演 習 １ ： U N I T Y の 基 礎

(01) 04/13

１ A｜Unityとエディタの連携

(02) 04/20

１ B｜Transform・キーイベント・マウスイベント

(03) 04/27

１ C｜剛体特性・カメラの視点

(04) 05/11

１ D｜プレハブ（gameobjectの雛形）, タグ, その他

(05) 05/18

１ E｜アバターのアニメーション

MediaPractice04

プレハブ（gameobjectの雛形），タグ

Project name*

MediaPractice04

Location*

/Users/kenri11/Dropbox/DocClass/_B3_メディアエ ...

3D 2D

Create project

Asset packages...

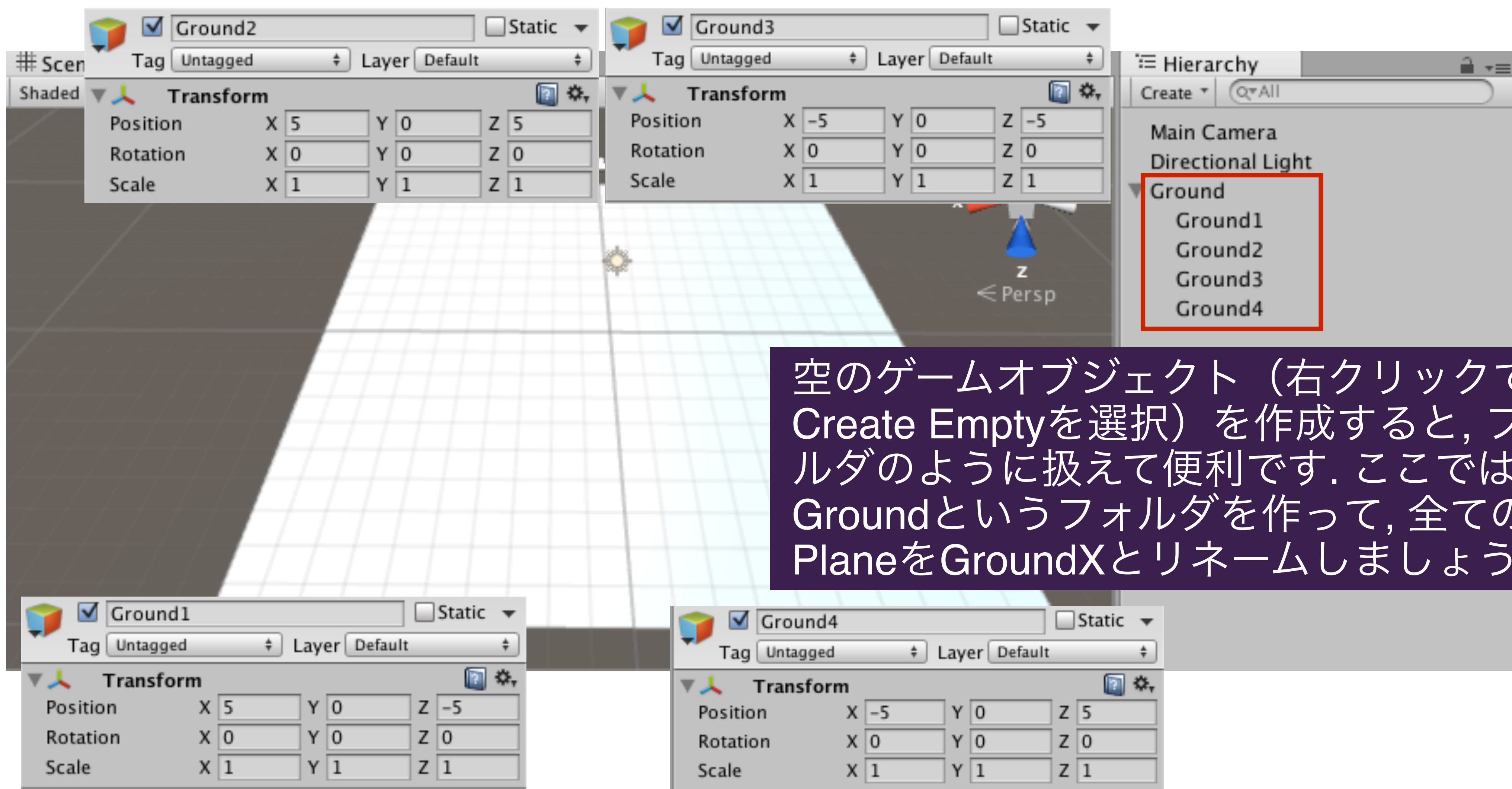
MediaPractice04では,

https://github.com/unity3d-jp/FirstTutorial/wiki

を土台にして, サンプルを作成していきます.

床の配置

- ４枚のPlaneを, それぞれがぴったりと重なるように配置し, 床として使用します.

