演 習 １ ： U N I T Y の 基 礎

(01) 04/22

１Ａ｜Unityとエディタの連携

(02) 04/29

１Ｂ｜Transform・キーイベント・マウスイベント

(03) 05/06

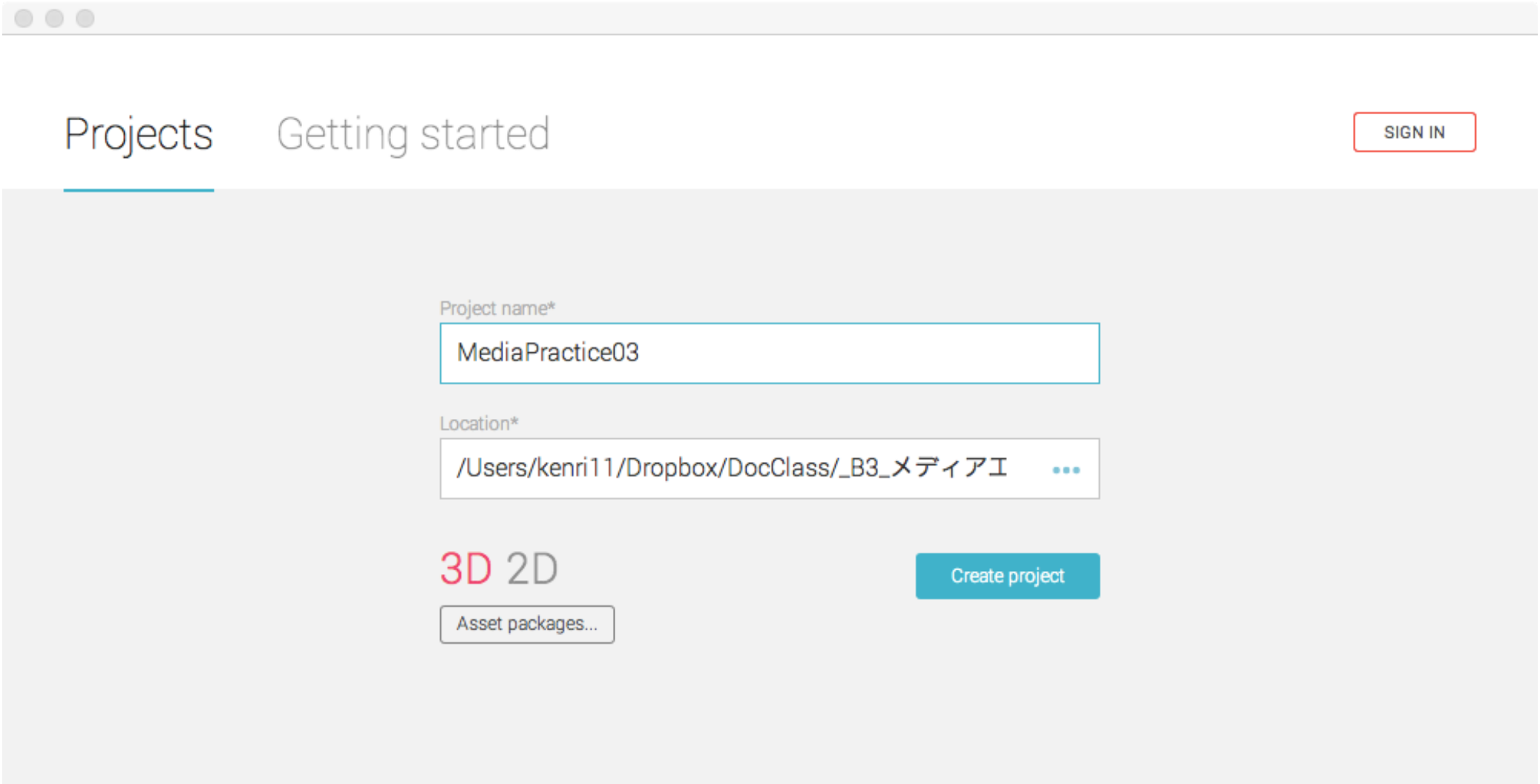
１Ｃ｜剛体特性・カメラの視点

(04) 05/13

１Ｄ｜プレハブ（gameobjectの雛形）, タグ, その他

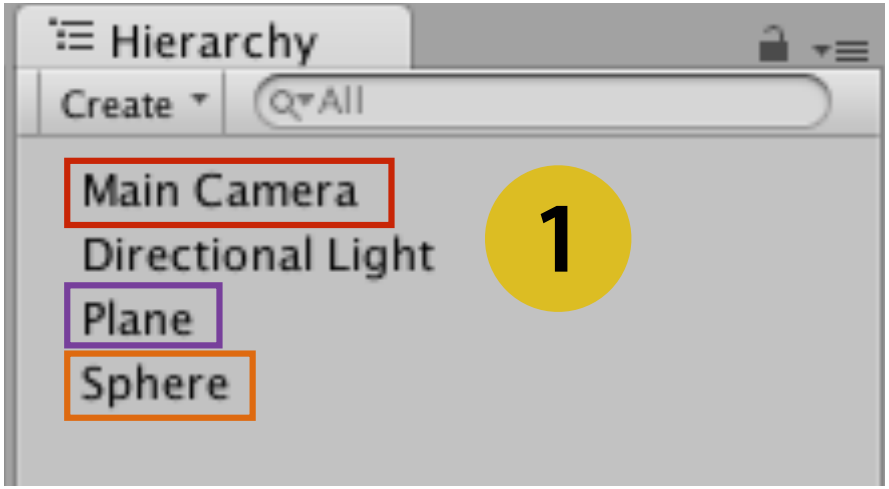
