

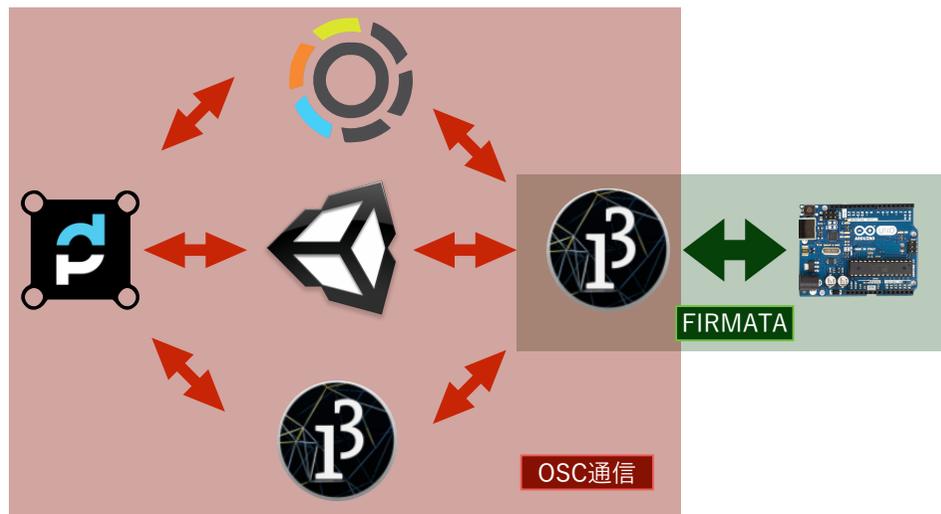
[演習]

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

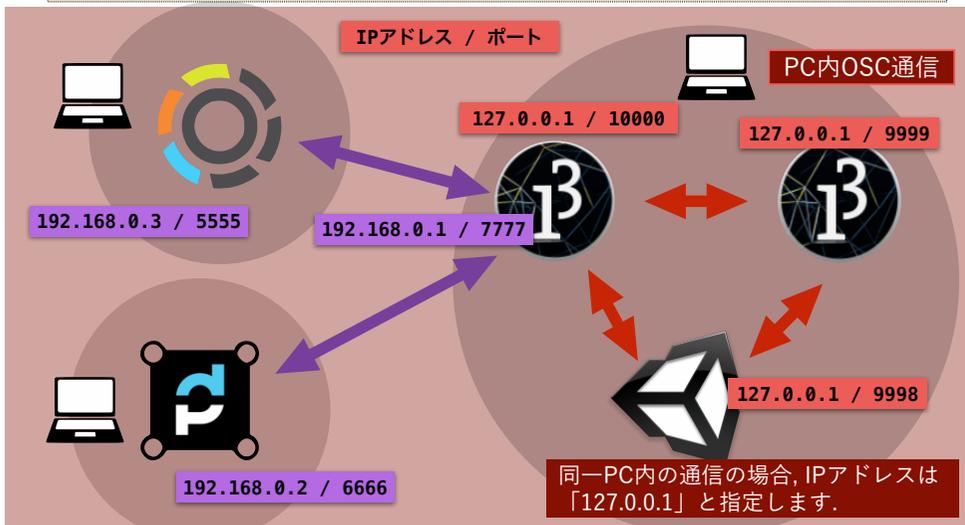
ArduinoとProcessingはFIRMATAで、ProcessingとPC内のアプリケーションはOSC通信で、相互に情報をやりとりすることができます。