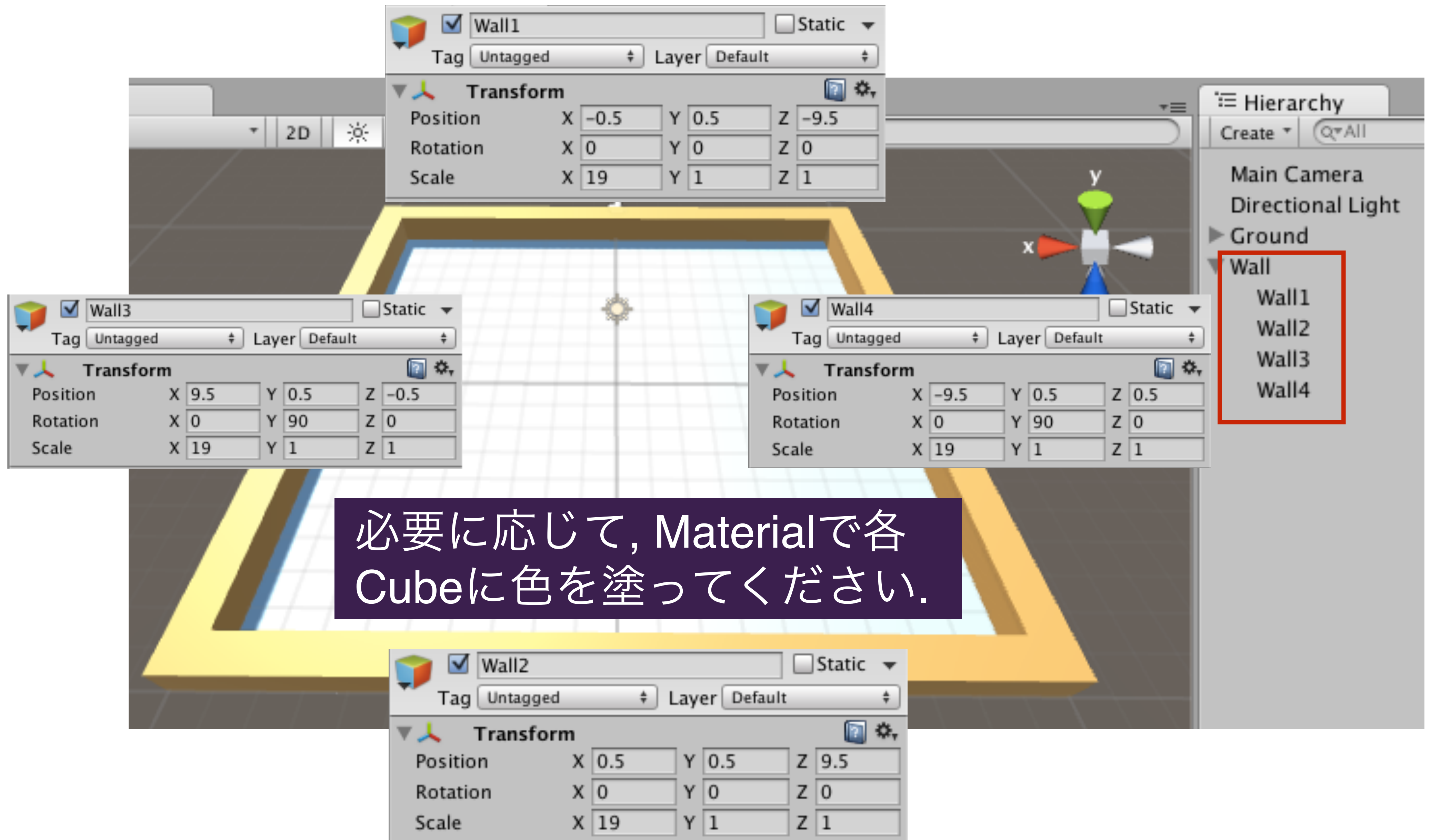


空のゲームオブジェクト（右クリックでCreate Emptyを選択）を作成すると, フォルダのように扱えて便利です. ここでは, Groundというフォルダを作って, 全てのPlaneをGroundXとリネームしましょう.

必要に応じて, Materialで各Planeに色を塗ってください.

