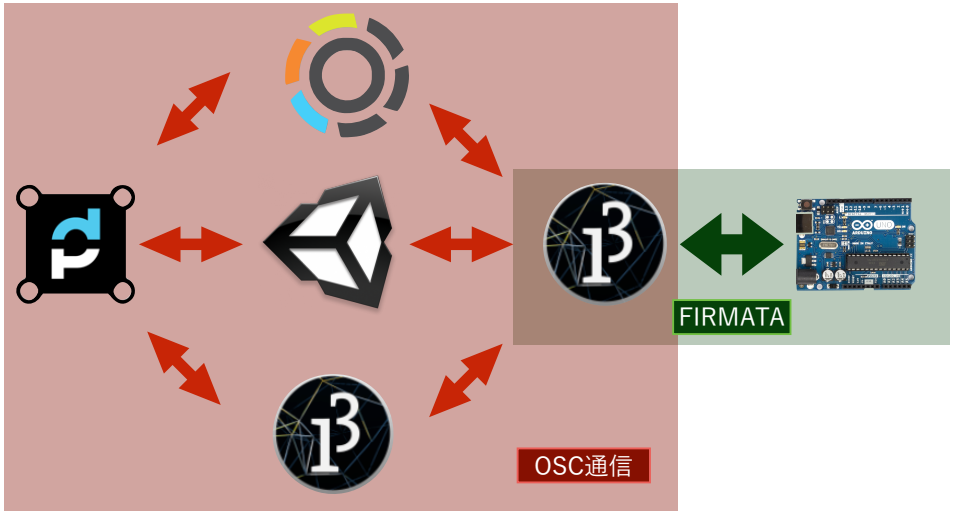
[演習]
UNITY - Processing
アプリケーション間通信（OSC通信）