MediaPractice03

剛体特性（ゲームエンジン）・カメラの視点

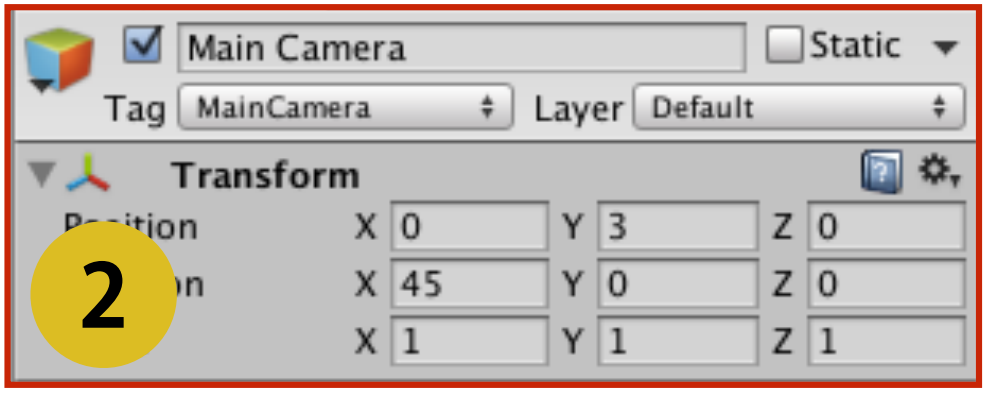


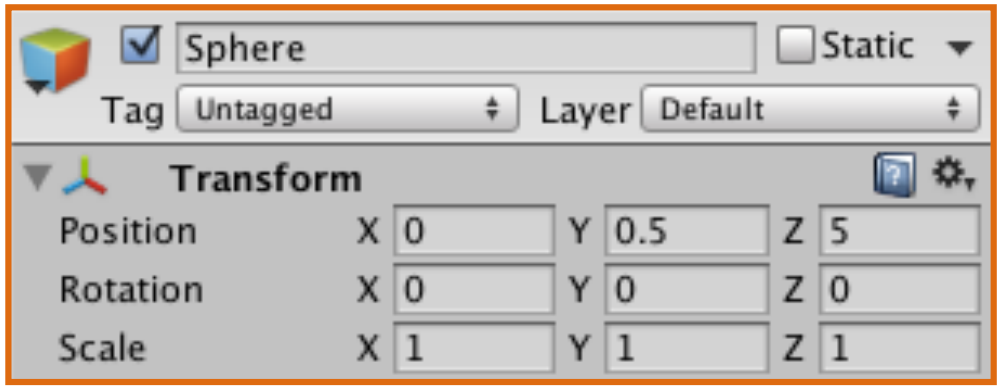
準備

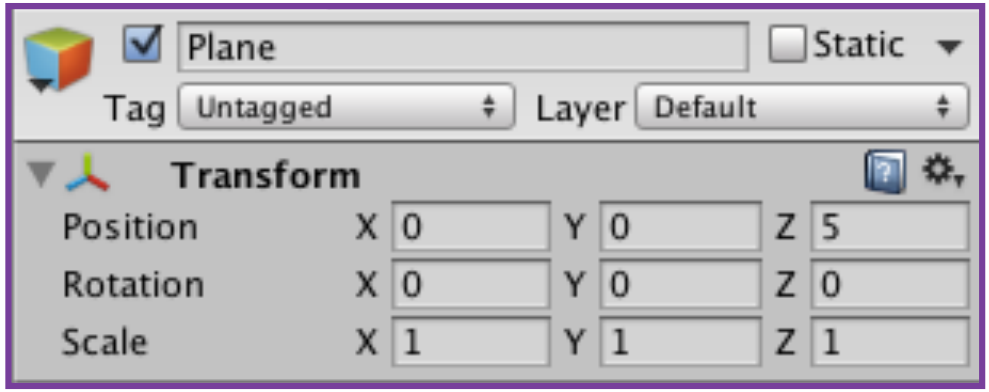
デフォルトで用意されているメインカメラとライトに加えて，新たにPlane（地面）とSphere（球）をHierarchyビューに追加します（適当に名前を変えてもらって結構です）．



