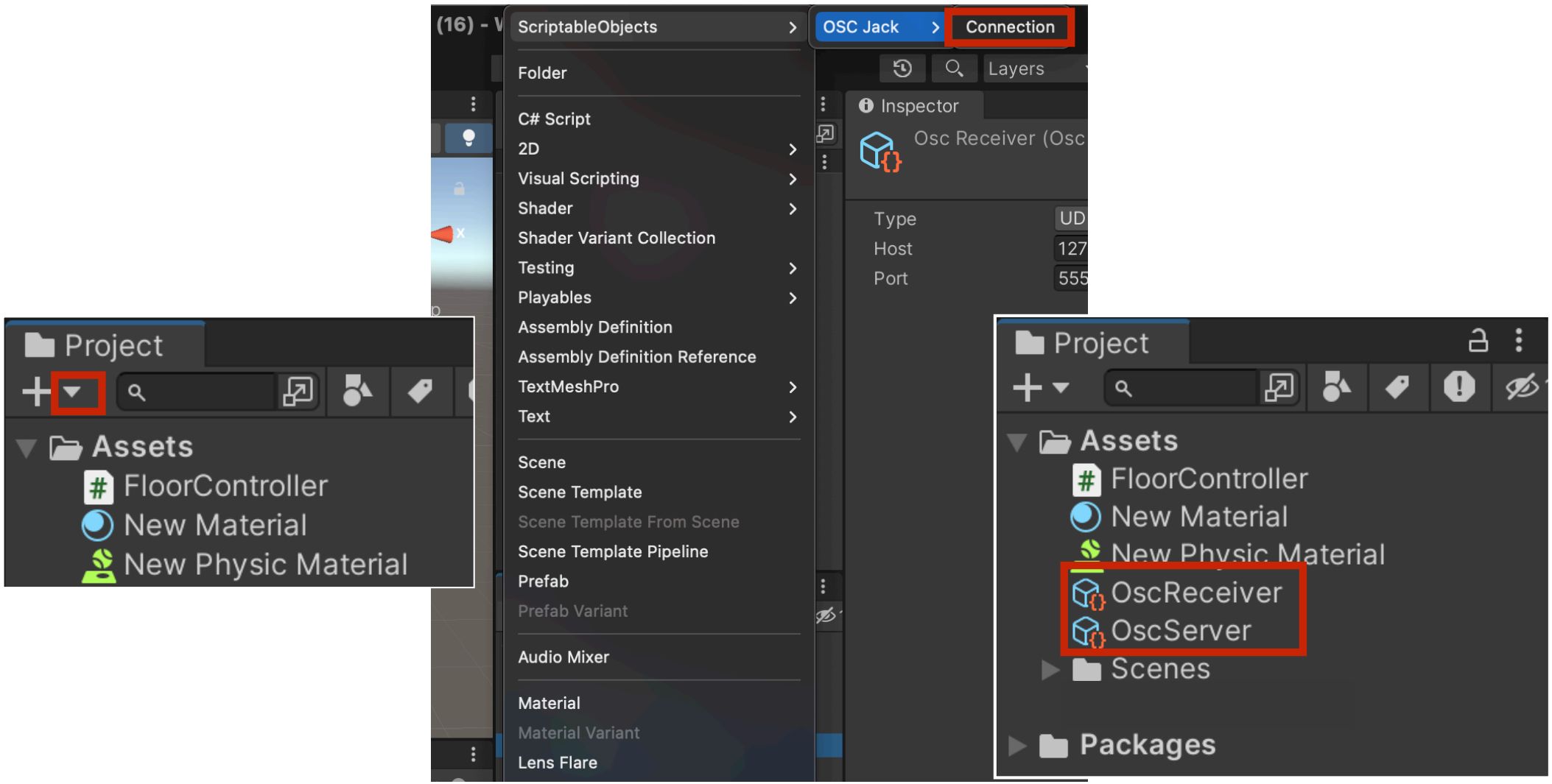
Window→Package Managerを開き、
ウィンドウ左端のバーからパズルマークを選び、OSC Jackをインストール



