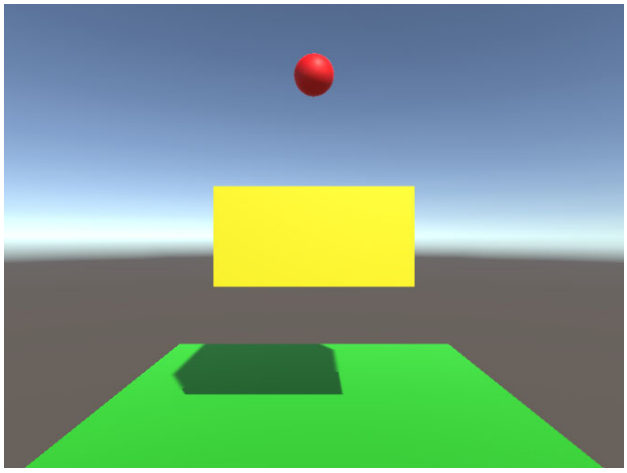
PhysX（フィジックス）

ゲームエンジンに内蔵されている物理演算機能を用いれば、複雑な衝突や自然現象のシミュレーションなどが容易に再現できます。Unity は物理演算を実現する為に、数あるエンジンの中から NVIDIA 社の PhysX（フィジックス）4.1 を実装しています。

ここでは、ゲームには必須の考え方である「接触判定」を確実に理解する演習に取り組みます。

また、直接、物理演算に関連した項目ではありませんが、ゲームでよく用いられる概念の「リスポーン機能」についても紹介します。



この例では、赤いボールは演算対象の仲間入りを指示しなければ、物理演算が行われず、落下する事すらできません。逆に黄色い箱には落下の指示は出しません。

赤いボールが落下できるようになると、物理判定で接触を検出する際に、黄色い箱には侵入を行い、緑の床では反射の反応を示します。赤いボールは黄色い箱をすり抜けて落下し、緑の床で何度かバウンドします。

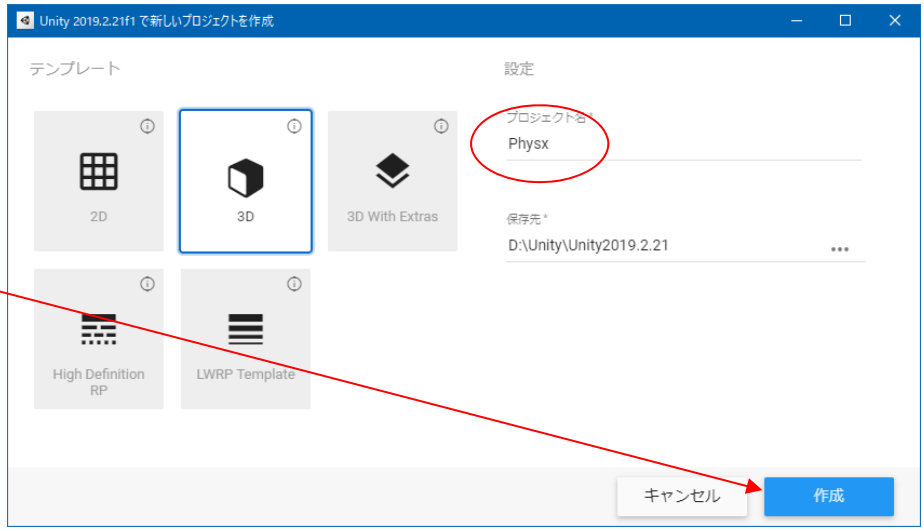
こういった挙動を Unity で設定し、その接触をプログラムから検出し、様々な命令を実行します。