各ゲームオブジェクトのtransformを以下のように変更してください．

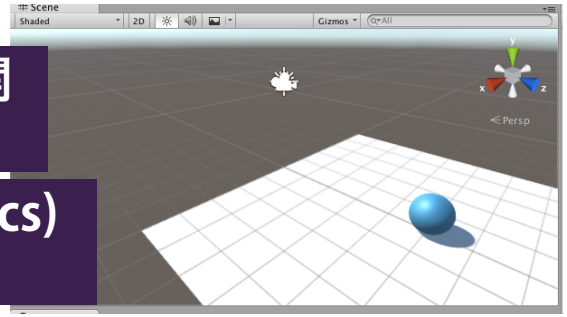


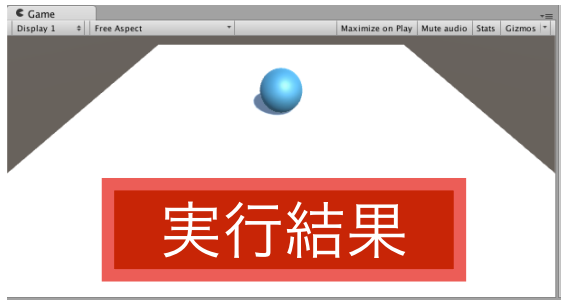




3 Sphereに適当な色のmaterialを関連付けてください（p162）．

4 Sphereにスクリプト (rigidscript.cs) を関連付けてください（p165）．



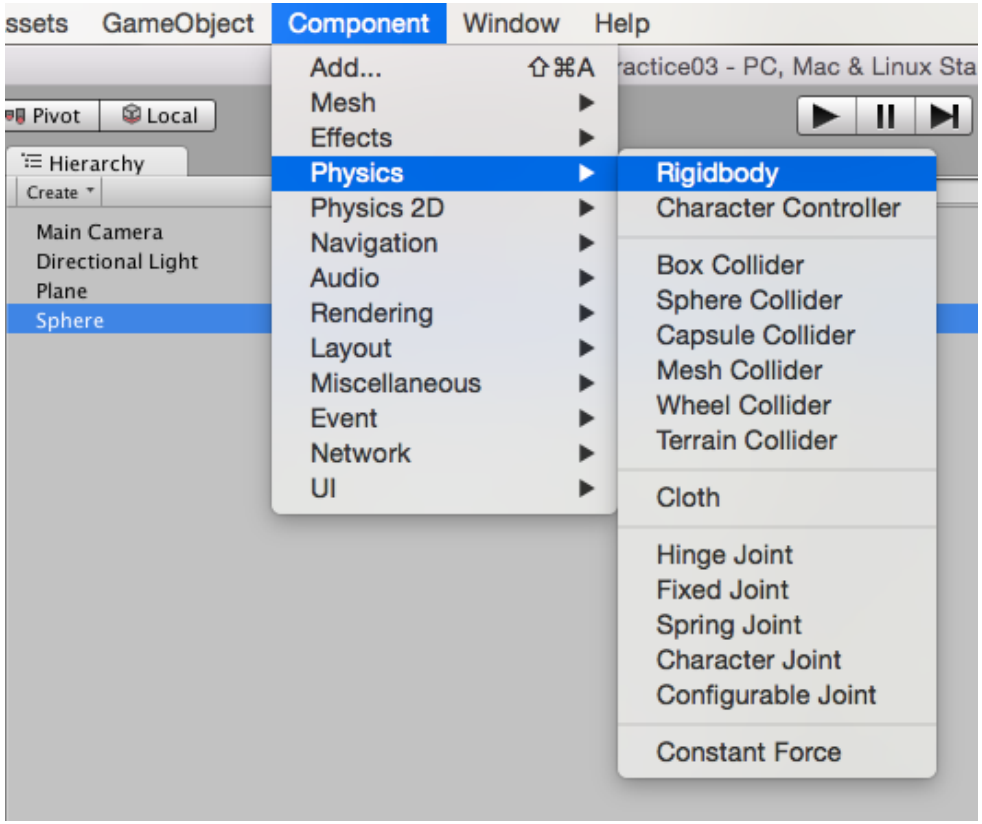


剛体特性の組み込み

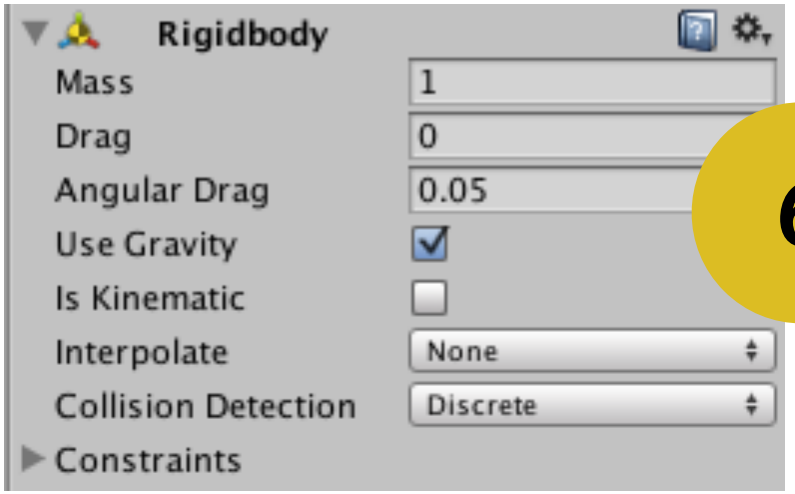
□ 特定のゲームオブジェクトに剛体特性を組み込むと, そのオブジェクトは, 外部から物理的な力（衝突・摩擦）を受けるようになります.

5

Hierarchyビューで, Sphereを選択した状態で, メニューより, <Component>→<Physics>→<Rigidbody>を選びます.



Rigidbodyのパラメータは初期状態のままでOKです.



6

Rigidbody	
Mass	質量
Drag	空気抵抗
Use Gravity	重力のON/OFF
Constraints	特定の軸の移動・回転を凍結します.

剛体に外力を加える

□ gameobjectの（事後に加えられた）コンポーネントに関するobjectを取り出す場合, 以下の書式を使います.

GameObject

Class GetComponent<classname>()

Rigidbody GetComponent<Rigidbody>()

ゲームオブジェクトのコンポーネントのうち, classnameに対応するクラスに対応するオブジェクトを取り出すためのメソッド.