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

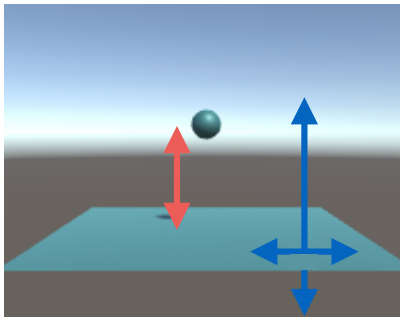
ポート情報を管理するConnectionの設定

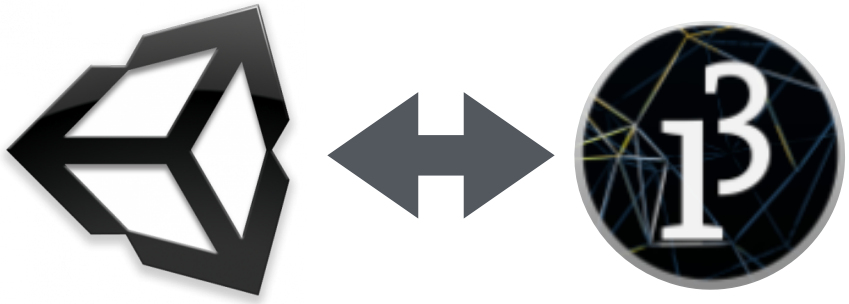
127.0.0.1 / 6666

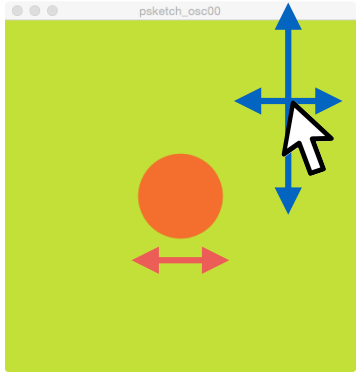
127.0.0.1 / 5555

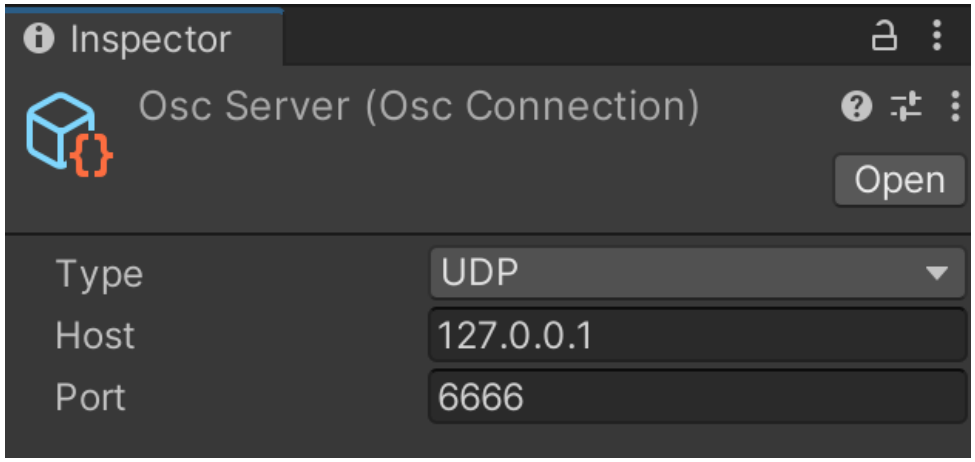
SERVER

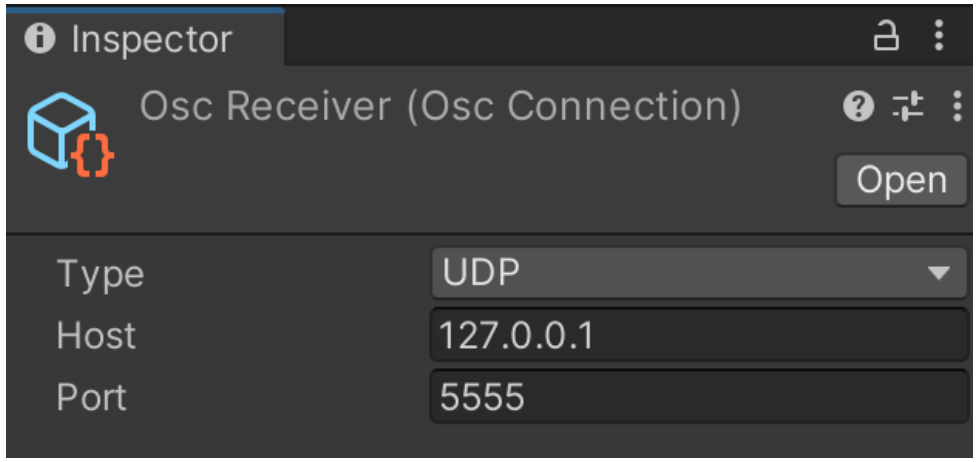
Client











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）。

UNITY側の準備

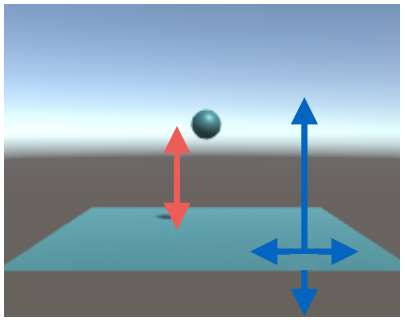
受信編

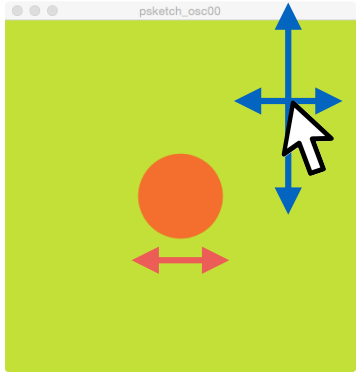
127.0.0.1 / 6666

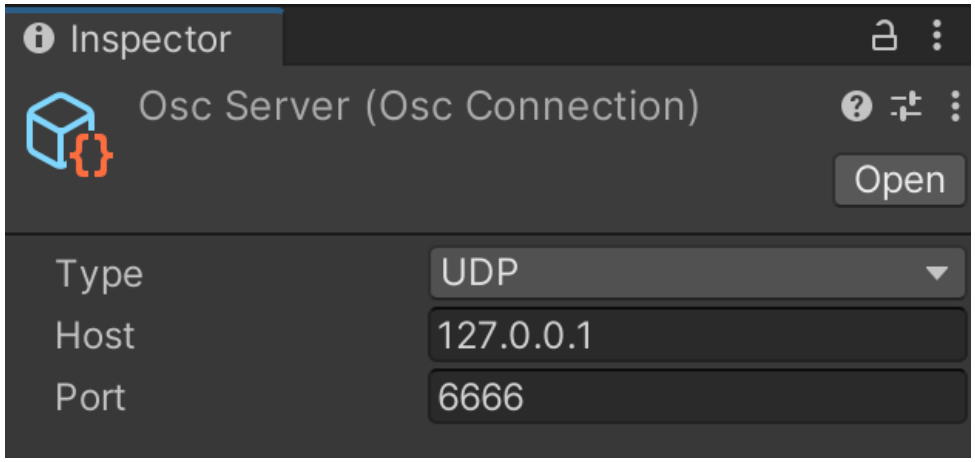