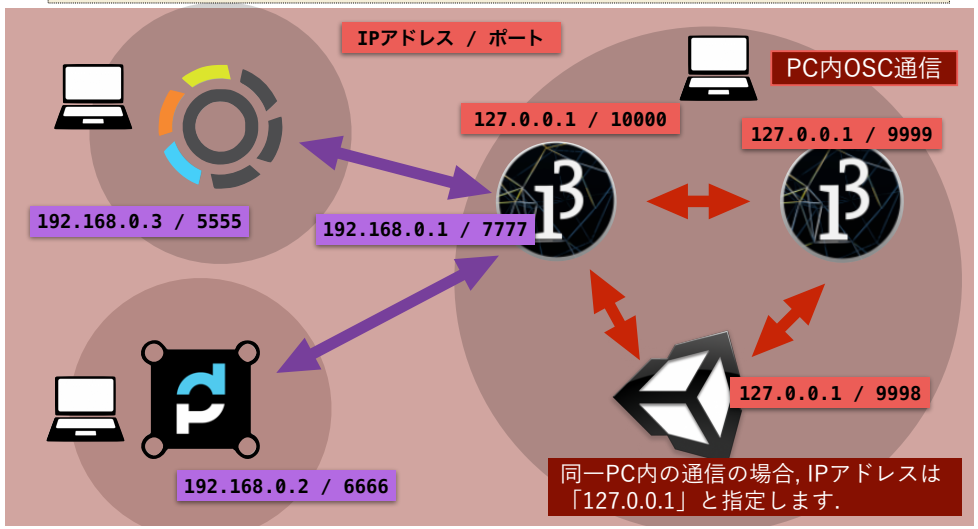
アプリケーション間通信の全体像

ArduinoとProcessingはFIRMATA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．



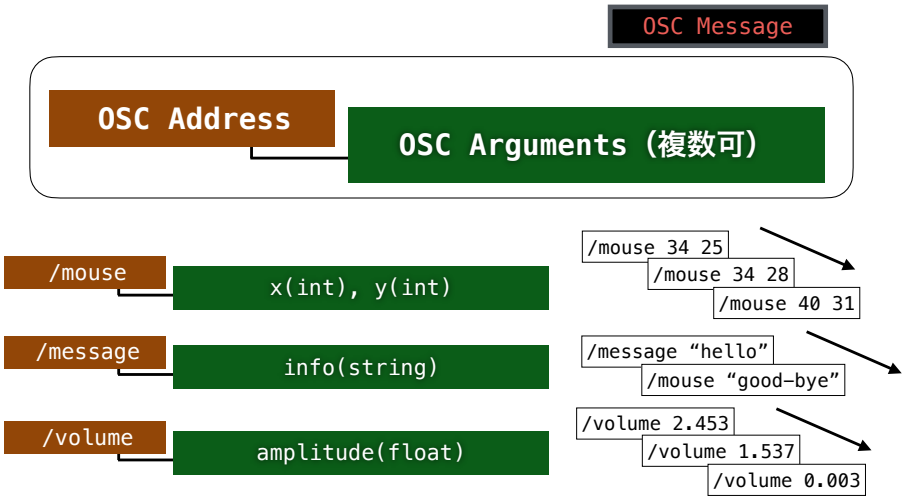
OSC（Open Sound Control）通信
IPアドレスとポート

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

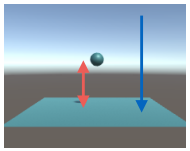
OSC通信は，送信するデータをパッケージにして送ります．パッケージの中身として，（1）どのような形式のデータが入っているのか（OSC Arguments），そして（2）パッケージの宛名（OSC Address）を指定する必要があります．

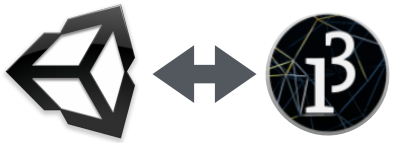


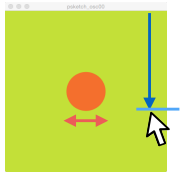
実装練習（UNITY 対 Processing の通信）

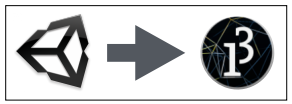
127.0.0.1 / 6666　　127.0.0.1 / 5555

SERVER　　Client









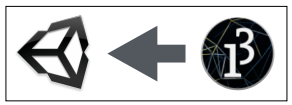
/eventA
- ボールと床の距離（float）

/eventB　球と床の距離を円の直径と連動させます.
- 衝突回数（float），衝突時の床の高さ（float）

衝突のタイミングで，背景色を変えます.

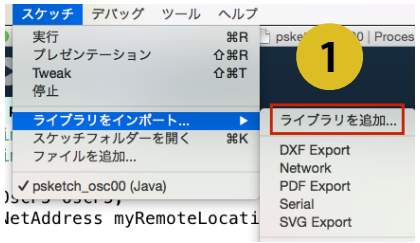
/mouse
- X座標（int），Y座標（int）

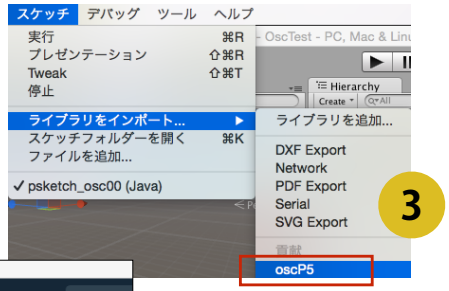
マウスのY座標を，床の高さと連動させます.

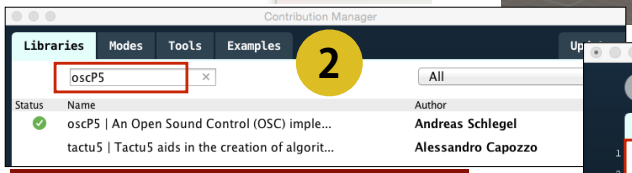


Processing側の準備

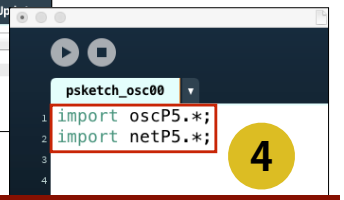
ライブラリ（oscP5）の追加







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（1/2）

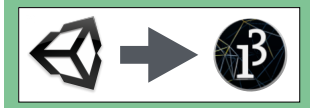
```
import oscP5.*;
import netP5.*;

OscP5 oscP5;
NetAddress myRemoteLocation;

float dif = 0; //ボールと床の距離
color col;     //背景色（衝突の度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,"getDif","/eventA");
  oscP5.plug(this,"getHit","/eventB");
}
```

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.

127.0.0.1 / 5555

127.0.0.1 / 6666

oscアドレス「/eventA（/eventB）」を受け取ると，自作関数「getDif（getHit）」をコールアップします.