壁の配置

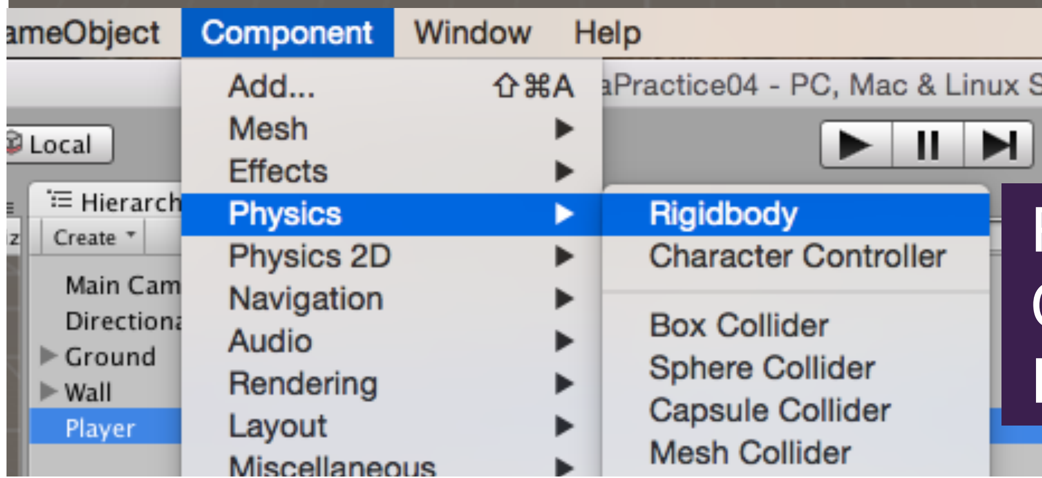
- 同様に，４枚のcubeを配置して，四方を囲む壁を作ります．



Playerを作る

- Sphereを少し高い位置に配置し，コンポーネントとして剛体特性（rigidbody）を追加します．

Sphereを Playerと改名しましょう．



Playerを選択した状態で，Componentメニューから Rigidbodyを追加します．

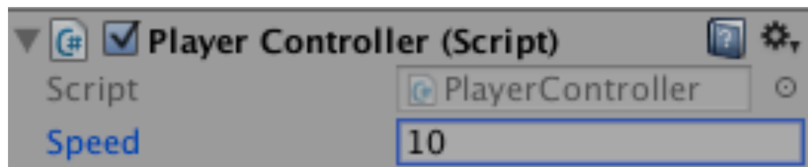
Playerをコントロールする

- Playerのコンポーネントとして新たにスクリプトを追加して, 上下左右の矢印 key で, シーン内のPlayerを動かせるようにします.

```
using UnityEngine;
using System.Collections;

public class PlayerController : MonoBehaviour {
    // speedを制御する
    public float speed = 10;

    void Start(){
    }

    void Update ()
    {
        float x =  Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Rigidbody rigidbody = GetComponent<Rigidbody>();

        // xとyにspeedを掛ける
        rigidbody.AddForce(x * speed, 0, z * speed);
    }
}
```

PlayerController.cs

Input
float *GetAxis(string axis)
axisの軸に関する入力値(-1~1)を取り出す
float *GetAxis("Horizontal")
水平方向に関する入力値を取り出す. Keyの場合, 左矢印で-1, 右矢印で+1, それ以外は0を返します.
float *GetAxis("Vertical")
上矢印で+1, 下矢印で-1, それ以外では0を返します.

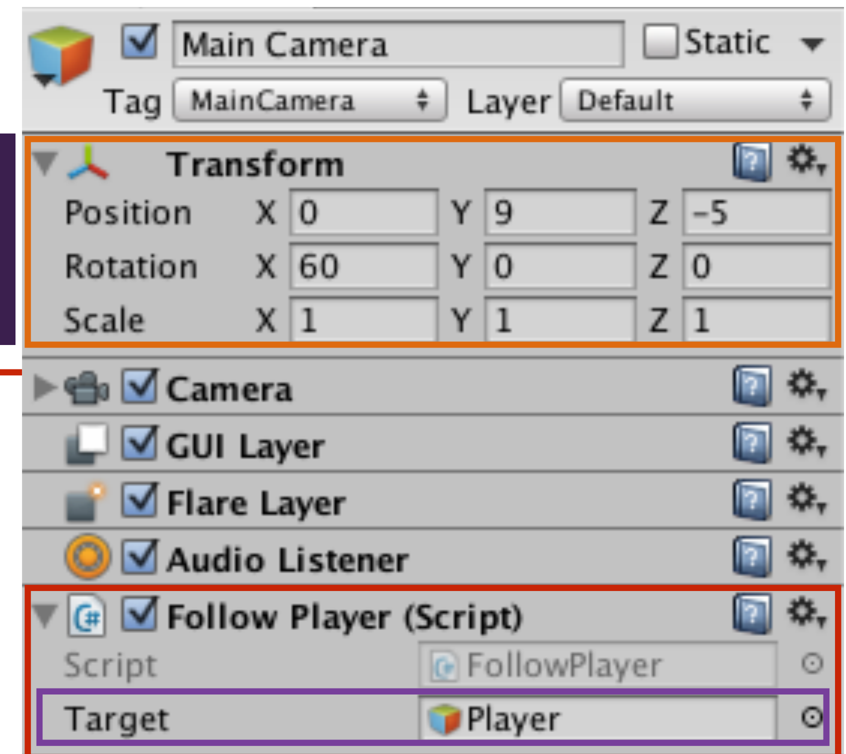


実行すると, 上下左右でPlayerが動きます. このときのスピードは, Inspectorビューの変数Speedの項目を直接変えることで調整できます.