127.0.0.1 / 5555

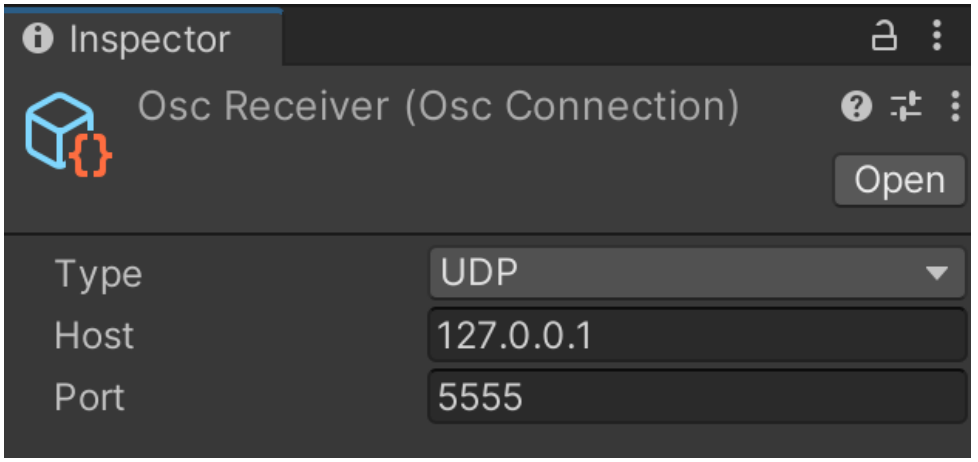
SERVER

Client





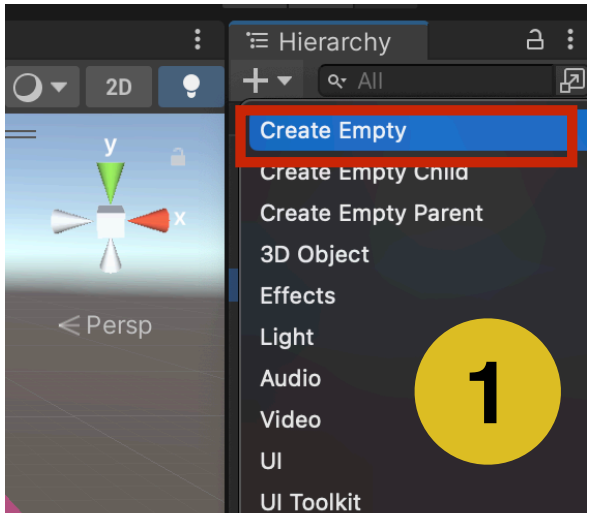




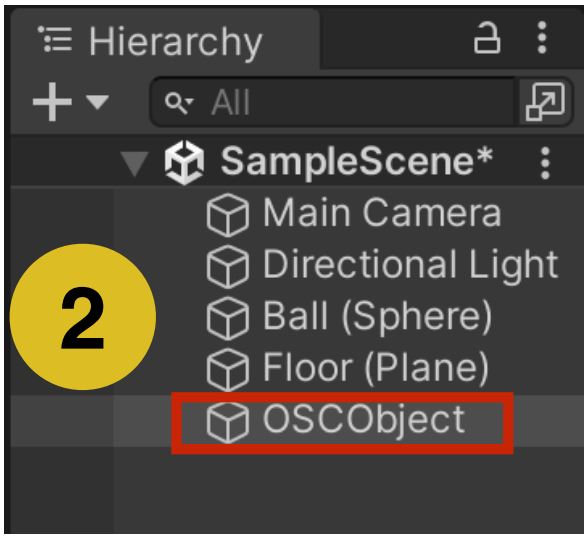
UNITY側の準備

受信編

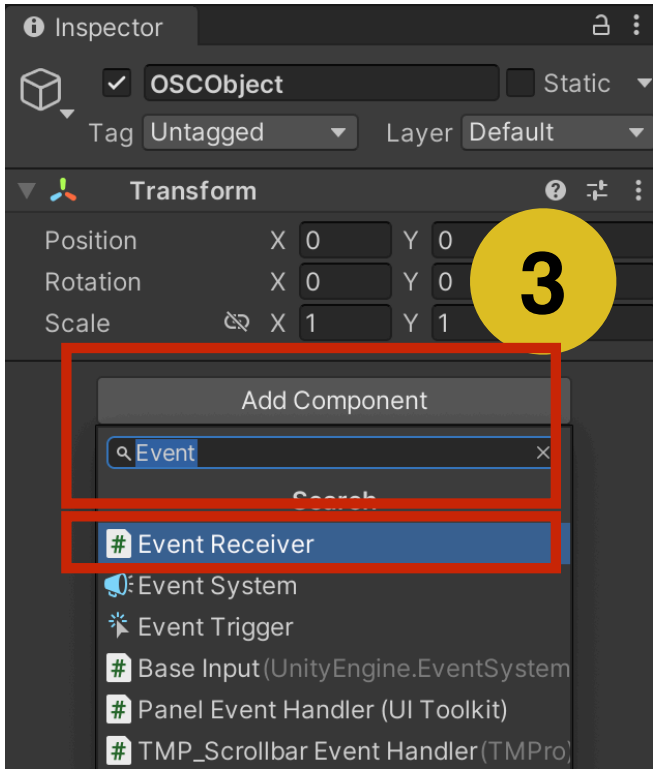
EventReceiverの設定

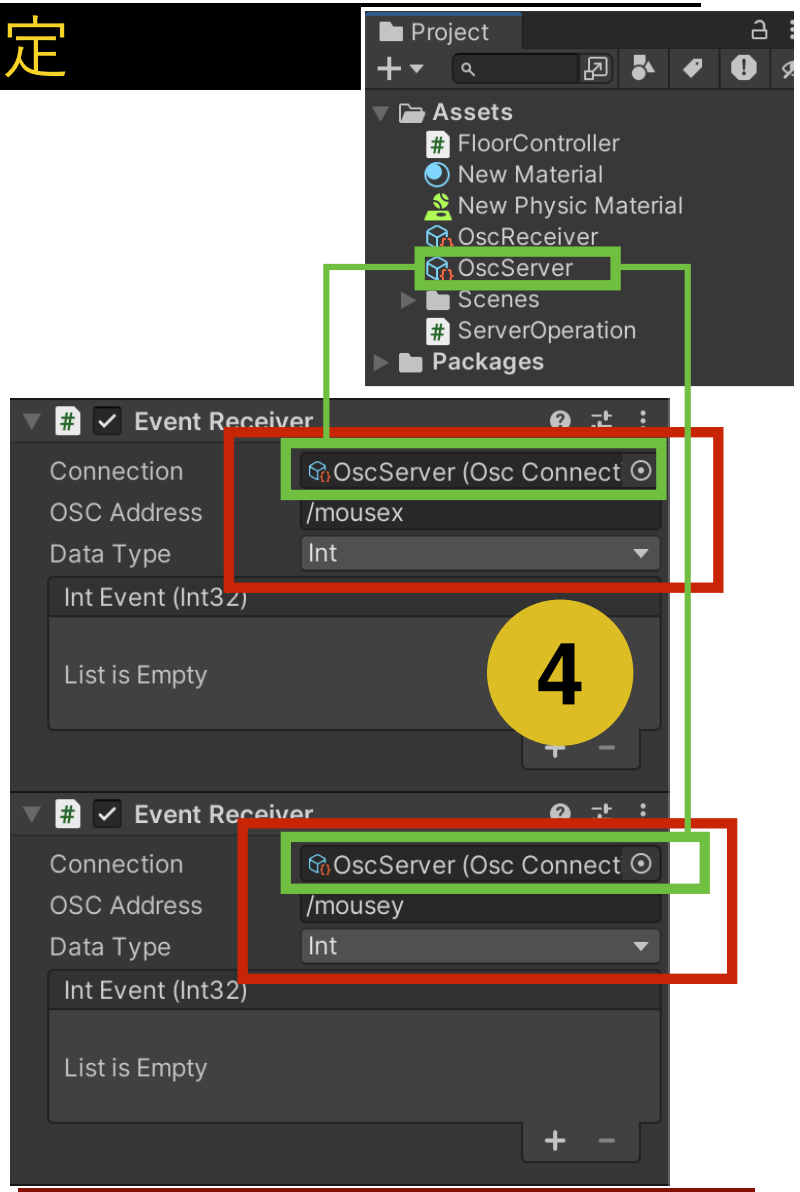


Add Componentより「EventReceiver」を2つ追加.します。受信するOSC情報を管理するためのコンポーネントです。



空のゲームオブジェクトを作成し、OSCObjectとリネームする。





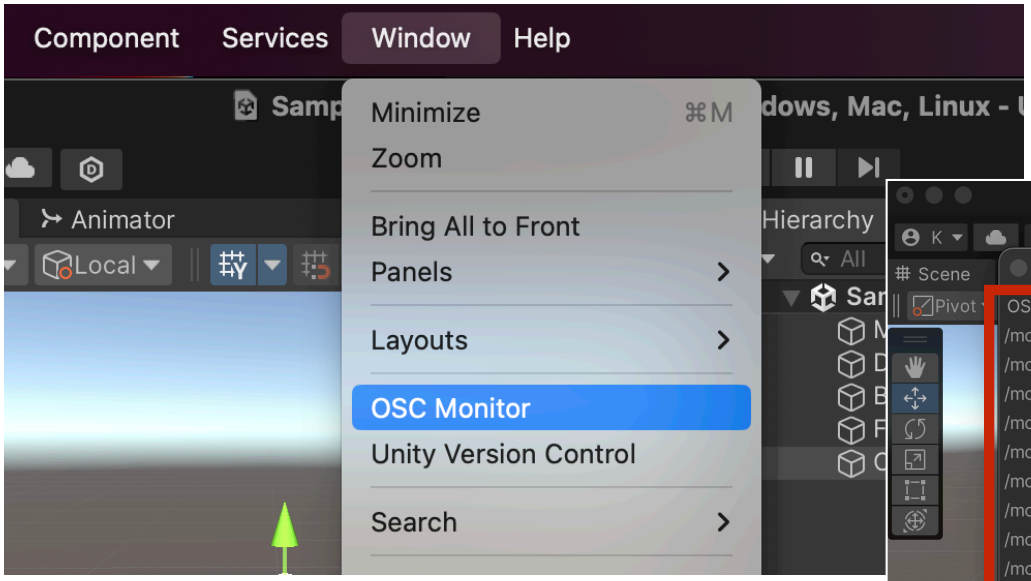
Connectionにすでに作成した「OscServer」をアタッチし、OSCAddressに/mousex、/mouseyを、Data Typeをそれぞれ「Int」に設定しておきます。