/eventA
- ボールと床の距離（float）

/eventB　球と床の距離を円の直径と連動させます.
- 衝突回数（float），衝突時の床の高さ（float）

衝突のタイミングで，背景色を変えます.

Processing側の準備

Processingのコード（2/2）

```
void draw() {

  background(col);  noStroke(); fill(255,100,0);
  ellipse(width/2.,height/2.,50.*dif,50.*dif);

  if(mousePressed){
    OscMessage myMessage = new OscMessage("/mouse");
    myMessage.add(mouseX); myMessage.add(mouseY);
    oscP5.send(myMessage, myRemoteLocation);
  }
}

void getDif(float d){
  dif = d;
}

void getHit(float count, float h) {
  col = color(random(255),random(255),random(255));
  println("[count,height_of_floor]=["+(int)count+","+h+"]");
}
```

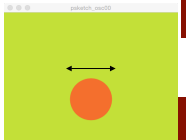
マウスが押されている場合，マウスのXY座標を，OSCメッセージとして，UNITYに送ります.

getDif関数が呼ばれるたびに，difを変更します.

```
oscP5.plug(this,"getDif","/eventA");
oscP5.plug(this,"getHit","/eventB");
```

コールアップ関数の引数の数・型は，対応するOSCアーギュメンツの形式に合わせます.

getHit関数が呼ばれるたびに，背景色（col）がランダムに変更します.

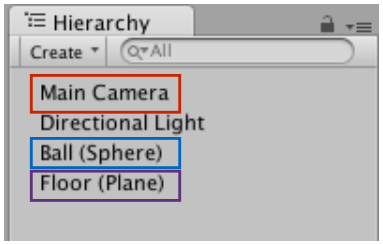


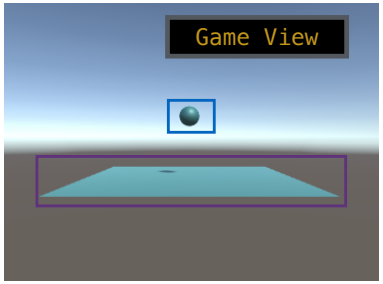
円の中心を，windowの中心位置に，直径を50*difで描画します.

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

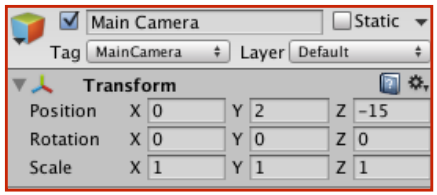
/mouse
- X座標（int），Y座標（int）

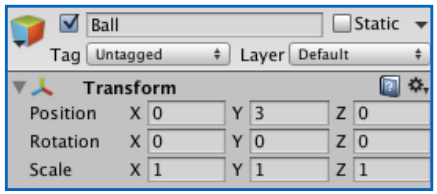
UNITY側の準備
基本環境の構築（ゲームオブジェクトの追加）

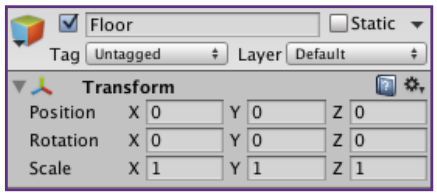




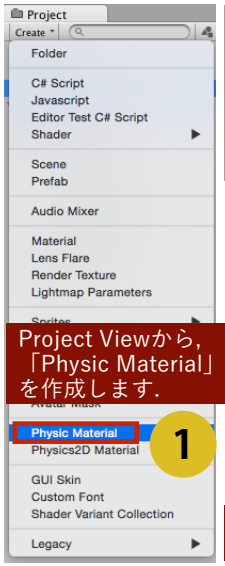
必要に応じて, Materialを追加して色をつけてください.



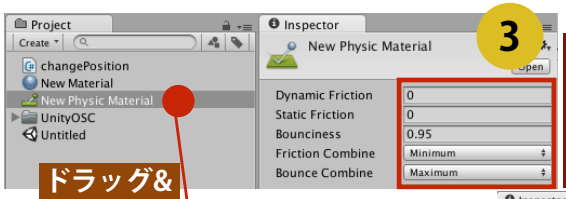




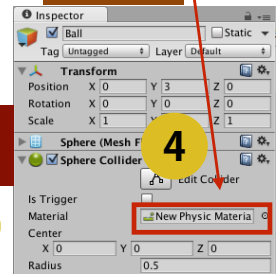
UNITY側の準備
基本環境の構築（ボールがバウンドするようにする）



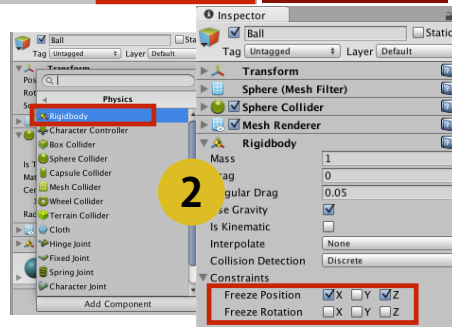
Project Viewから,「Physic Material」を作成します.