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。

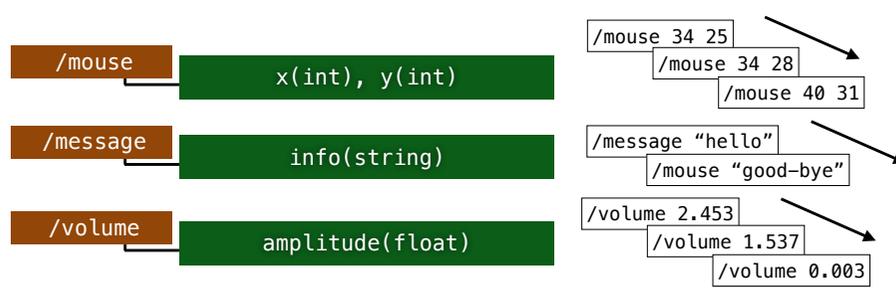


OSC (Open Sound Control) 通信

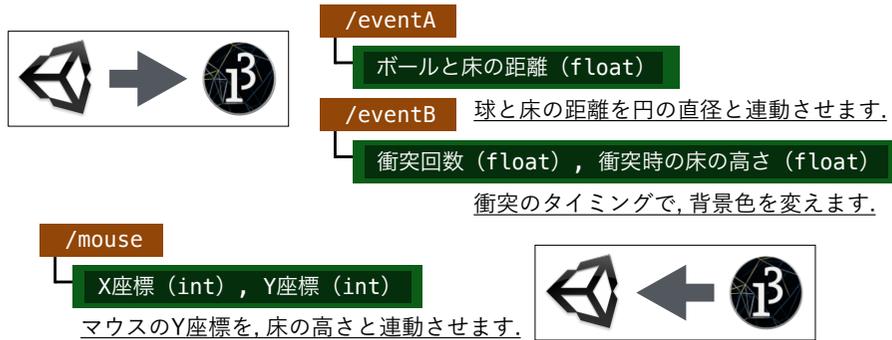
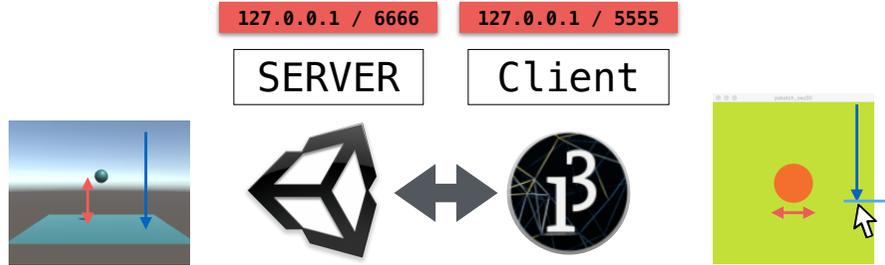
OSC Message

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。

OSC Message



# 実装練習 (UNITY 対 Processing の通信)



# Processing側の準備 ライブラリ (oscP5) の追加

1 ライブラリを追加...

2 oscP5

3 oscP5

4 import oscP5.\*;  
import netP5.\*;

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)

```

1 import oscP5.*;
2 import netP5.*;
3
4 OscP5 oscP5;
5 NetAddress myRemoteLocation;
6
7 float dif = 0; //ボールと床の距離
8 color col; //背景色 (衝突の度にランダムに変わる)
9
10 void setup() {
11   size(400,400); frameRate(10); background(0);
12
13   oscP5 = new OscP5(this,5555);
14   myRemoteLocation = new NetAddress("127.0.0.1",6666);
15
16   oscP5.plug(this,"getDif","/eventA");
17   oscP5.plug(this,"getHit","/eventB");
18 }

```

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)

```

20 void draw() {
21
22   background(col); noStroke(); fill(255,100,0);
23   ellipse(width/2.,height/2.,50.*dif,50.*dif);
24
25   if(mousePressed){
26     OscMessage myMessage = new OscMessage("/mouse");
27     myMessage.add(mouseX); myMessage.add(mouseY);
28     oscP5.send(myMessage, myRemoteLocation);
29   }
30
31   getDif関数が呼ばれるたびに、difを変更します。
32   void getDif(float d){
33     dif = d;
34   }
35
36   void getHit(float count, float h) {
37     col = color(random(255),random(255),random(255));
38     println("[count,height_of_floor]=["+(int)count+" "+h+""]);
39   }