Playerを追跡する

- Main Cameraに対して, 以下のスクリプトを追加して, 最初の位置関係を保ったまま, Playerを追跡します.

初期状態で, Main CameraがPlayerを斜め上から俯瞰できるように, Transformを以下のように変更します.

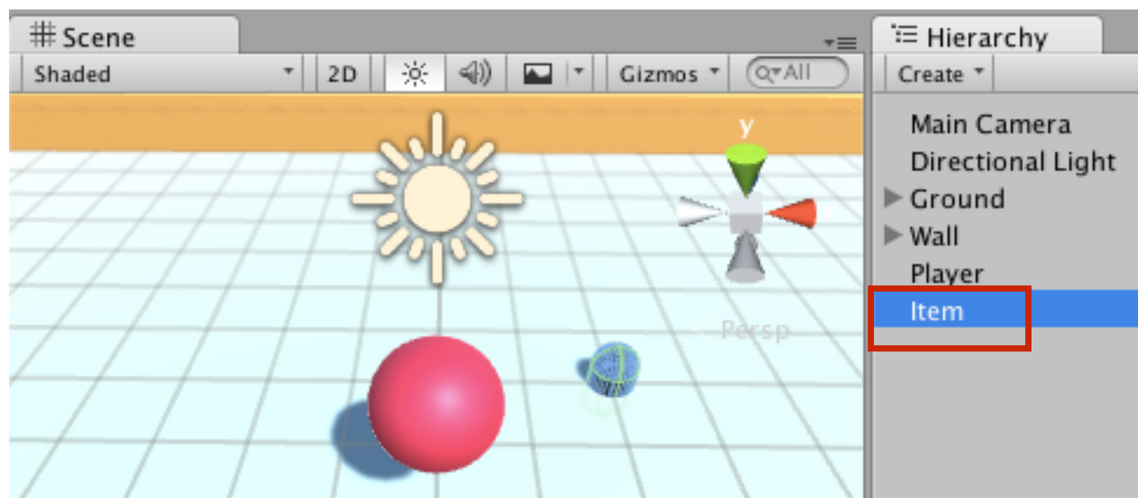


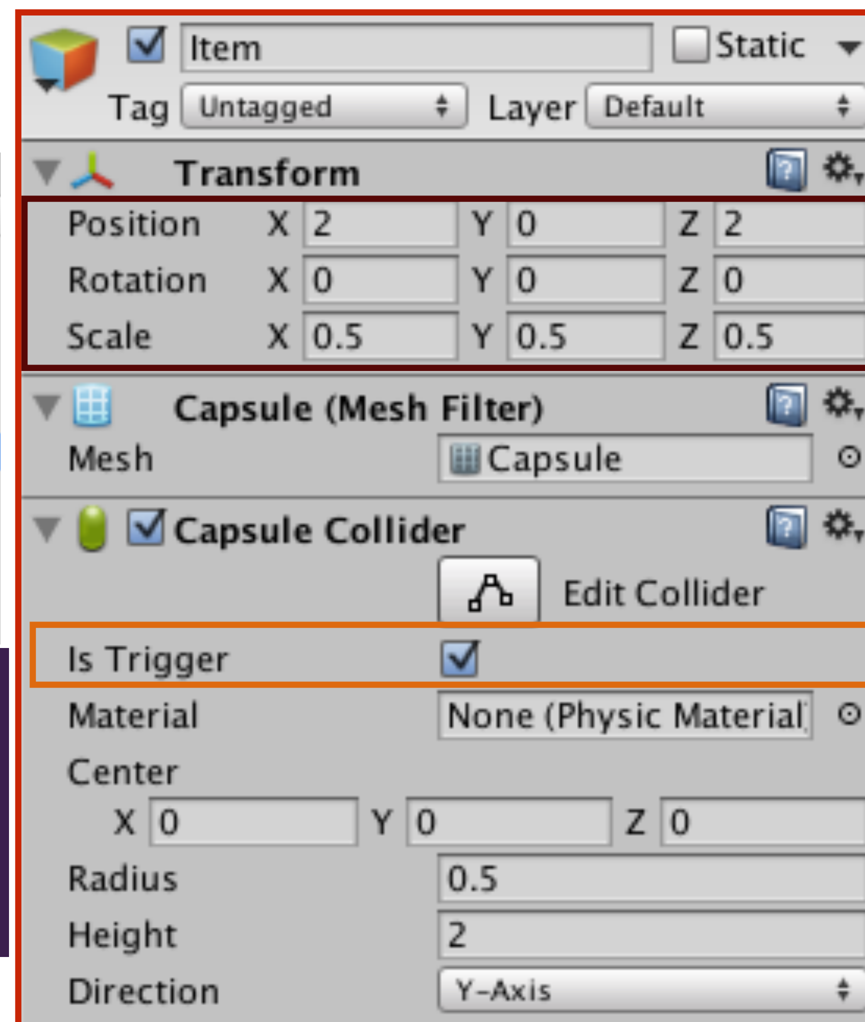

```
using UnityEngine;
using System.Collections;

public class FollowPlayer : MonoBehaviour {
    public GameObject target = null; //Player
    private Vector3 offset;     // 相対座標

    void Start () {
        //自分自身とtargetとの相対距離を求める
        offset = transform.position - target.transform.position;
    }
    void Update () {
        // 自分の座標にtargetの座標を代入する
        transform.position = target.transform.position + offset;
    }
}
```