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.

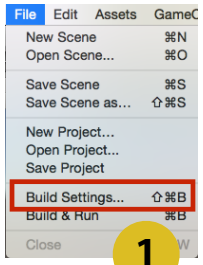


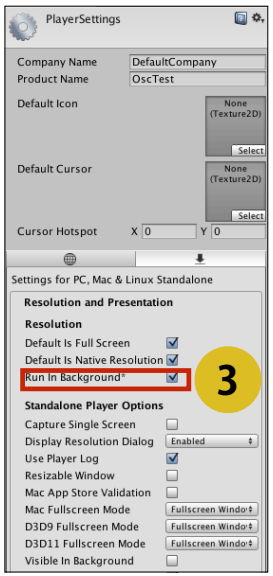
作成した物理特性をBallと関連付けます.



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）

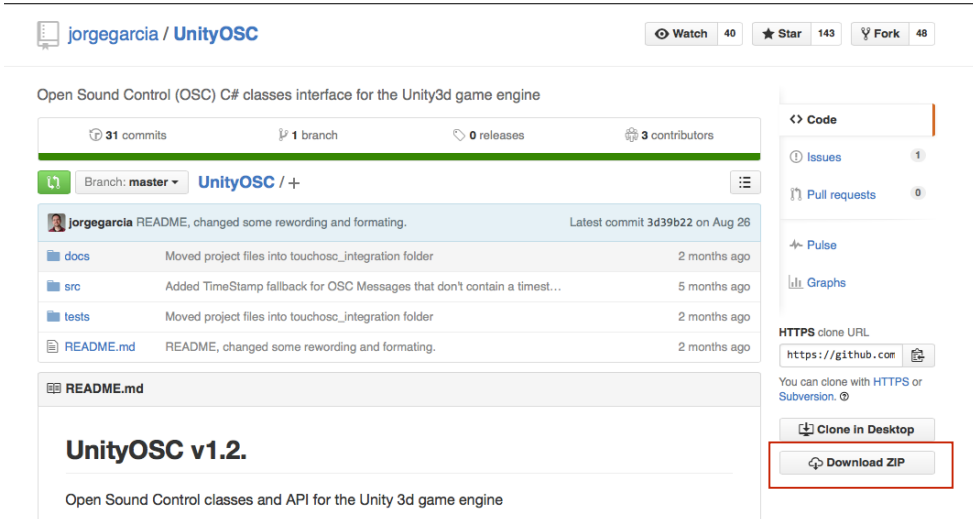




Processingの実行アプリケーションを立ち上げている間にも, UNITYの描画処理をキャンセルしないための措置です.

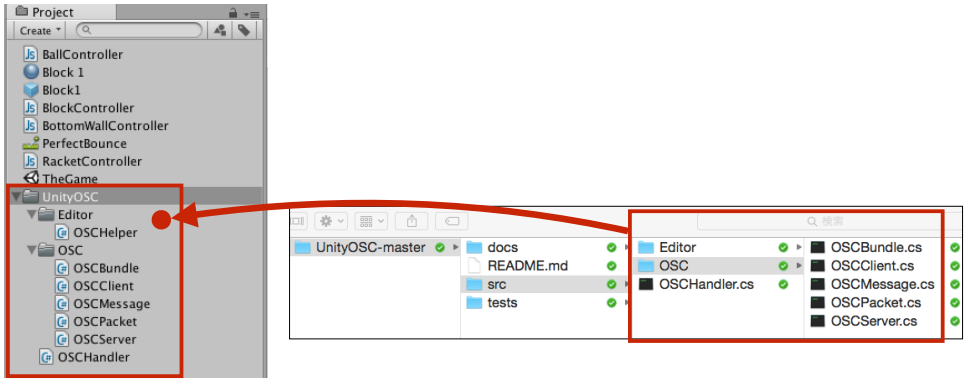
UNITY側の準備
UnityOSCのダウンロード

https://github.com/jorgegarcia/UnityOSC



UNITY側の準備
UnityOSCの作業Projectへの追加

- DLしたファイル内のsrcのフォルダを、ProjectのAssetsの中にドラッグ＆ドロップしてください。srcは適当な名前にリネームしましょう（以下の例では, UnityOSCにします）。