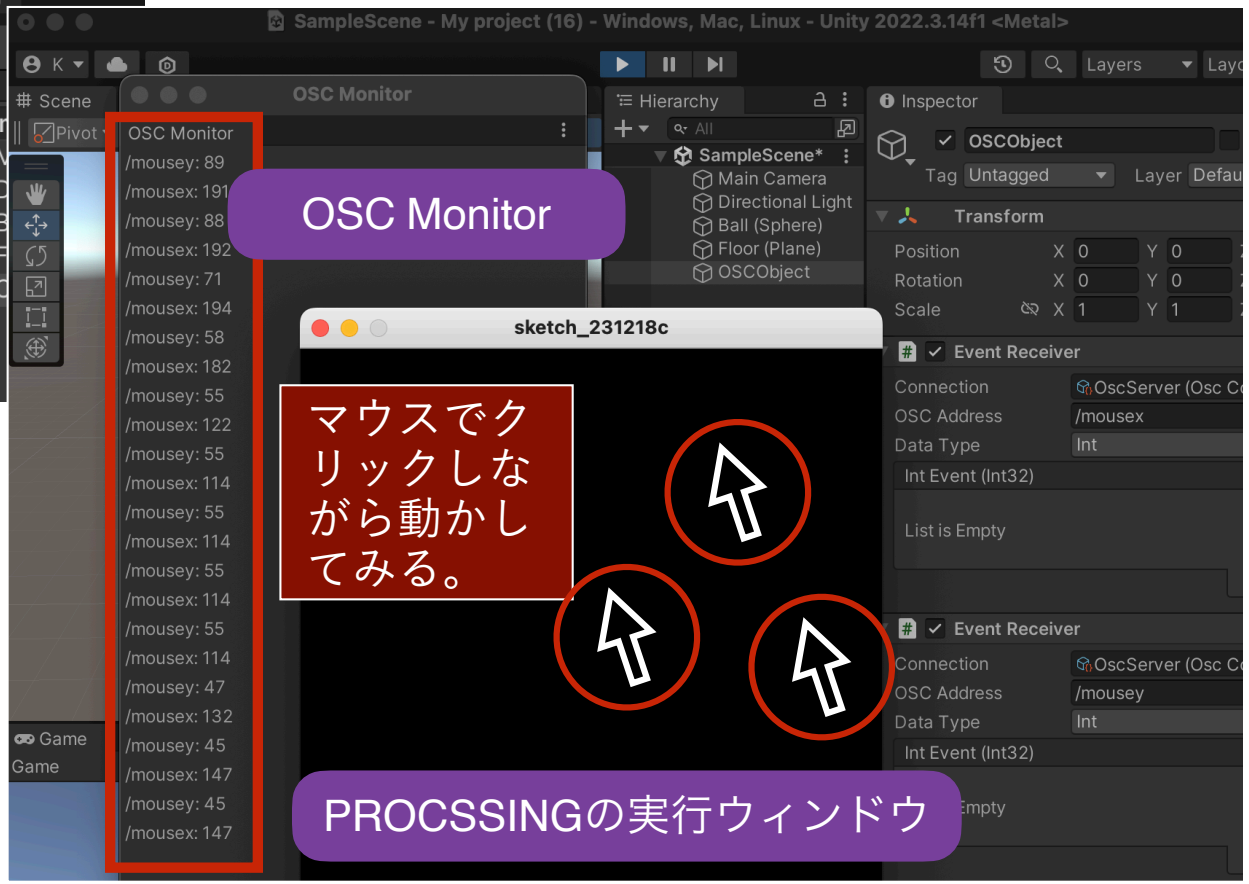
UNITY側の準備

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、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。