Targetの項目には, ヒエラルキービューのPlayerをドラッグ&ドロップして指定します.

Itemの追加・衝突の検知

- Hierarchyビューに, 3D Objectの一つであるCapsuleを追加し, Itemとリネームしてください. Playerの近くに適当に置きます.

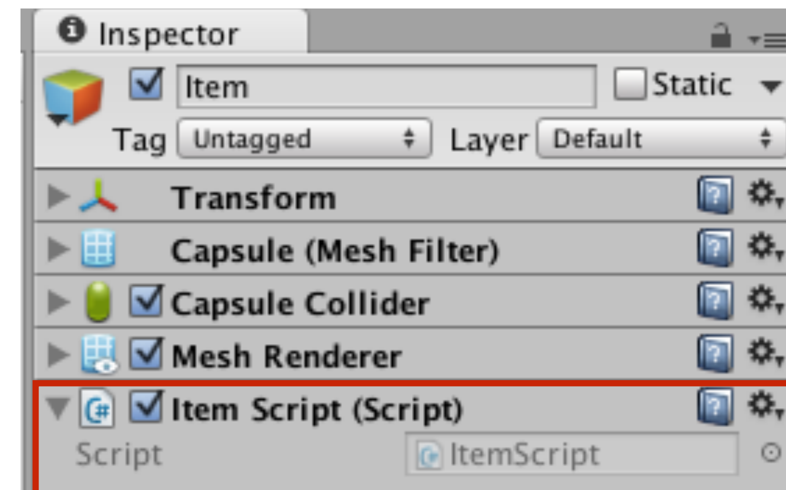




Colliderコンポーネントの, 「Is Trigger」にチェックを入れることで, 他のオブジェクトと空間的に重なりを持つときに, イベントを発生させることができます.

Itemに触れたゲームオブジェクトを消す！！

- ゲームオブジェクト「Item」に対して，「ItemScript.cs」という名前のスクリプトを追加し，自分に接触したゲームオブジェクトの表示を消すコードを埋め込みます．



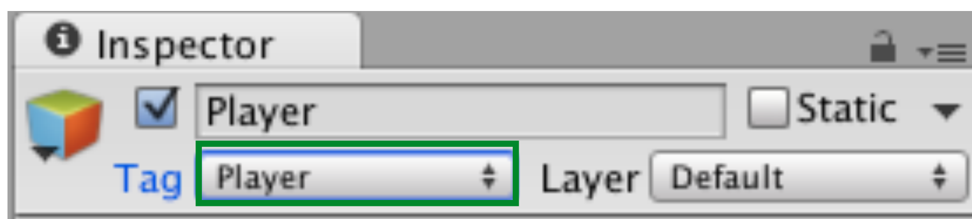

```
public class ItemScript : MonoBehaviour {
    void Start () {
    }
    void Update () {
    }

    // トリガーとの接触時に呼ばれるコールバック
    void OnTriggerEnter (Collider hit)
    {
        // 接触対象はPlayerタグですか？
        if (hit.CompareTag ("Player")) {
            // このコンポーネントを持つGameObjectを破棄する
            Destroy(gameObject);
        }
    }
}
```