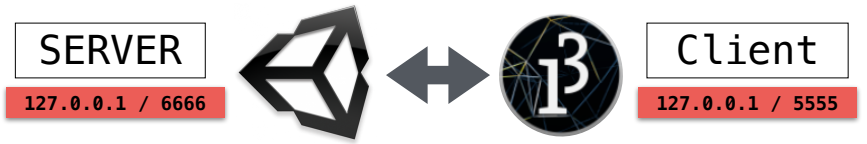


UNITY側の準備
OSCHandler.csの編集

- OSCHandler.csを開き, Init() 関数（92行目以降）内に, OSCサーバのClientとServerのアドレス・ポートに関する情報を付加します。ここでは, 以下のような接続環境を想定しましょう。

```
93    public void Init()
94    {
95        //Initialize OSC clients (transmitters)
96        //CreateClient("SuperCollider", IPAddress.Parse("127.0.0.1"), 5555);
97        CreateClient("Processing", IPAddress.Parse("127.0.0.1"), 5555);
98
99
100       //Initialize OSC servers (listeners)
101       //CreateServer("AndroidPhone", 6666);
102       CreateServer("Unity", 6666);
103
104   }
```

OSCHandler.cs
Init()



UNITY側の準備
OSCManager.csの追加・編集

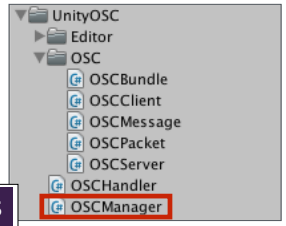
http://lab.kenrikodaka.com/_download/UNITY/OSCManager.cs

- 以上より, OSCManager.cs をダウンロードし, OSCHandler.csと同じ階層にドラッグ&ドロップしてください.

```
7  public class  OSCManager : MonoBehaviour {
8
9      private long _lastOscTimeStamp = -
10
11     /** OSC Argument(s) */
12     public int dataA1 = 0;
13     public int dataA2 = 0;
```

宣言部・Start()

OSC ArgumentsがOSCManagerクラスのフィールドとして定義されています.（名前は自由に変えてください）




```
25     void Update () {
26         OSCHandler.Instance.UpdateLogs();
27
28         foreach( KeyValuePair<string, ServerLog> item in OSCHandler.I
29             for( int i=0; i < item.Value.packets.Count; i++ ) {
30                 if( _lastOscTimeStamp < item.Value.packets[i].TimeSta
31
32                     _lastOscTimeStamp = item.Value.packets[i].TimeSta
33
34                     string address = item.Value.packets[i].Address;
35
36                     if(address == "/mouse") {
37                         dataA1 = (int)item.Value.packets[i].Data[0];
38                         dataA2 = (int)item.Value.packets[i].Data[1];
39                     }
```

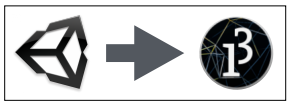
OSCManager.cs

Update()

OSC Address が /mouse の場合, パケットの二つのデータをdataA1, dataA2に代入します.

/mouse
int x, int y

OSC Message



/eventA
ボールと床の距離（float）

/eventB
球と床の距離を円の直径と連動させます.
衝突回数（float）, 衝突時の床の高さ（float）
衝突のタイミングで, 背景色を変えます.

sendDataOSCA()　sendDataOSCB()

OSCManager.cs

```
54     public void sendDataOSCA(float dataA1){
55         var sampleVals = new List<float>(){dataA1};
56         OSCHandler.Instance.SendMessageToClient("Processing", "/eventA", sampleVals);
57     }
58
59     public void sendDataOSCB(float dataB1, float dataB2){
60         var sampleVals = new List<float>(){dataB1, dataB2};
61         OSCHandler.Instance.SendMessageToClient("Processing", "/eventB", sampleVals);
62     }
```

publicを追加してください.

dataA1（dataB1・dataB2）を要素とするリスト sampleVals を作成します.