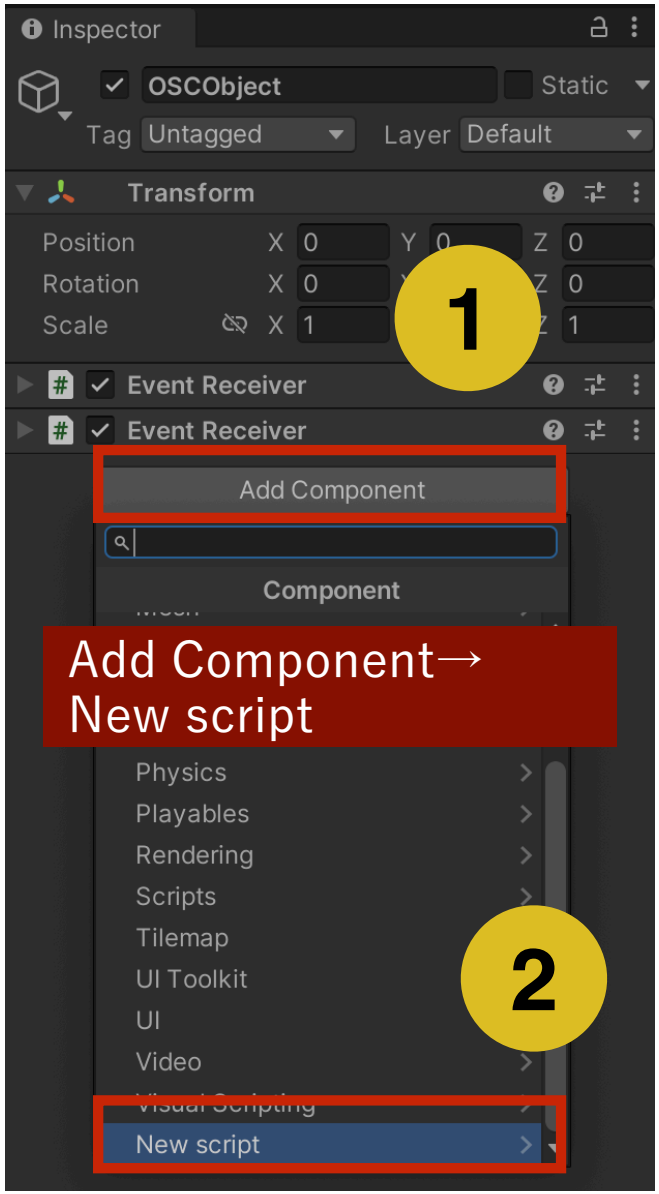


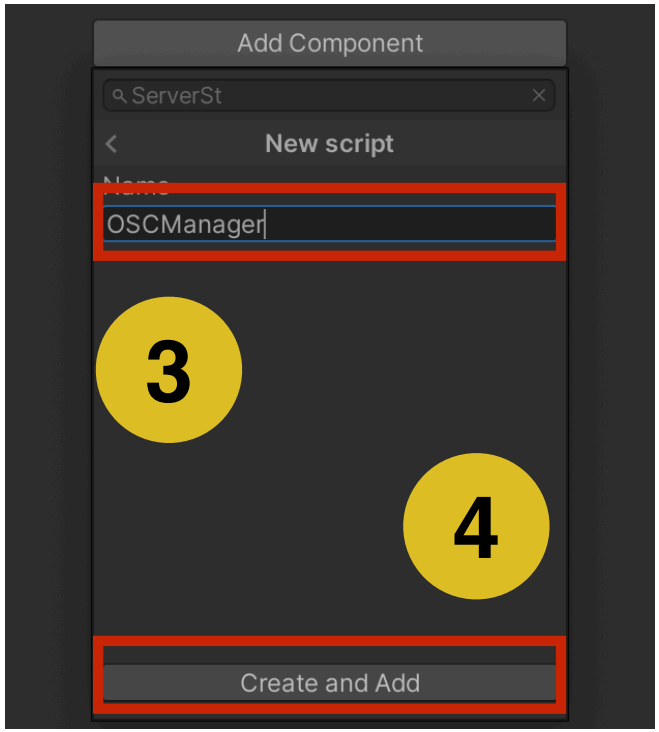
UNITY側の準備

受信編

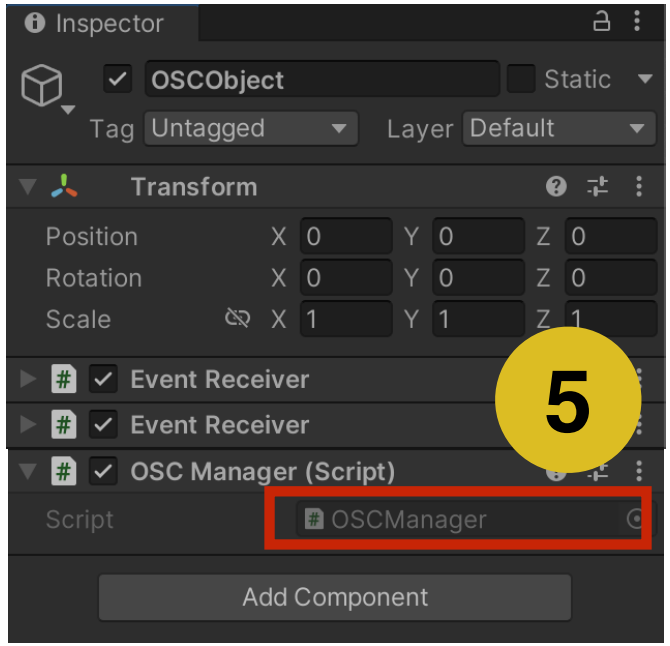
コールバック関数の作成と関連付け（1/3）

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

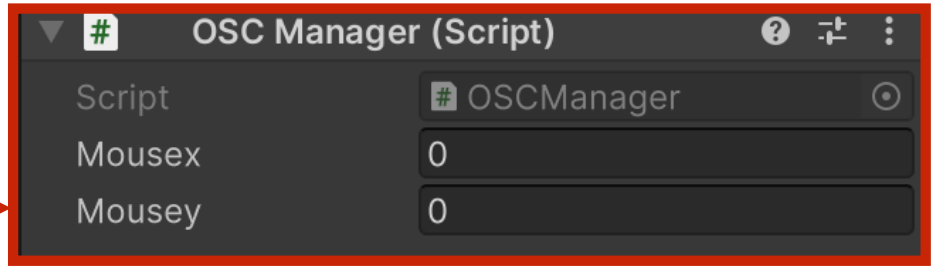
コールバック関数の作成と関連付け（2/3）

デフォルトのstart関数とupdate関数は消去してください。

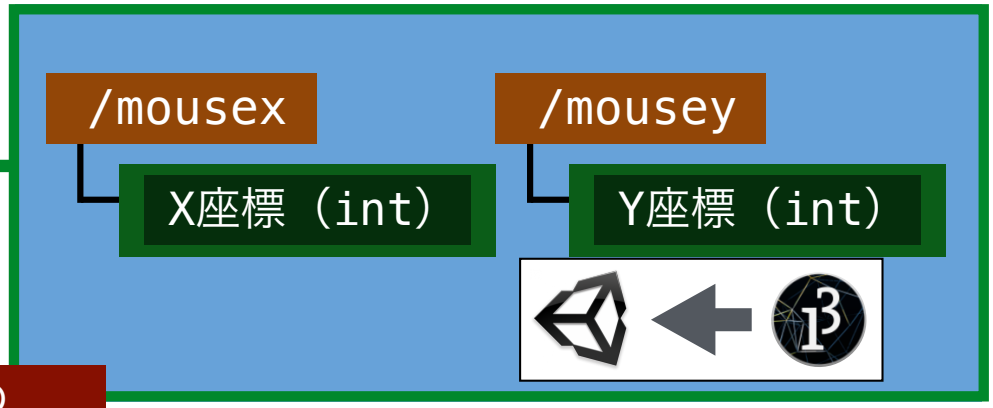
OSCManager.cs

Assets > 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。



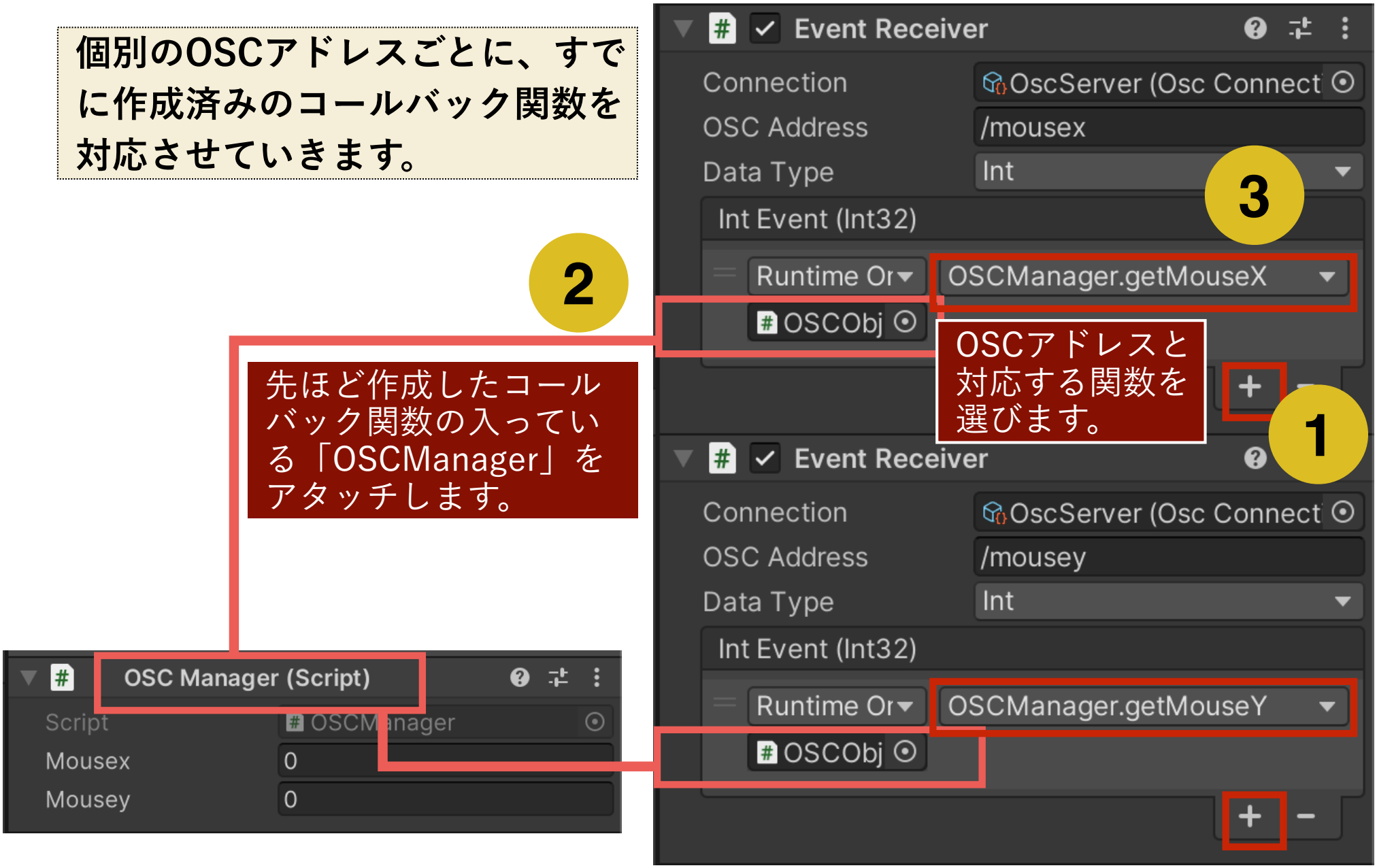
/mousex、/mouseyを受け取るためのコールアップ関数を作成しておきます。それぞれ、引数がint型であることに注意。（おそらく複数の引数はとれない模様）