シーンの準備

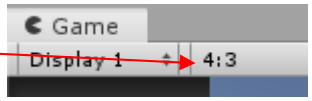
【STEP1】 新規プロジェクトの作成

● Unity を起動し、新規プロジェクト Physx を作成します。

ボタン作成を押下します。

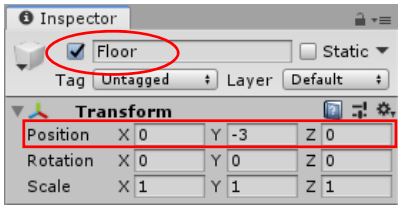


● Game 画面の比率を 4:3 にします。

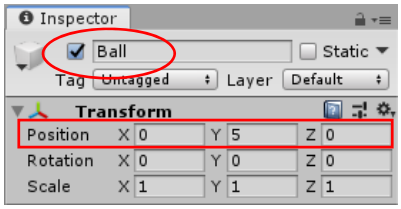


【STEP2】シーンを準備する

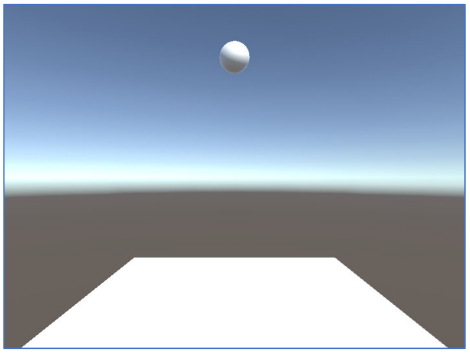
- ヒエラルキー欄の Create（クリエイト）から 3D Object > Plane（プレーン）を選択します。名称を Floor にします。
  この平面オブジェクトを選択している状態で、インスペクタでパラメータを設定します。



- 同様にして Create（クリエイト）から 3D Object > Sphere（スフィア）を選択します。名称を Ball にします。
  インスペクタでパラメータを設定します。

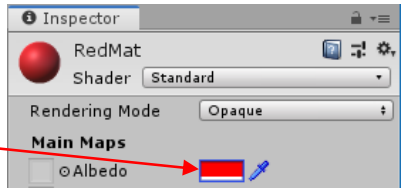


こんな感じになります。

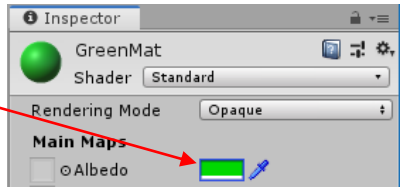


色が無くて判りにくいので、質感（色だけですが）を作成して割り当てていきます。

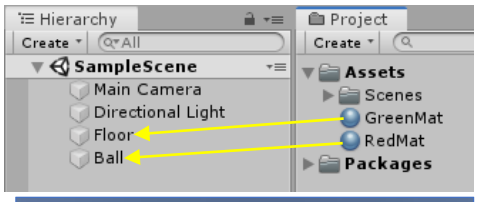
- プロジェクト欄の Create から Material（マテリアル）を選択し、名称を RedMat とします。
  インスペクタで Albedo（アルベド）のカラーピッカーをクリックして、赤色を設定しておきます。



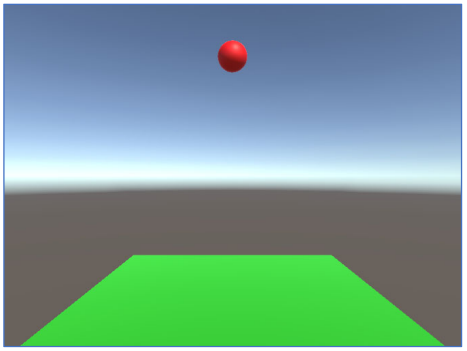
- 同様にして、新しいマテリアル GreenMat を作成し、緑色を割り当てておきます。