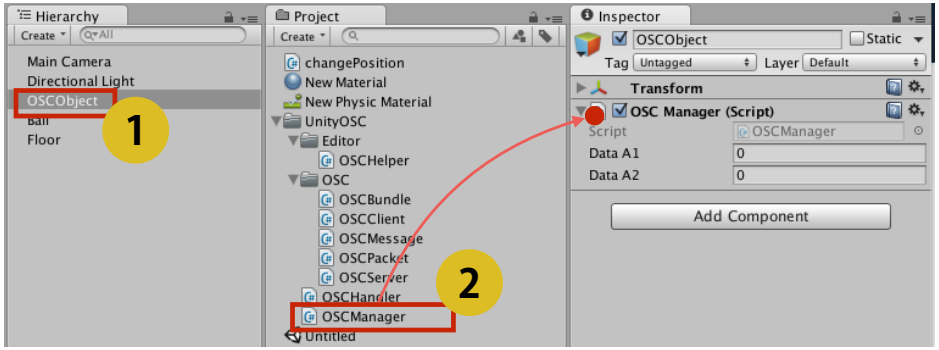
OSCHandler.csにおいて"Processing"で定義されたネットワーク端末に, /eventA（/eventB）をOSC Address として, sampleValsを送ります.

- ここで定義する関数 の 引数の型と数は, OSC Argumentsの型と数に合わせて修正してください.

UNITY側の準備

空のゲームオブジェクトの作成

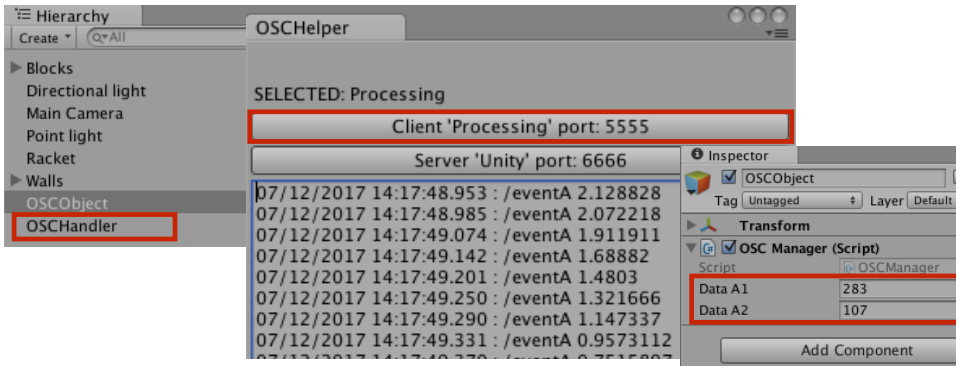
- シーンの中に, 空のオブジェクト（Create Empty）を作成し, 名前をOSCObjectにリネームします.
- 既に作成しているOSCManagerをOSCObjectのインスペクタにドラッグ&ドロップします.
- これで, シーンが起動すると同時に, OSCManager.csが動作します.