ItemScript.cs

Collider

`boolean CompareTag(string tag)`

接触したオブジェクトのタグが引数と同一の時に限ってtrueを返す．



Object

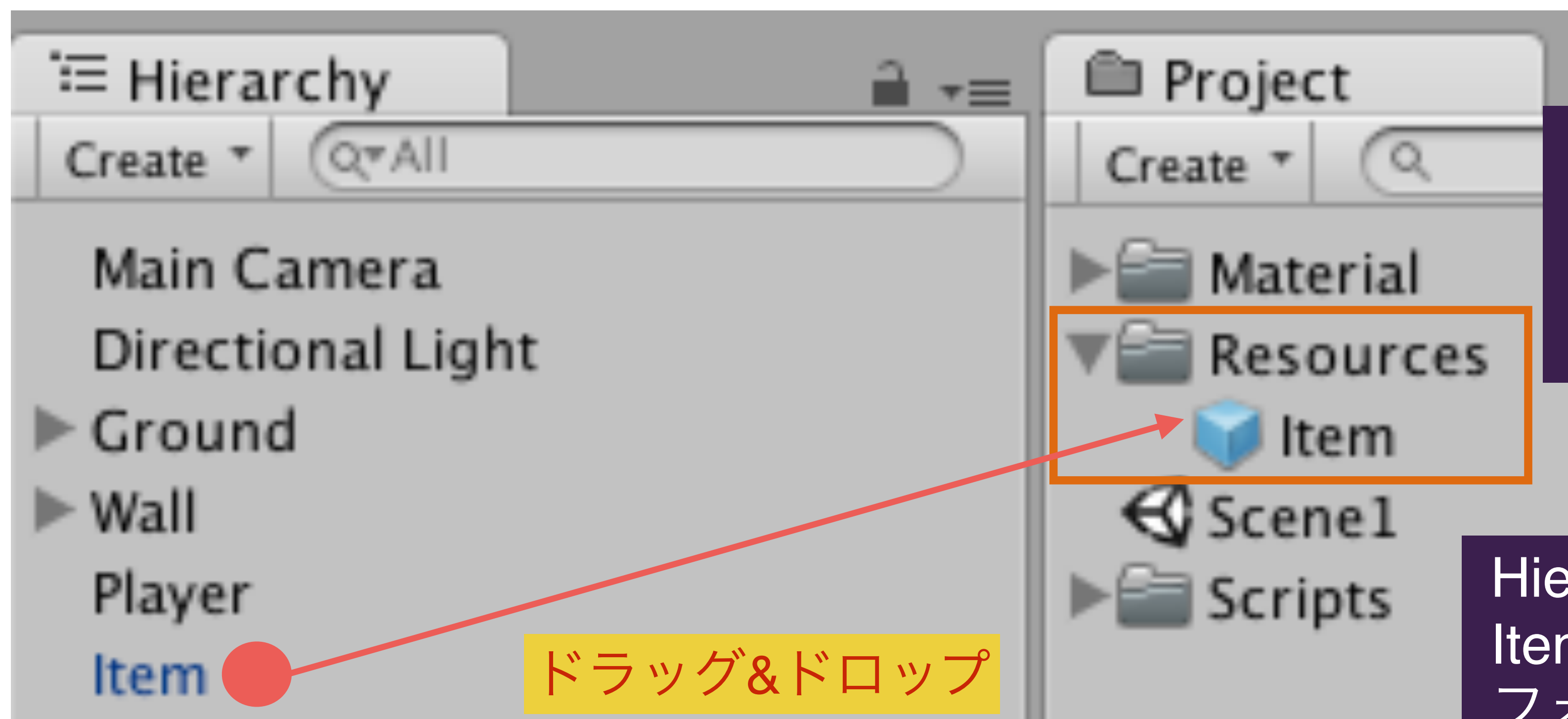
`void *Destroy(Object obj)`

引数のオブジェクトを廃棄します（結果，シーンから見えなくなります）．

Inspectorビューより，PlayerのTagにPlayerを指定することで，衝突したobject のうち，「Player」タグを保持するもののみが消失します．

プレハブの作成方法

- 一度作成したゲームオブジェクト「Item」のPrehab（ゲームオブジェクトの雛形）を作成します.
- ゲームオブジェクトを Prehab とすることで，Prehab と同様のパラメータを持つゲームオブジェクトのインスタンス（Prehab のcloneと呼ばれます）を容易に生成することができます.



1 Projectブラウザで，「Resources」という名前のフォルダをつくります.

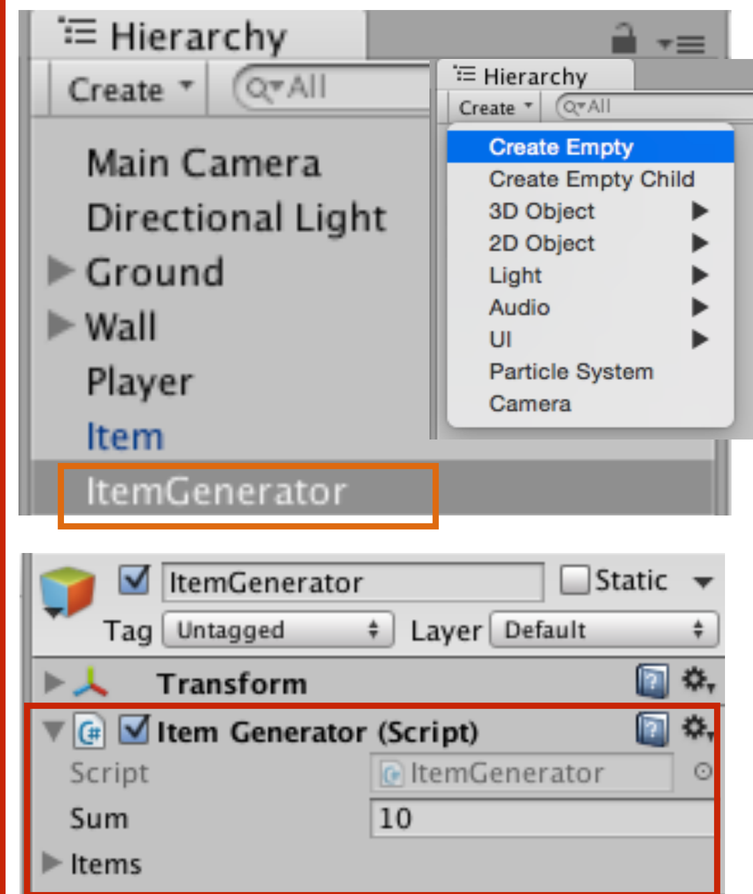
2 Hierarchyビューから，Itemを「Resources」フォルダの中にドラッグ&ドロップします.