```

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。

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アーギュメント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。

**/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)

The image shows the Unity Hierarchy and Inspector panels. In the Hierarchy, 'Main Camera' is selected and highlighted with a red box, 'Ball (Sphere)' with a blue box, and 'Floor (Plane)' with a purple box. The Inspector panels show the Transform properties for each object: Main Camera (Position X:0, Y:2, Z:-15), Ball (Position X:0, Y:3, Z:0), and Floor (Position X:0, Y:0, Z:0). A Game View window shows a 3D scene with a camera, a ball, and a floor plane.

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)

The image shows the Unity Project, Inspector, and Hierarchy panels. In the Project panel, a 'New Physic Material' is created and highlighted with a red box and a yellow circle '1'. A red arrow labeled 'ドラッグ&ドロップ' points to the 'New Physic Material' button in the Inspector. The Inspector shows the 'New Physic Material' settings with 'Bounciness' set to 0.95, highlighted with a red box and a yellow circle '3'. The Hierarchy panel shows the 'Ball' object with 'Rigidbody' and 'Sphere Collider' components added, highlighted with a yellow circle '2'. A red box labeled '作成した物理特性をBallと関連付けます。' points to the 'Ball' object in the Hierarchy. A red box labeled 'Project Viewから、「Physic Material」を作成します。' points to the 'New Physic Material' button in the Project panel. A red box labeled 'コンポーネントとしてRigidbodyを追加し、Y座標(上下)のみを動かようにします。' points to the 'Rigidbody' component in the Inspector.

## 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)

The image shows the Unity Build Settings and Player Settings panels. In the Build Settings panel, 'Build Settings...' is highlighted with a red box and a yellow circle '1'. In the Player Settings panel, 'Run in Background' is checked and highlighted with a red box and a yellow circle '3'. A yellow circle '2' is also present in the Player Settings panel.

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

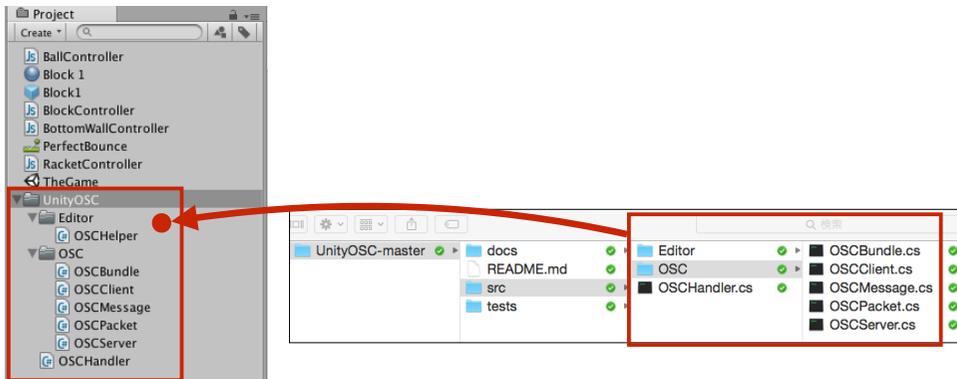
<https://github.com/jorgegarcia/UnityOSC>

The image shows the GitHub repository page for 'jorgegarcia / UnityOSC'. The repository name is highlighted with a red box. The page shows 31 commits, 1 branch, 0 releases, and 3 contributors. The README.md file is highlighted with a red box. The 'Download ZIP' button is highlighted with a red box.