Rigidbody

void AddForce (float x, float y, float z)

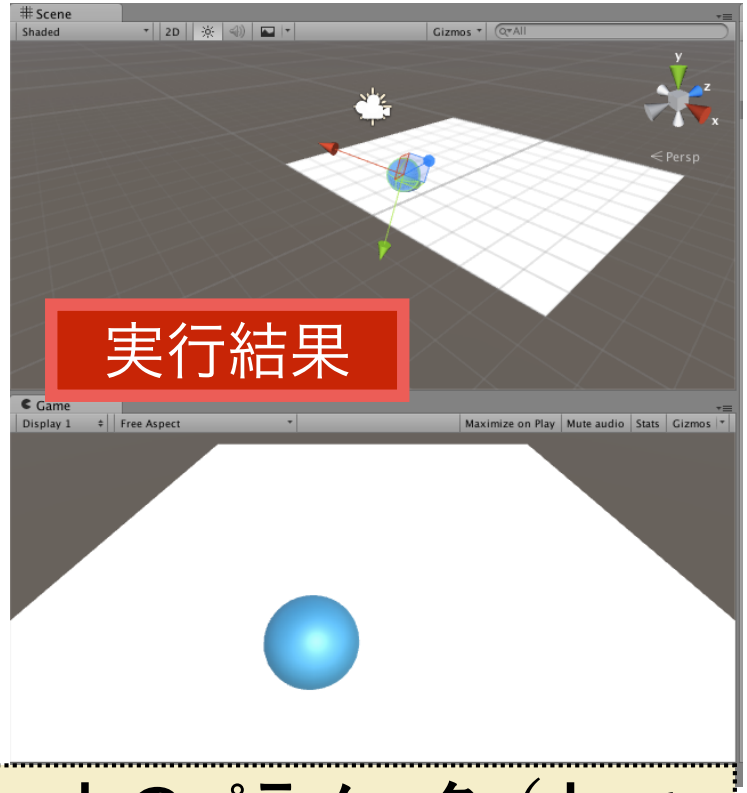
剛体に対して, 3つの軸に対応する外力を加える.

rigidscript.cs

```
public class rigidscript : MonoBehaviour {

    void Start () {
    }

    void Update () {
        if (Input.GetKey (KeyCode.UpArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (0, 0, 1);
        }
        if (Input.GetKey (KeyCode.DownArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (0, 0, -1);
        }
        if (Input.GetKey (KeyCode.RightArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (1, 0, 0);
        }
        if (Input.GetKey (KeyCode.LeftArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (-1, 0, 0);
        }
    }
}
```



上下左右で, 球に力が加わります. rigidbodyのパラメータ（drug, useGravity）を変化させたときの挙動を確認してください.