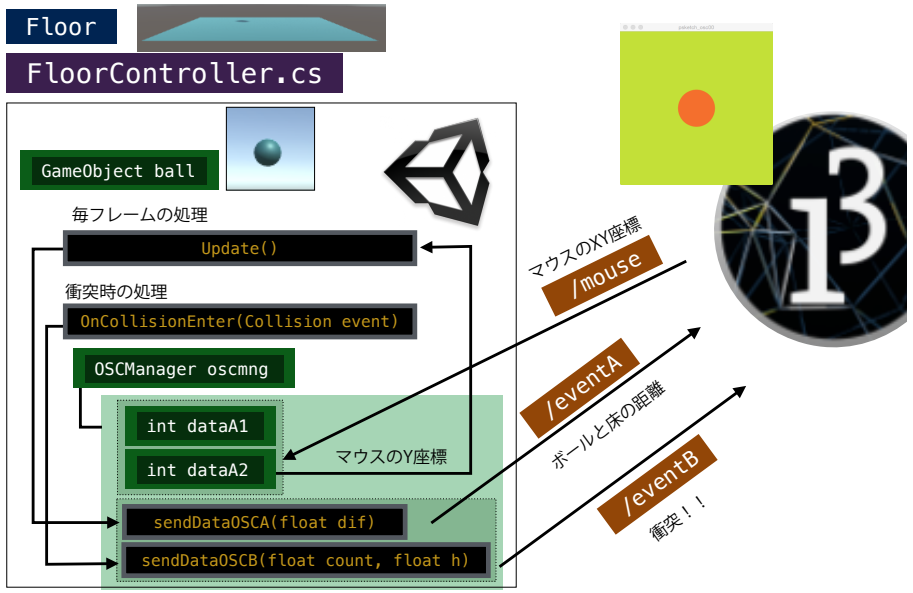


UNITY側の準備

OSCHelperによるOSC Messageのモニタ

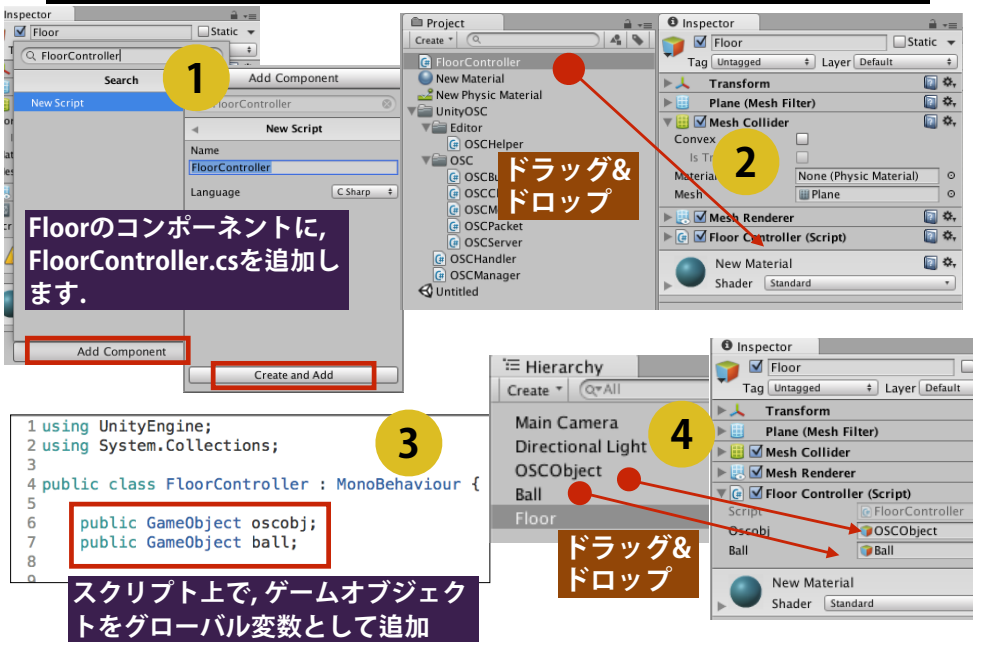
- これで実行（Run）すると, シーンの中に OSCHandler が現れるはずです.
- メニューバーの Window から OSC Helper を選択し, Serverをクリックし, Processingを実行すると, マウス座標に関する OSCメッセージが正しく送られていることを確認することができます.
- 同時に, OSCObjectのインスペクタの Data A1・DataA2 の値が正しく反応していることを確認してください.





UNITY側の準備（インタラクション部）

FloorController.csコンポーネントの作成



UNITY側の準備（インタラクション部）

FloorController.csを編集

```
4 public class FloorController : MonoBehaviour {
5
6     public GameObject oscobj;
7     public GameObject ball;
8
9     private OSCManager oscmng;
10    int count = 0;
11
12    void Start () {
13        oscmng = (OSCManager)oscobj.GetComponent ("OSCManager");
14    }
15
16    void Update () {
17        //床のY座標を設定
18        float y = 0f - oscmng.dataA2 * 0.01f;
19        this.transform.position = new Vector3 (0f, y, 0f);
20        //ボールと床のY座標の差分をProcessingに送る
21        float dif = ball.transform.position.y - y;
22        oscmng.sendDataOSCA (dif);
23    }
24
25    void OnCollisionEnter(Collision collision) {
26        Debug.Log ("Collision!! " + count);
27        count++;
28        oscmng.sendDataOSCB(count,this.transform.position.y);
29    }
```

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

Processingのマウスの Y座標に相当します.

毎フレーム行われる処理を記述します.

ボールと床の距離を Processingに送ります.

床が, なんらかの物体と交錯した場合にコールアップされるイベント関数です.

衝突回数と衝突時の床の高さをProcessingに送ります.