Instantiate関数でプレハブを大量生産する

```
public class ItemGenerator : MonoBehaviour {

    public int sum = 10; //生成するItemの数
    public GameObject[] items;  //参照用のItem配列

    // Use this for initialization
    void Start () {
        items = new GameObject[sum];

        for (int i = 0; i < sum; i++) {
            //Resourcesフォルダの中に入っている, Itemという名前のプレハブから
            //ゲームオブジェクトのインスタンスを生成して, items[i]とする.
            items [i] = (GameObject)Resources.Load ("Item");

            //初期位置を, 中心からの距離5の円周上でランダムに決定します.
            float randRot = 360.0f * Random.value;
            float rx = 5.0f * Mathf.Cos (randRot);
            float rz = 5.0f * Mathf.Sin (randRot);
            float ry = 0.0f;
            Vector3 ipos = new Vector3 (rx, ry, rz);
            //初期角度（回転はゼロ）
            Quaternion irot =
                Quaternion.Euler (new Vector3 (0.0f, 0.0f, 0.0f));

            //items[i]を, 初期位置=ipos・初期角=irotで, シーンに配置します.
            Instantiate (items[i], ipos, irot);
        }
    }

    // Update is called once per frame
    void Update () {

    }
}
```

ItemGenerator.cs

空のゲームオブジェクト「ItemGenerator」を作成し, その中のコンポーネントとして, 左図の「ItemGenerator.cs」を追加します.



ItemGenerator.cs

Resources

`Object *Load(string path)`

Resourcesフォルダの中に入っている, 引数のパスに位置するプレハブをロードして, インスタンスをつくります.

```
//Resourcesフォルダの中に入っている, Itemという名前のプレハブから
//ゲームオブジェクトのインスタンスを生成して, items[i]とする.
items [i] = (GameObject)Resources.Load ("Item");

//初期位置を, 中心からの距離5の円周上でランダムに決定します.
float randRot = 360.0f * Random.value;
```

Quaternion

`Quaternion *Euler(Vector3 rot)`

引数のオイラー角（Vector3型：三次元）を四次元のQuarternion型に変換する.

```
Quaternion irot =
    Quaternion.Euler (new Vector3 (0.0f, 0.0f, 0.0f));

//items[i]を, 初期位置−ipos・初期角−irotで, シーンに配置します.
Instantiate (items[i], ipos, irot);
```

Object

`void *Instantiate(Object obj)`

`void *Instantiate(Object obj, Vector3 pos, Quaternion rot)`

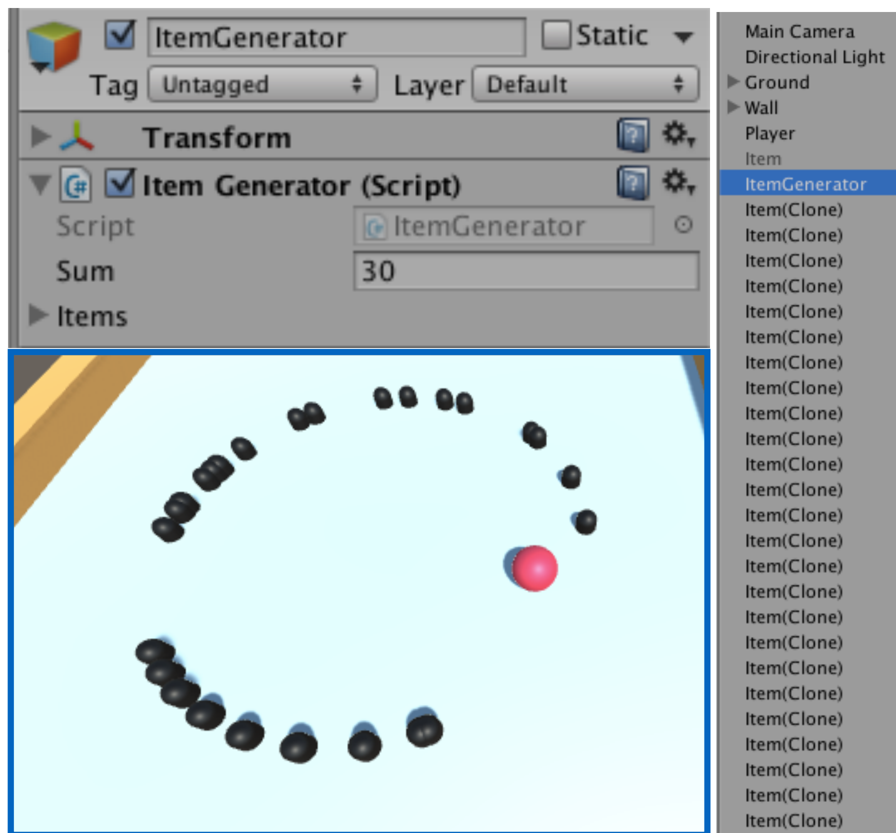
gameobjectのインスタンスを生成. 引数で初期位置を指定可能.