カメラの追従１

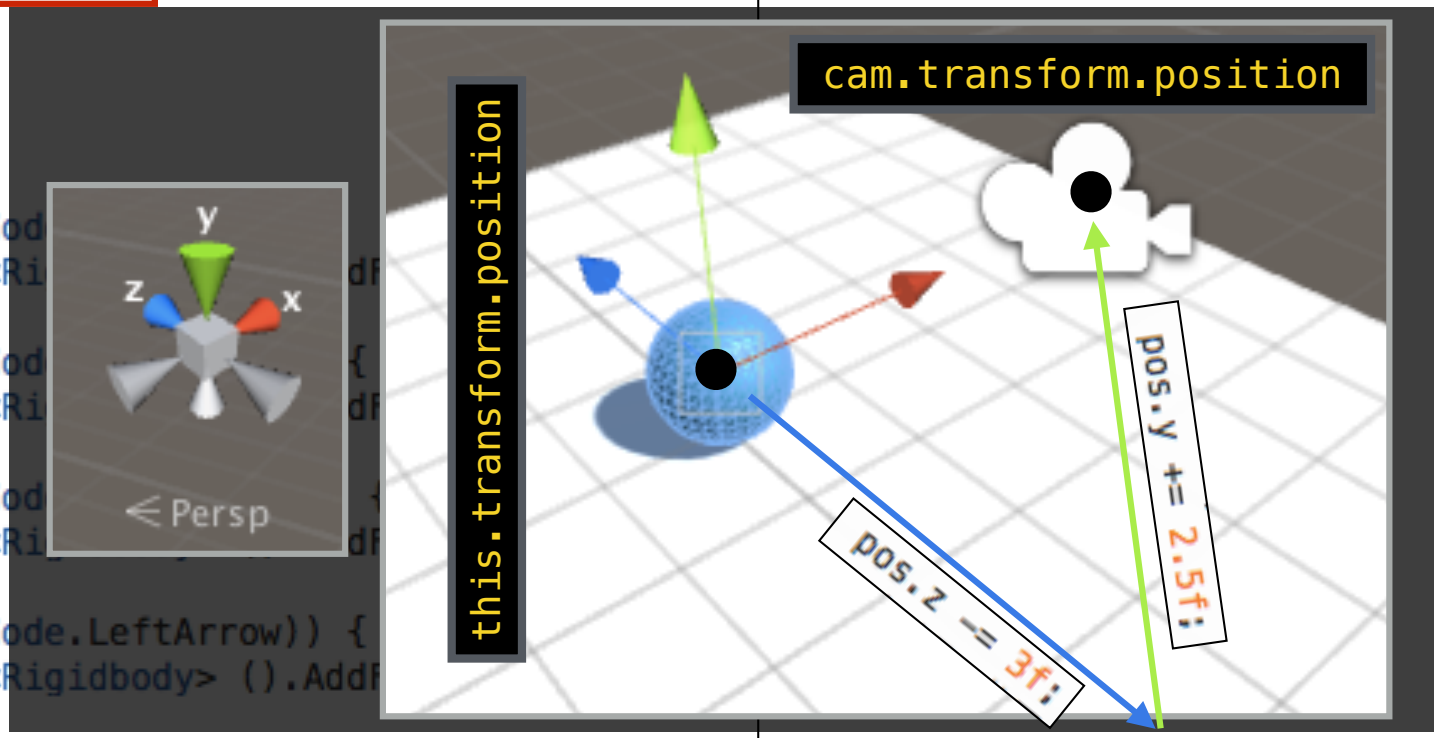
□ 赤枠の部分をrigidscript.csに追記してください.

```
public class rigidscript : MonoBehaviour {
    //メインカメラ（GUI上で直接関連付ける）
    public GameObject cam = null;

    void Start () {
    }

    void Update () {
        if (Input.GetKey (KeyCod
            this.GetComponent<Ri
        }
        if (Input.GetKey (KeyCod
            this.GetComponent<Ri
        }
        if (Input.GetKey (KeyCod
            this.GetComponent<Ri
        }
        if (Input.GetKey (KeyCode.LeftArrow)) {
            this.GetComponent<Rigidbody> ().AddF
        }
        //Sphereの現在の位置
        Vector3 pos = this.transform.position;
        pos.y += 2.5f; //2.5上方
        pos.z -= 3f;   //3.0後方
        //カメラの位置をSphereの斜め上後方とする.
        cam.transform.position = pos;
    }
}
```