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。



UNITY側の準備

受信編

コールバック関数の動作確認

/mousex

X座標（int）

/mousey

Y座標（int）

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。

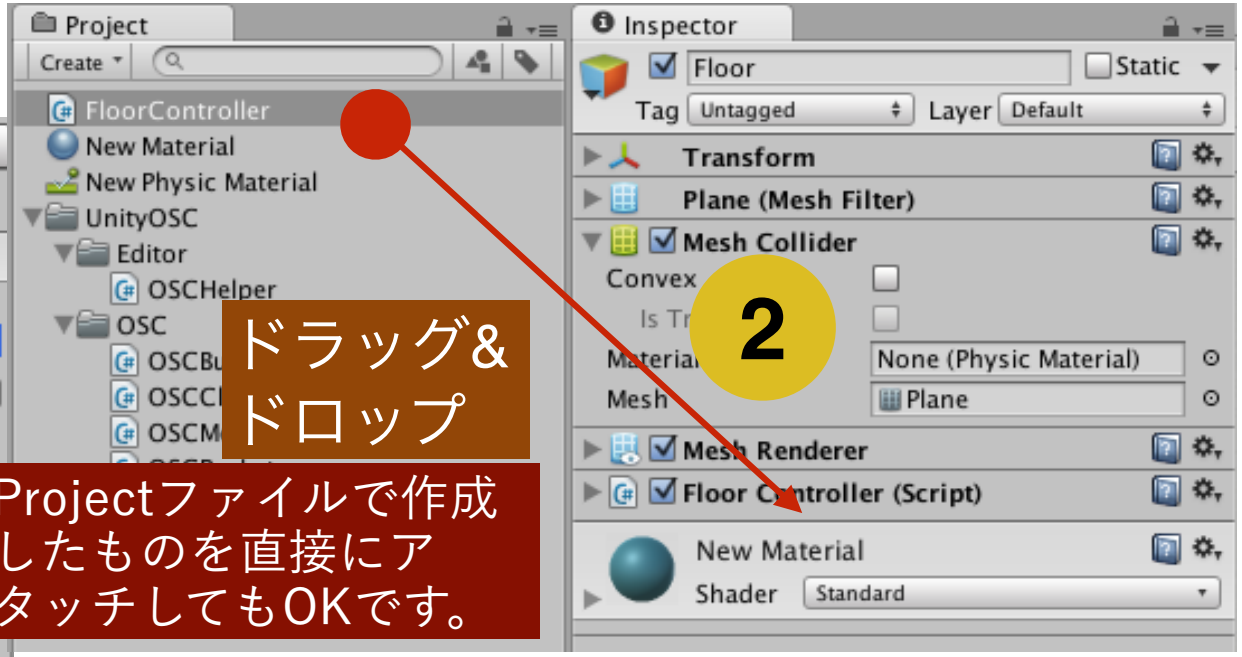
UNITY側の準備

受信編

FloorController.csコンポーネントの作成

1

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



2

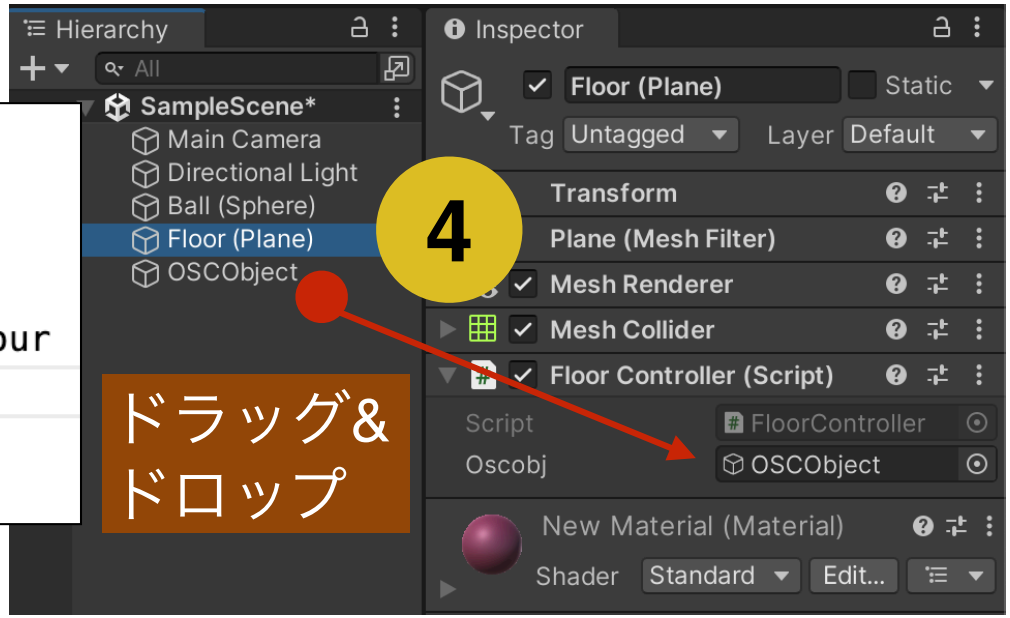
ドラッグ&ドロップ

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。

```
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

UNITY側の準備

受信編

FloorController.csを編集

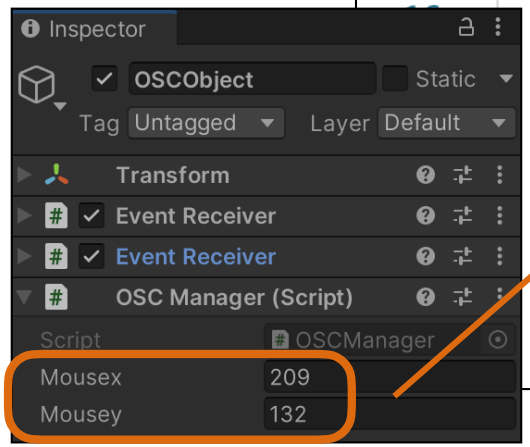
FloorController.cs ●

Assets > 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）

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。

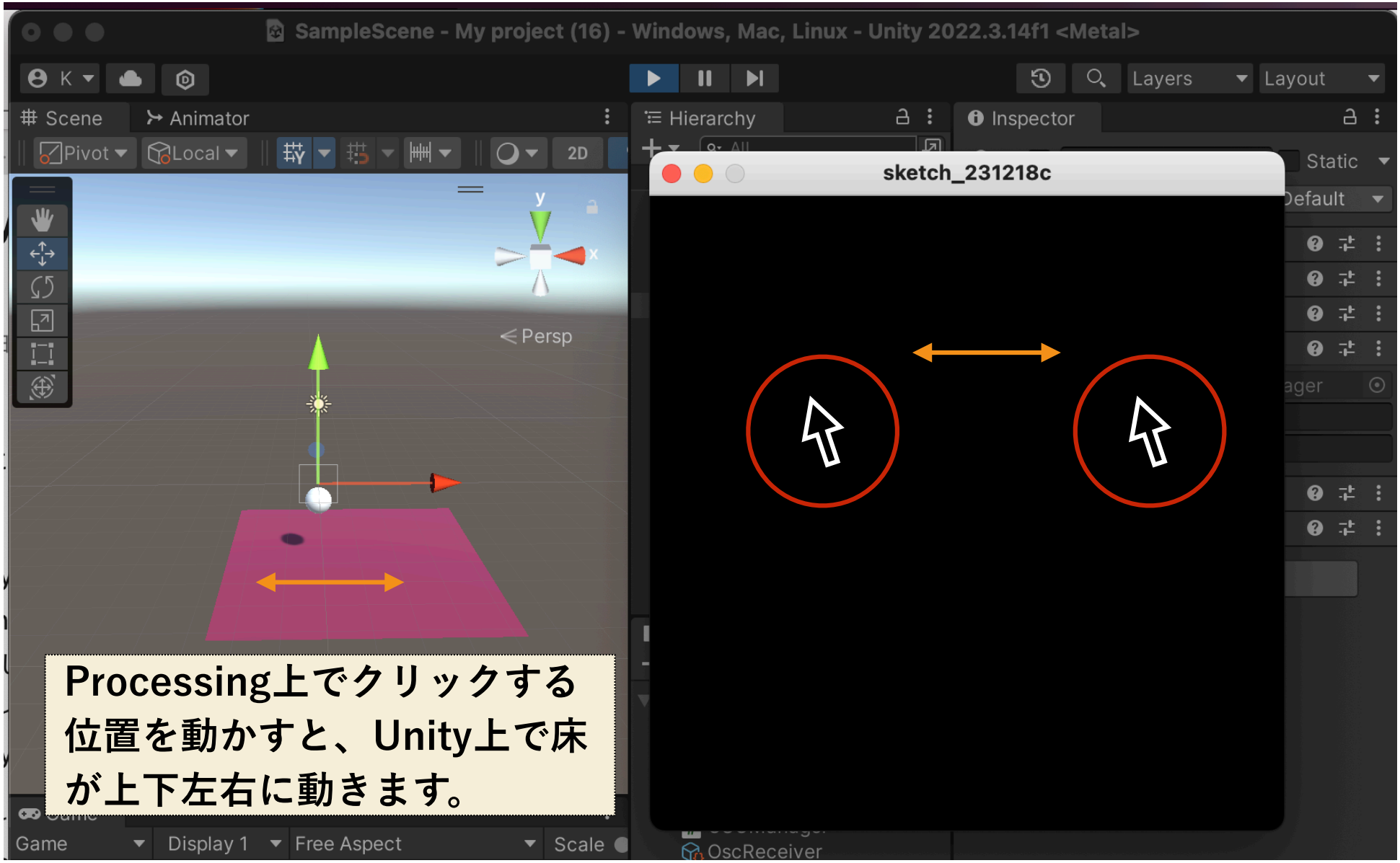
OSCObjectの2つのpubic変数にアクセスしています。

毎フレーム行われる処理を記述します.

UNITY側の準備

受信編