Prehabは, オブジェクト指向におけるクラスのようなものと考えればよいです. クラス（オブジェクトの設計図）から, コンストラクタによって, オブジェクトのインスタンスが生成されるように, Unityのゲームオブジェクトのインスタンスは, PrehabからInstantiateにより生成されます.

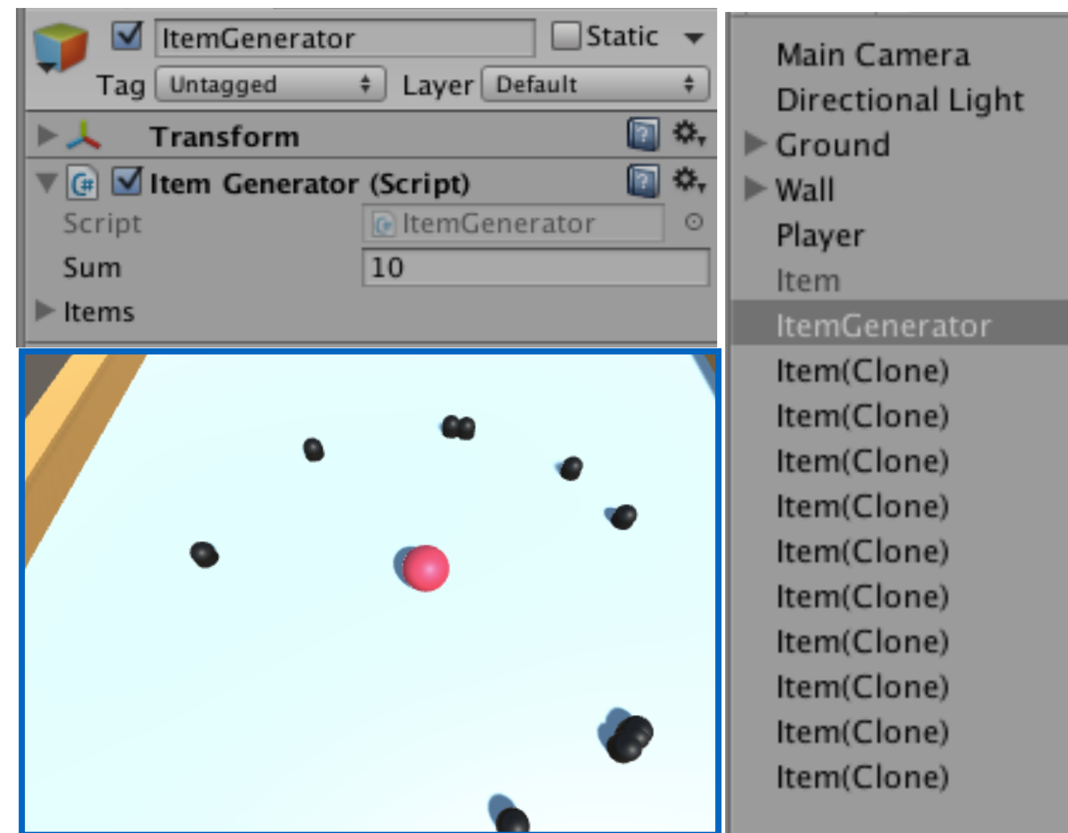
Instantiate関数でプレハブを大量生産する（結果）

- ItemGenerator の変数「Sum」の値を変更することで, 実行時に生成する Item の数を決定することができます.

実行結果（sum=30）



実行結果（sum=10）



Hierarchyビューに, 生成したインスタンスの数に応じて, プレハブのクローンのリストが表示されていることに注意して下さい.