メインカメラとSphereの位置関係



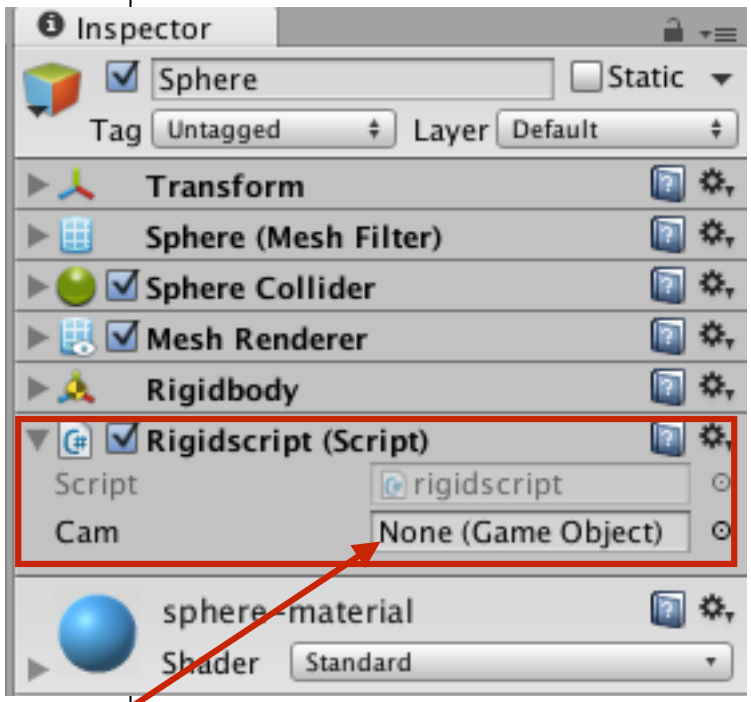
rigidscript.cs

カメラの追従１

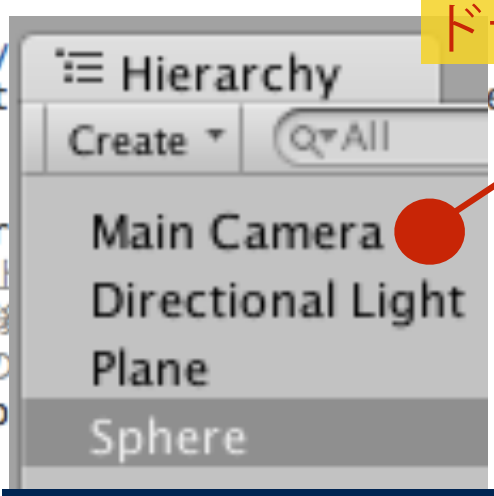
```
public class rigidscript : MonoBehaviour {
    //メインカメラ（GUI上で直接関連付ける）
    public GameObject cam = null;
```

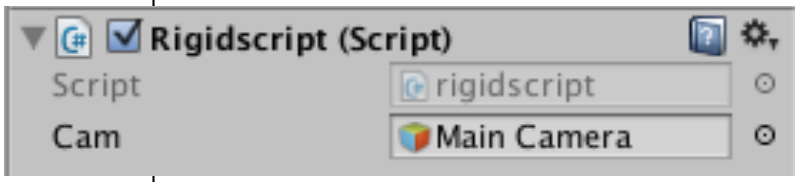
グローバル変数に, 内容が未定のゲームオブジェクトを定義しておきます. この時点ではまだ, Rigidscriptの Camには何も関連付けられていないことに注意してください「None（Game Object）」.

```
            this.GetComponent<Rigidbody> ().AddForce (0, 0,
        }
        if (Input.GetKey (KeyCode.DownArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (0, 0,
        }
        if (Input.GetKey (KeyCode.RightArrow)) {
            this.GetComponent<Rigidbody> ().AddForce (1, 0,
        }
        if (Input.GetKey (Key
            this.GetComponent                    e (-1, 0, 0);
        }
        //Sphereの現在の位置
        Vector3 pos = this.tr
        pos.y += 2.5f; //2.5上
        pos.z -= 3f;   //3.0後
        //カメラの位置をSphereの
        cam.transform.positio
    }
}
```