- 作成したマテリアルをマウスでドラッグ＆ドロップする方法で、それぞれのオブジェクトに取り付けます。



こんな感じになります。

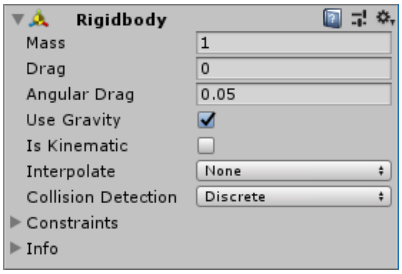


物理演算機能の利用

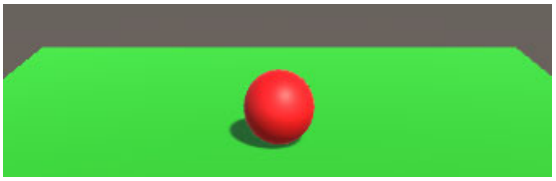
【STEP3】 物理演算の影響下に入る

- ここで、プレイボタンを押下してみます。
  
  ボールが落下しないことを確認します。
  ボールに対して、物理演算の仲間入りをする能力を持たせていないからです。

- ヒエラルキー欄の Ball を選択し、インスペクタの Add Component を押下して、Physics > Rigidbody（リジッドボディ）を選択します。



- プレイボタンを押下します。
  
  今度は上手くボールが落下します。
  Rigidbody コンポーネントが物理演算の仲間入りを宣言していることが判ります。



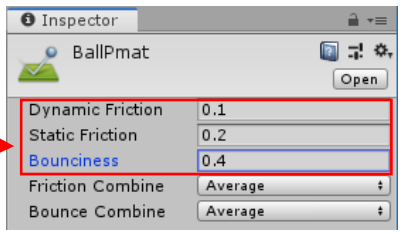
ここでの注意点は、Rigidbody が落下の能力を持たせた訳では無い！ということです。似ていますが、物理演算 PhysX が計算してくれる対象群に、加える目印を付けた、という感覚が近いです。

【STEP4】 物理マテリアル

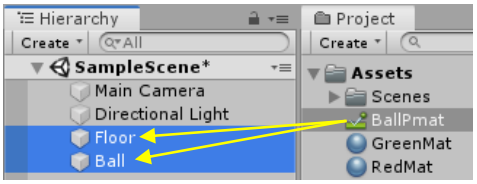
現状で物理演算のシミュレーションが始まったのですが、ボールが床から少しも跳ね返りません。これは、材質がゴムなのか？金属なのか？を表現する「物理マテリアル」を与えていないからです。

先の「マテリアル」とは少し異なる「物理マテリアル」を作成して与えます。

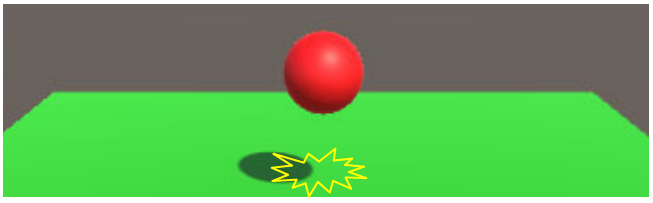
- プロジェクト欄の Create から Physic Material（フィジック マテリアル）を選択し、BallPmat と命名します。物理マテリアルです。
- インスペクタでパラメータを設定します。



- 作成した物理マテリアル BallPmat をマウスでドラッグ＆ドロップする方法で Floor、Ball の両方のオブジェクトに取り付けます。使い回しですね。



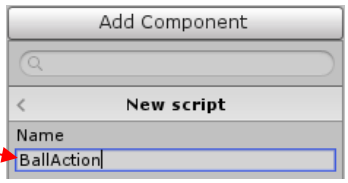
- プレイボタンを押下します。
  
  今度はボールが跳ね返ることが判ります。



【STEP5】 物体の接触(反射)を検出する

物体と物体の接触が目で見て理解できますが、これをプログラムで検出して他のイベントのきっかけに用いて行かなくてはなりません。この接触をプログラムで検出する方法に取り組みます。

- ヒエラルキー欄の Ball を選択し、インスペクタの Add Component から New script を選択し、名称を BallAction とします。