FloorController.csを編集



UNITY側の準備

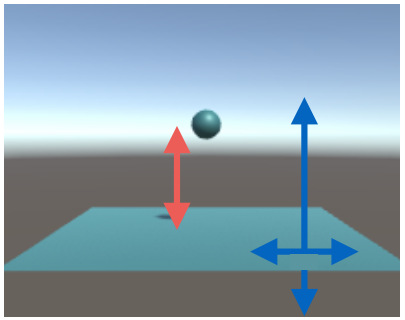
送信編

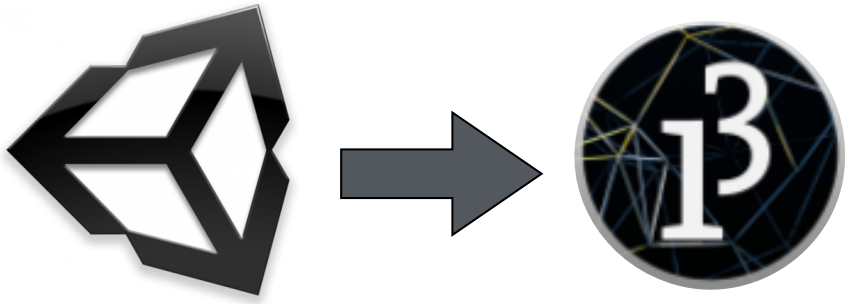
127.0.0.1 / 6666

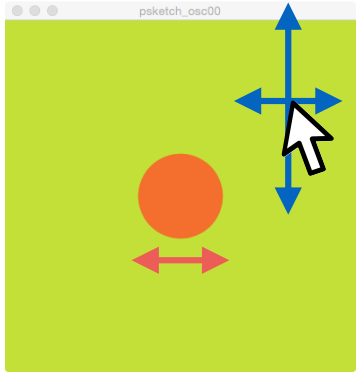
127.0.0.1 / 5555

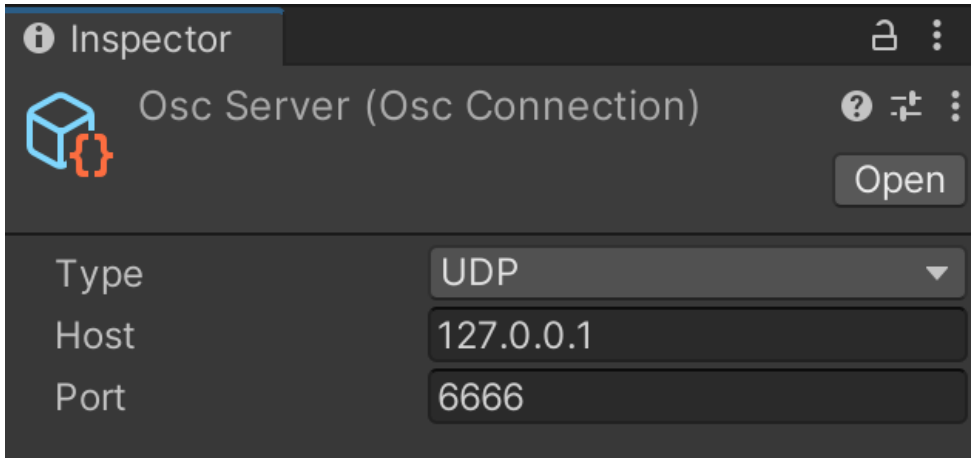
SERVER

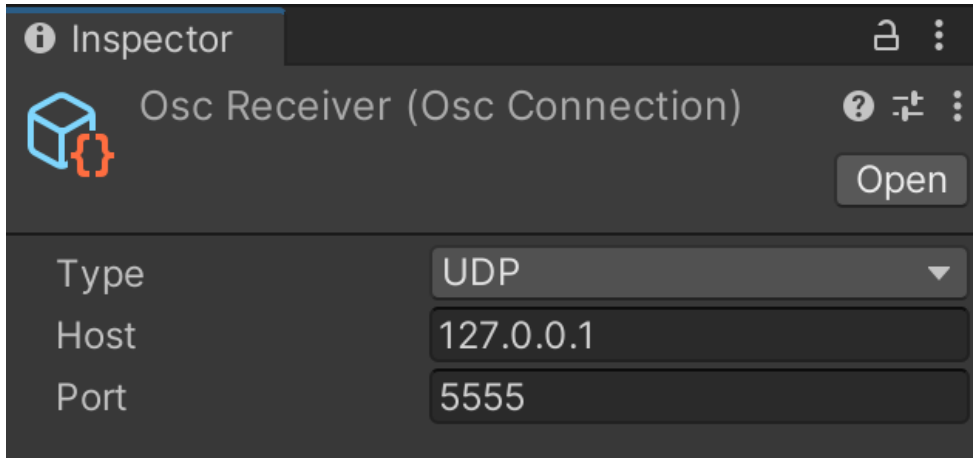
Client









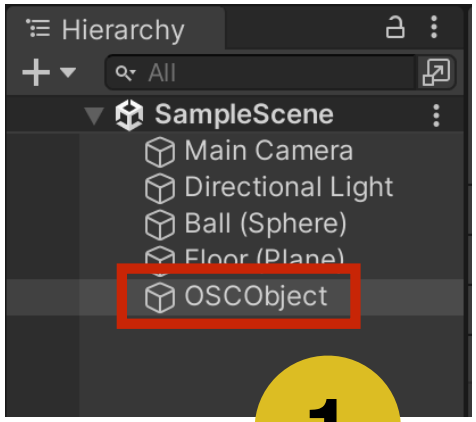


UNITY側の準備

送信編

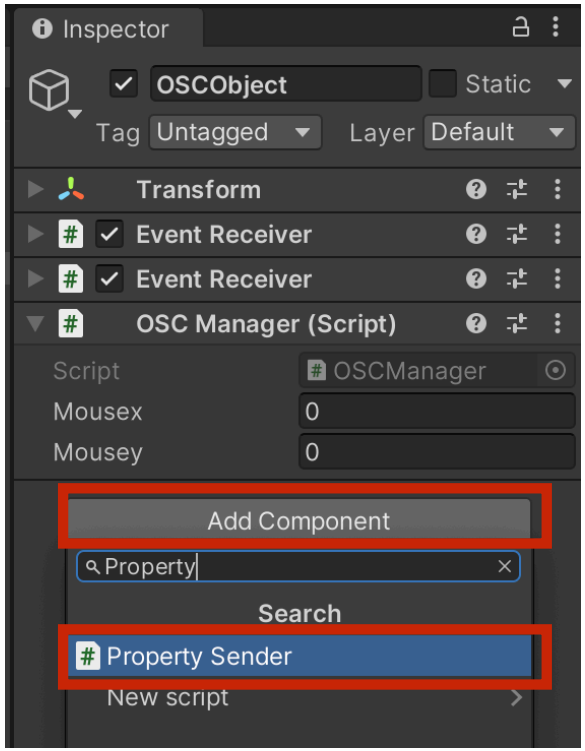
GameObjectの三次元位置（XYZ）を送信する（1/2）

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。



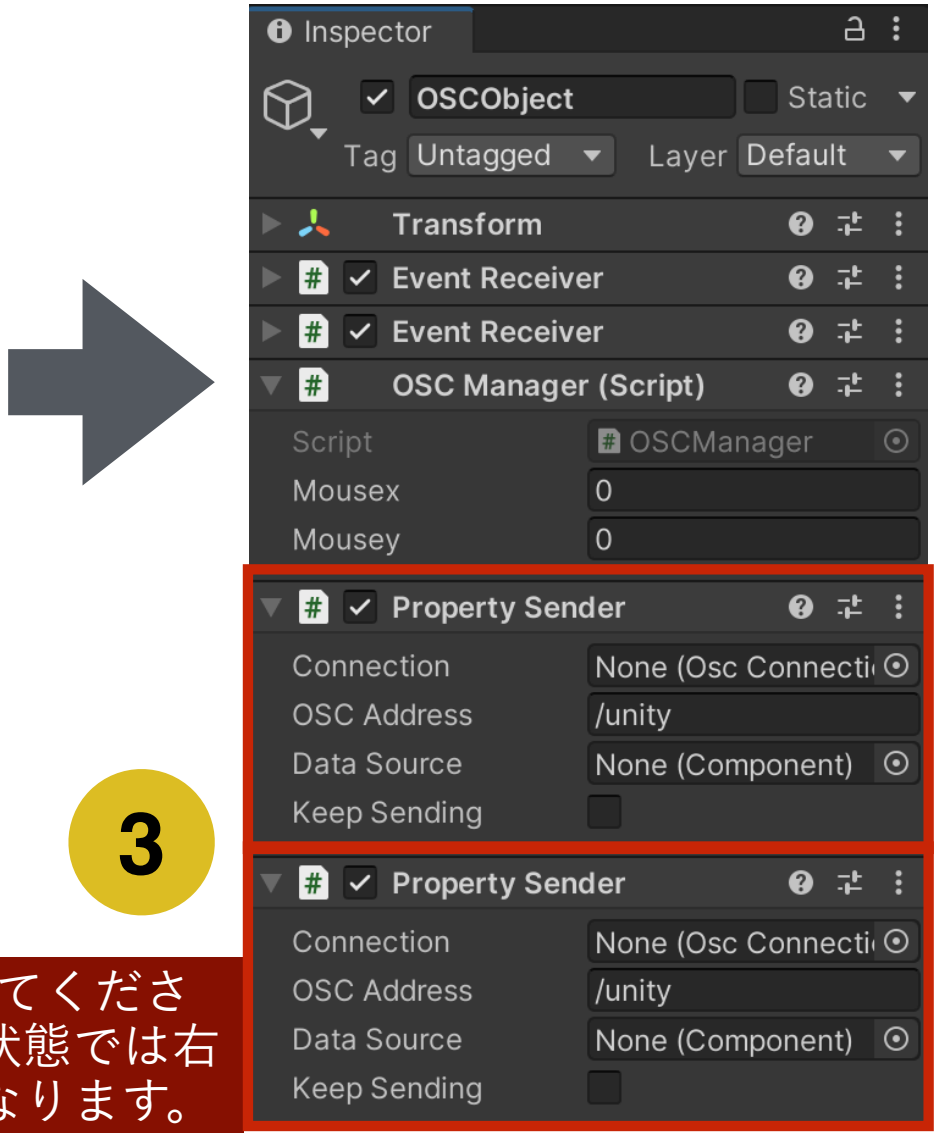
1

OSCObjectを選択した状態で、



2

Add ComponentでProperty Senderを追加します。



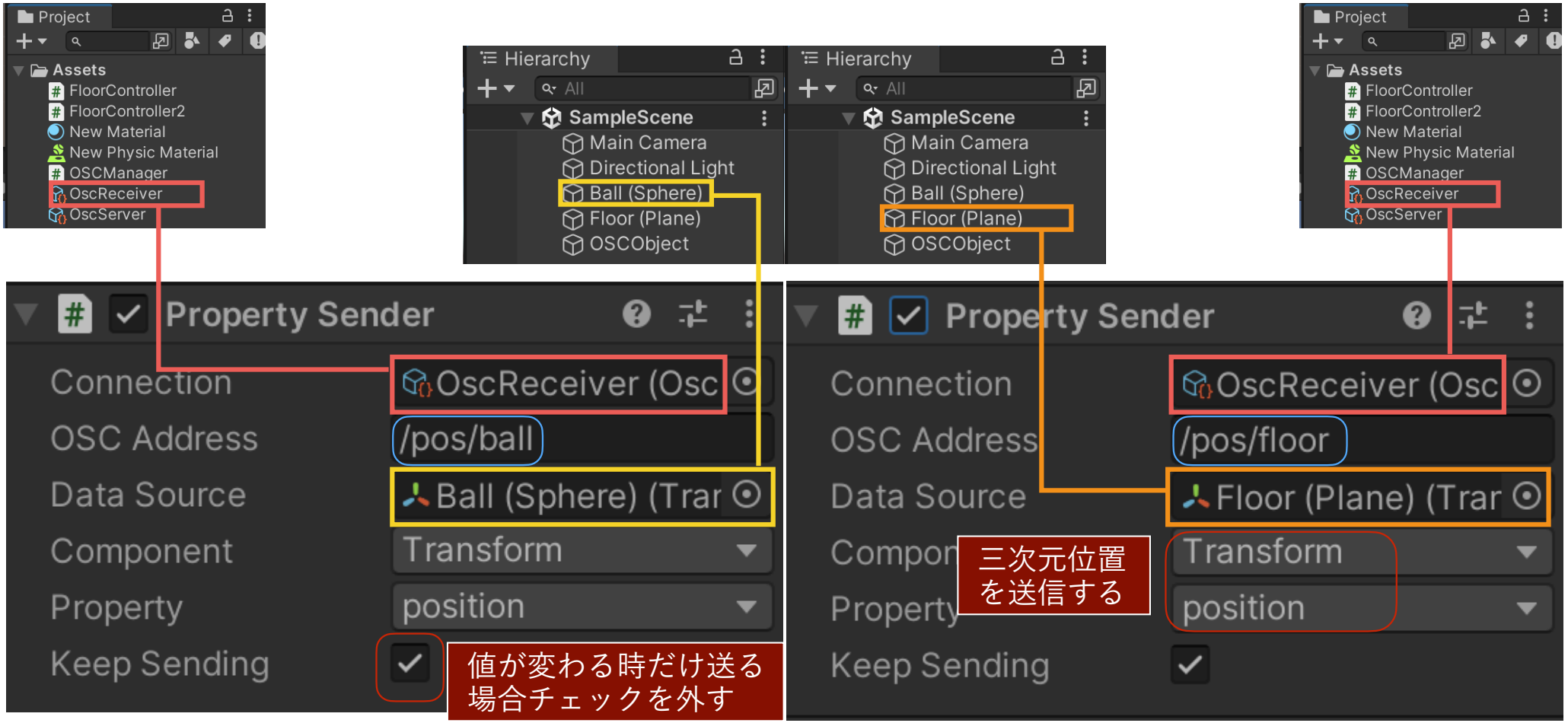
3

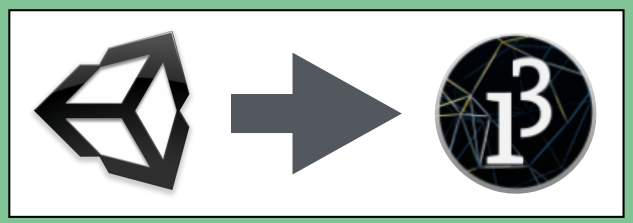
2つ追加してください。初期状態では右のようになります。

UNITY側の準備

送信編

床と球の三次元位置（XYZ）を送信する（2/2）





`/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.

UNITY側の準備

送信編

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

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using OscJack;
5
6  public class FloorController : MonoBehaviour
7  {
8      public GameObject oscobj = null;
9      private OSCManager oscmng;
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     }
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();
34     }
35
36     void OnCollisionEnter(Collision collision){
37         count++;
38         Debug.Log("Collision!!" + count);
39         client.Send("/hit", count);
40     }
41 }
```

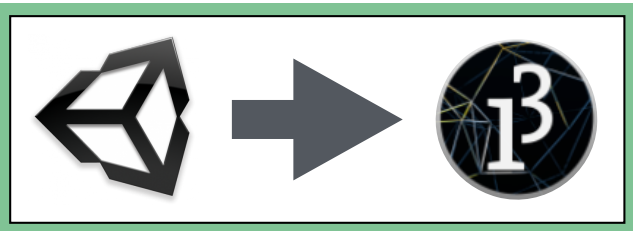
OscJackのライブラリを使います。

送信先のOSC情報をまとめたものです。

Processing側の端末をオブジェクト化したものです。

ネットワークが壊れた時の処理？おまじないと思って書いて下さい。

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.

UNITY側の準備

送信編

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