ドラッグ&ドロップ



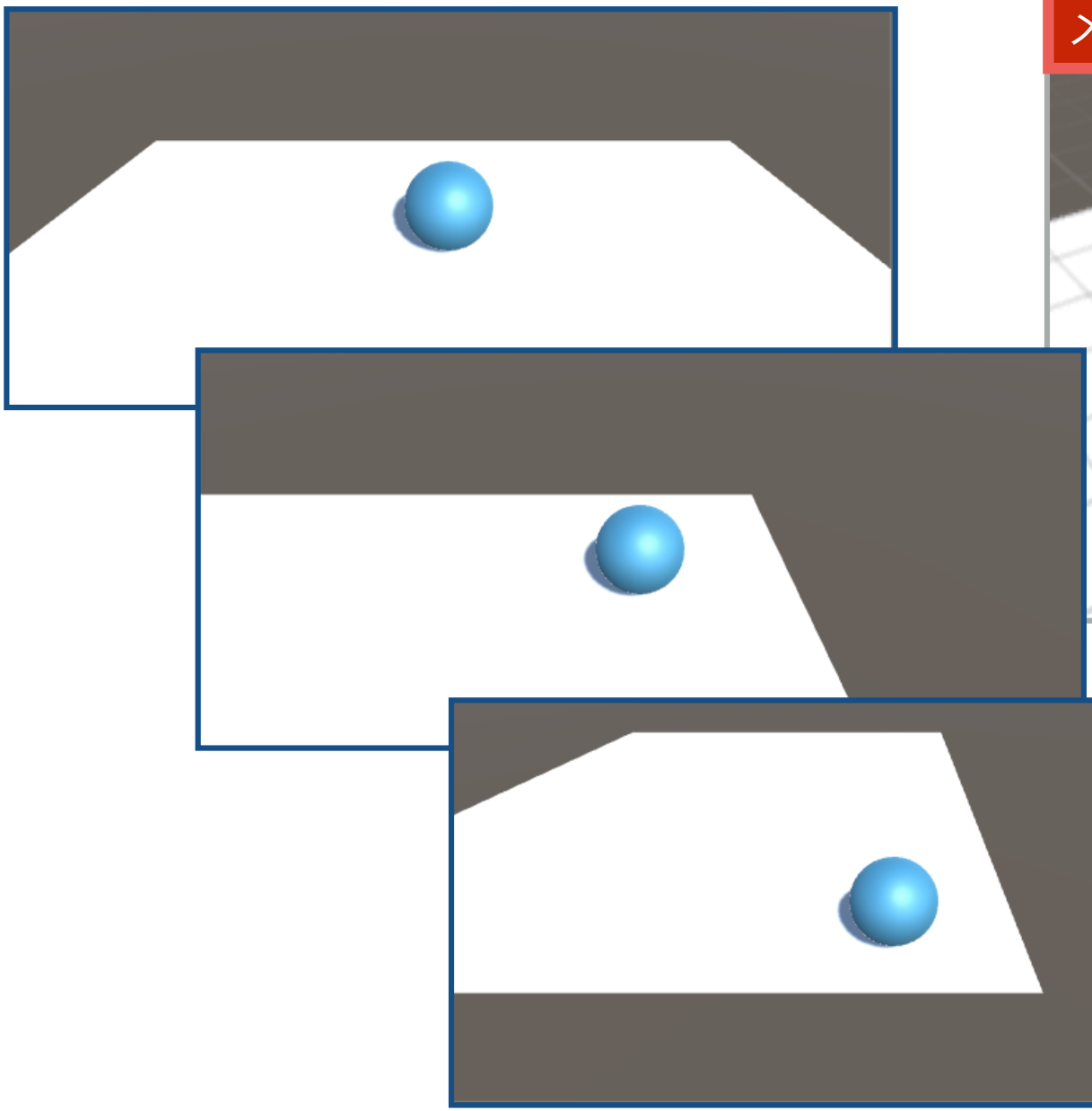


Sphereを選択した状態で, Camの領域に, Main Cameraを直接ドラッグ&ドロップすることによって, rigidscript.cs内のcamとメインカメラが正しく対応付けられます. スクリプトから名前で関連づけて読み込む場合は, Findを使います.