- スクリプト BallAction を次のように編集します。

反射系は以下の3つがあります。

① 当たった瞬間を OnCollisionEnter で検出
(オン コリジョン エンター)

② 離れた瞬間を OnCollisionExit で検出
(オン コリジョン イグジット)

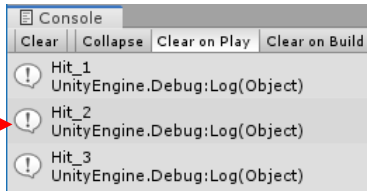
③ 接触中ずっとを OnCollisionStay で検出
(オン コリジョン ステイ)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallAction : MonoBehaviour {

    int HitCnt = 0; //ヒット回数

    void OnCollisionEnter(Collision other) {
        HitCnt++; //1加算
        Debug.Log("Hit_" + HitCnt);
    }

    void Start () {

    }
    void Update () {

    }
}
```

- プレイボタンを押下します。

画面下部の1行の欄に、小さくコンソールメッセージが出ることが判ります。

この行をクリックすると、大きく表示されます。ボールに接触現象が何度発生したかが検出できたことになります。

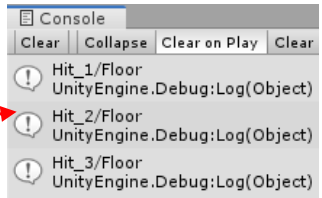


ところで、other(という名前の物体)が Collision 型として付加されていますが、あれは何でしょう?
Collision は衝突の意味で、この接触に関与した相手側の物体(ボールから見て床のこと)を指します。
Collision 型とは言え、元々はゲームオブジェクトですから、gameObject.name でたどると、ヒエラルキー欄での名称(ここでは Floor)を得られます。スクリプト BallAction を次のように編集し、実際に表示します。