## 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）。



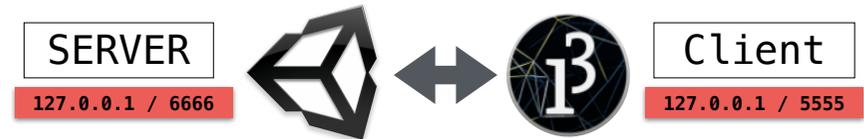
## UNITY側の準備

### 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 = 宣言部・Start()
10
11     /** OSC Argument(s) */ OSC ArgumentsがOSCManagerクラス
12     public int dataA1 = 0; のフィールドとして定義されていま
13     public int dataA2 = 0; す。(名前は自由に変えてください)
```

```
25 void Update () {
26     OSCHandler.Instance.UpdateLogs(); OSCManager.cs
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         }
40     }
```

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。

OSC Message  
/mouse  
int x, int y

The screenshot shows the 'UnityOSC' folder structure with 'OSCManager.cs' being added to the 'OSC' folder.



```
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。

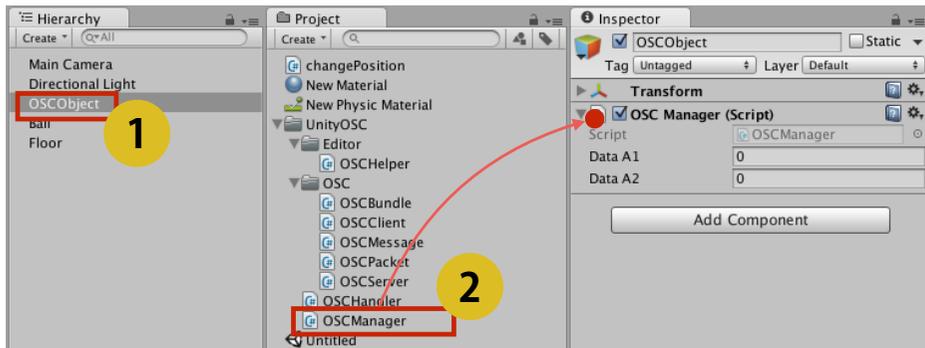
OSCHandler.csにおいて"Processing"で定義されたネットワーク端末に, /eventA (/eventB) を OSC Address として, sampleValsを送ります。

- 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

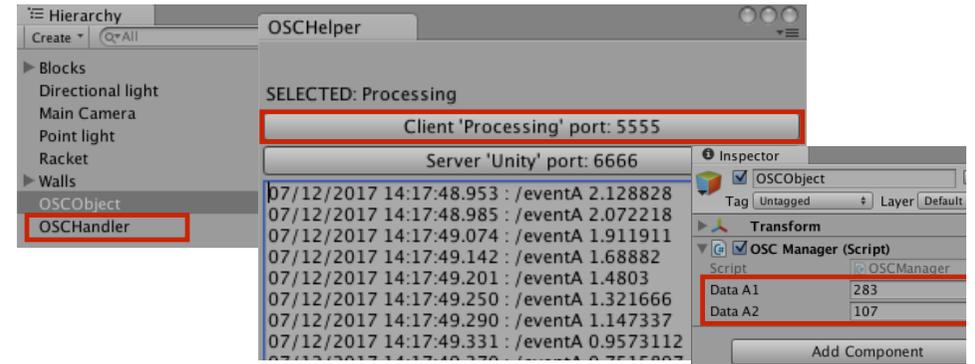
- シーンの中に、空のオブジェクト (Create Empty) を作成し、名前をOSCObjectにリネームします。
- 既に作成しているOSCManagerをOSCObjectのインスペクタにドラッグ&ドロップします。
- これで、シーンが起動すると同時に、OSCManager.csが動作します。