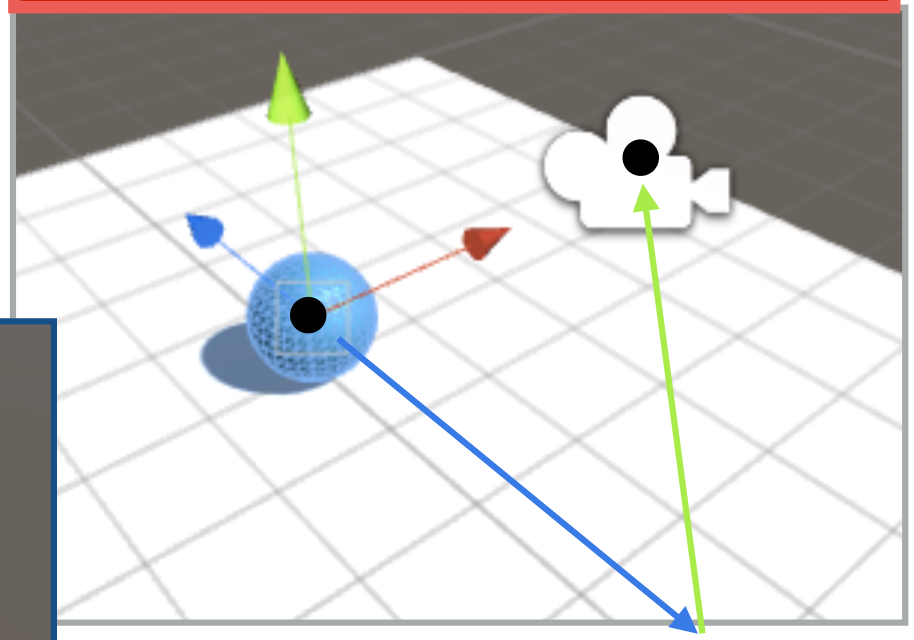
rigidscript.cs

カメラの追従１

実行結果

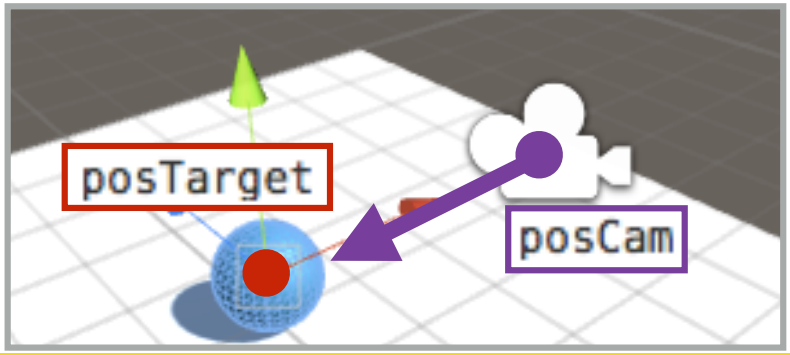


メインカメラとSphereの位置関係

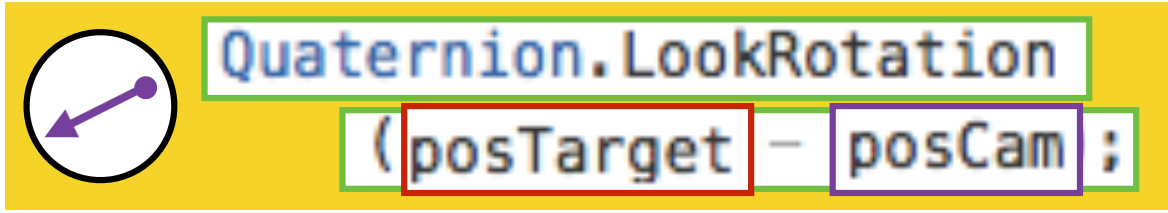


カメラの追従２（LookRotation）

- rigidscript.cs の Update関数の後半部分を以下のように修正すると, カメラの位置は保ったまま, 常にボールの中心へとカメラの視点を向けるようになります.

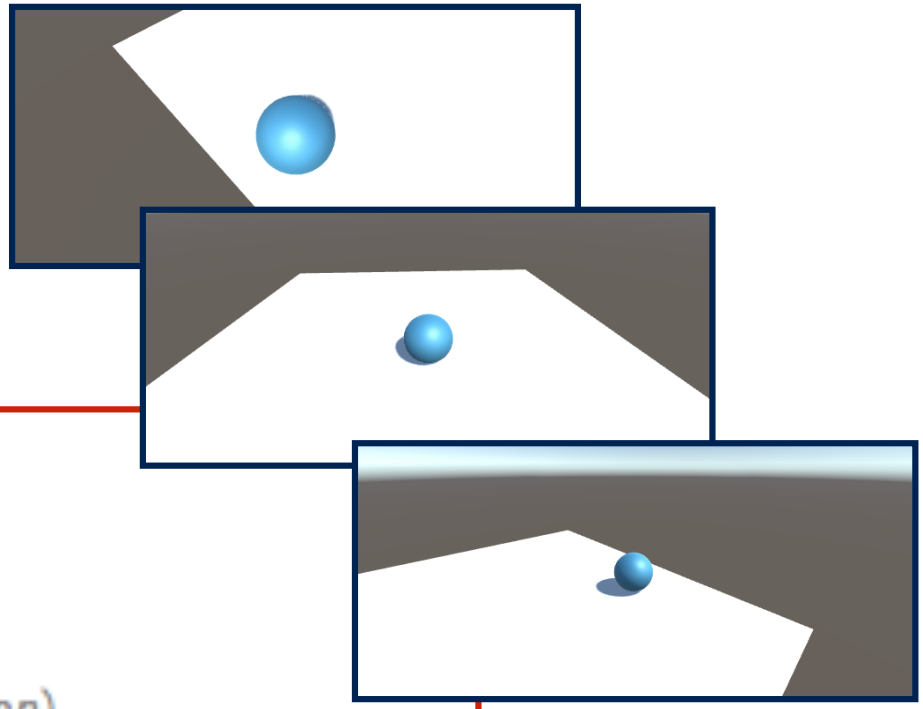



```
//Sphereの現在の位置
Vector3 pos = this.transform.position;
pos.y += 2.5f; //2.5上方
pos.z -= 3f;   //3.0後方
//カメラの位置をSphereの斜め上後方とする.
cam.transform.position = pos;
```



Quaternion ***LookRotation**(Vector3 posTarget – Vector3 pos)

pos から posTargetを向いた時の角度を（Quaternion型）で取得.



```
//Sphereの現在の位置
Vector3 posTarget = this.transform.position;
//Cameraの位置
Vector3 posCam = cam.transform.position;

//posCamからposTargetを向いた時の角度（Quaternion）
Quaternion rot = Quaternion.LookRotation (posTarget - posCam);
//カメラの角度（rotation）をrotにセットする.
cam.transform.rotation = rot;
```

rigidscript.cs

実行結果

小課題

- rigidscript.cs をさらに修正し（あるいは新たにスクリプトをつくって）, カメラが球の周りを回転しながら（回転の仕方は自由）, 球への視点を維持するようにしてください. さらに, 上下の矢印キーで, 球への距離を操作できるようにしましょう.