```
//~前略~
    void OnCollisionEnter(Collision other) {
        HitCnt++; //1加算
        Debug.Log("Hit_" + HitCnt + "/" + other.gameObject.name);
    }
//~後略~
```

- プレイボタンを押下します。

ボールから見て、当たった相手の名前が表示されています。



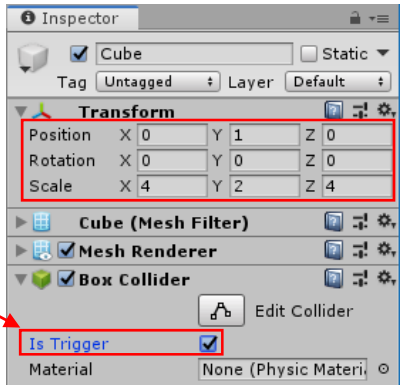
この「他のもの」。名称は何でもいいのですが、世界規模で other が使われており、それに準じます。

【STEP6】 侵入可能な物体を配置する

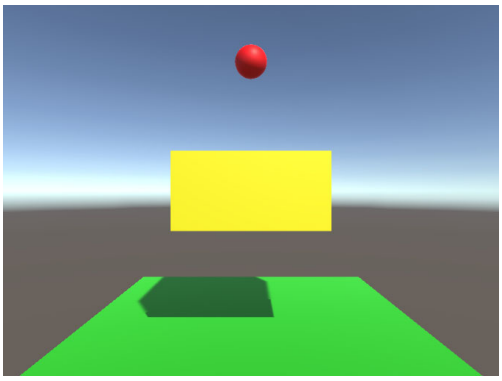
物体の接触後、2つの反応に分岐して物理演算が行われます。一つは先ほど動作確認した「反射」と、もう一つは「侵入」です。ボールが侵入するオブジェクトを用意して、それを検出してみます。

- ヒエラルキー欄の Create から 3D Object > Cube（キューブ）を選択します。（名前はこのままでもいいでしょう。）
  インスペクタでパラメータを設定します。

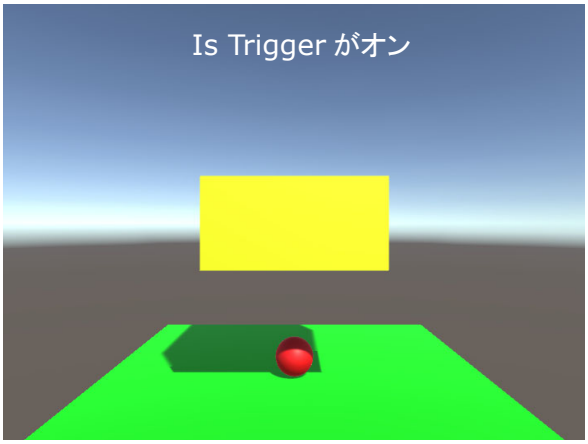
  今回は特別に、コンポーネント Box Collider の項目 Is Trigger をチェックします。
  これが反射と侵入を切り替えるパラメータとなります。

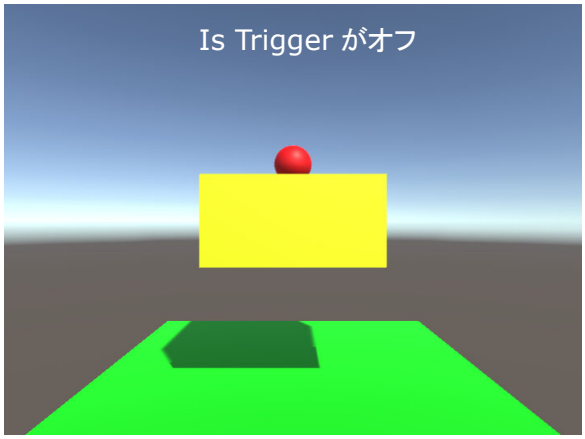


- 【命題】
  色のマテリアル YellowMat を作成し、黄色い質感に設定したら、この Cube に割り当てて下さい。



- プレイボタンを押下します。 
  ボールが黄色い箱を通過して落ちていくことが判ります。
  チェック項目 Is Trigger のオン／オフによって、反射と侵入が切り替わることを確認してみましょう。（最後にはオンにする。）



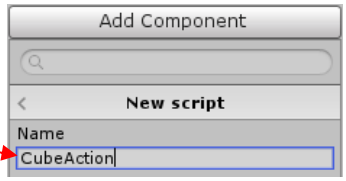


IsTrigger がオンで侵入系に変身するなら、直訳の意味は異なりますが、侵入系をオンにする操作であると覚えても構わなさそうです。

【STEP7】 物体の侵入を検出する

では、この黄色い箱に接触し、侵入していく様子をプログラムで検出してみます。

- ヒエラルキー欄の Cube を選択し、インスペクタの Add Component から New script を選択し、CubeAction と命名します。



- スクリプト CubeAction を次のように編集します。

侵入系は以下の3つがあります。

① 入った瞬間を OnTriggerEnter で検出
（オン トリガー エンター）

② 出て行った瞬間を OnTriggerExit で検出
（オン トリガー イグジット）

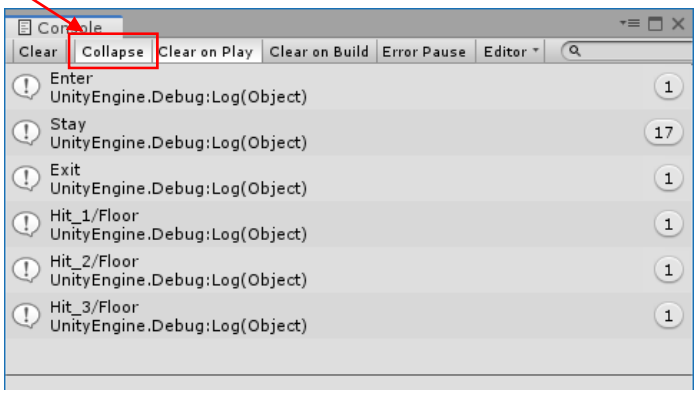
③ 侵入中の毎フレームを OnTriggerStay で検出
（オン トリガー ステイ）

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CubeAction : MonoBehaviour {

    void OnTriggerEnter(Collider other) {
        Debug.Log("Enter");
    }

    void OnTriggerStay(Collider other) {
        Debug.Log("Stay");
    }

    void OnTriggerExit(Collider other) {
        Debug.Log("Exit");
    }

    void Start () {

    }
    void Update () {

    }
}
```