## UNITY側の準備

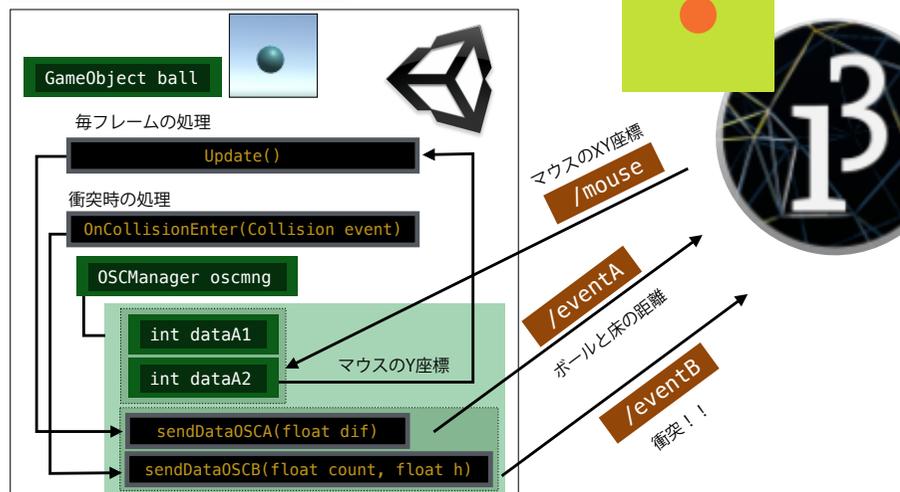
### OSCHelperによるOSC Messageのモニタ

- 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。



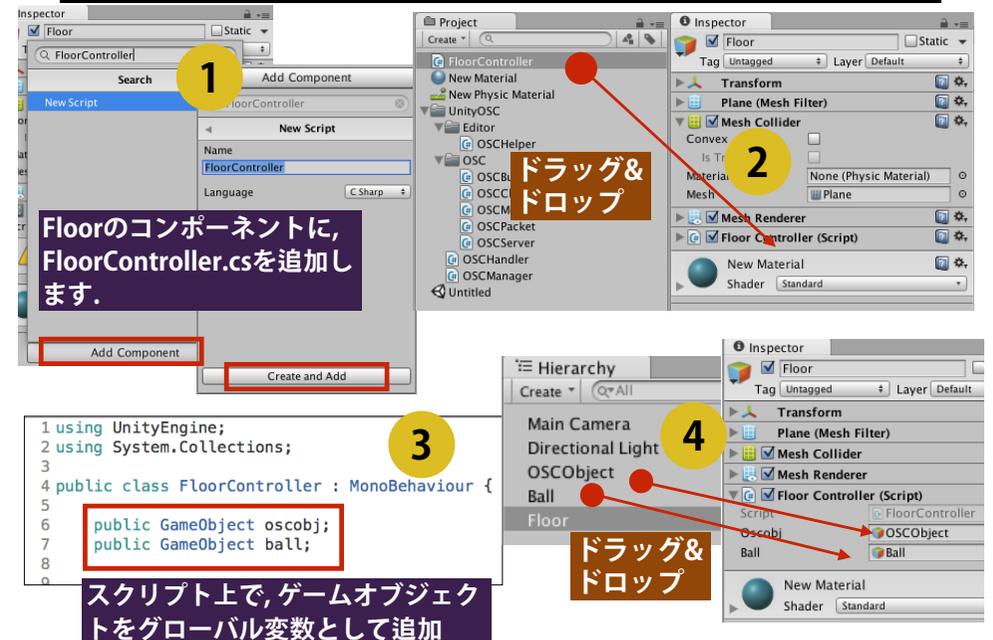
Floor

FloorController.cs



## 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



1 using UnityEngine;  
2 using System.Collections;  
3  
4 public class FloorController : MonoBehaviour {  
5  
6 public GameObject oscobj;  
7 public GameObject ball;  
8  
9  
0

ドラッグ&ドロップ  
Floorのコンポーネントに、FloorController.csを追加します。

ドラッグ&ドロップ

## 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.sendData0SCA (dif);
23    }
24
25    void OnCollisionEnter(Collision collision) {
26        Debug.Log ("Collision!! " + count);
27        count++;
28        oscmng.sendData0SCB (count, this.transform.position.y);
29    }
30 }
```

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。