- プレイボタンを押下します。

コンソールに多くのメッセージが出るので、Collapse を押下して、折りたたんで確認します。

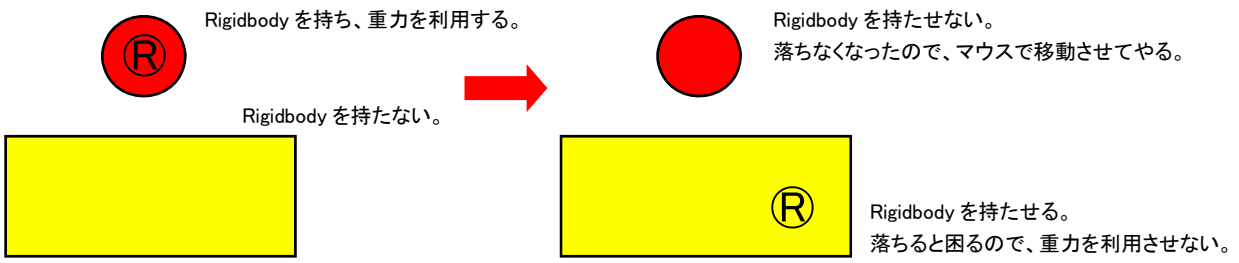
侵入が1回
滞在中が17回
脱出が1回
床に反射が3回 と、読み取れます。
17回になるかどうか？は動かしているPCの処理状態によって増減します。



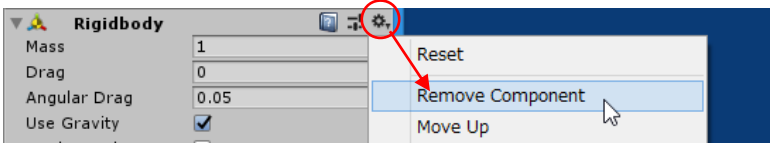
- おや？反射の Collision3兄弟は接触した相手を Collision 型とみなし other と称していましたが、侵入の Trigger3兄弟は、相手を同様に other と称していますが、Collider（コライダー）型とみなしています。
よく似ていて紛らわしいですが、侵入の場合は相手を形状で把握する必要がある為、衝突した点だけのデータでは不足するのです。
反射の Collision3兄弟 → 相手が Collision 型
侵入の Trigger3兄弟 → 相手が Collider 型（どんな形が侵入しているのか？が必要）

【STEP8】コンポーネント Rigidbody は誰が持つべき？

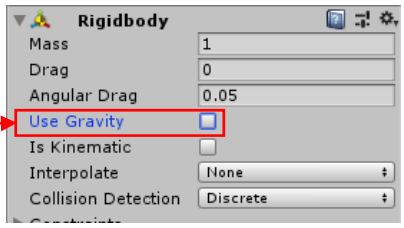
赤いボールに Rigidbody コンポーネントを持たせているのは、「落ちなさい」の意味ではなく、物理演算の仲間入りをさせる為です。ですが、黄色い箱（Cube）には持たせていません。あれ？黄色い箱は物理演算の仲間入りをしていないのに、「侵入者あり！」とか叫んでいますヨ？どちらか一方が持てばいいのでしょうか？実験してみます。



- ヒエラルキー欄の Ball を選択し、インスペクタの Rigidbody コンポーネントについて、歯車マークから Remove Component を選択して、これを外します。

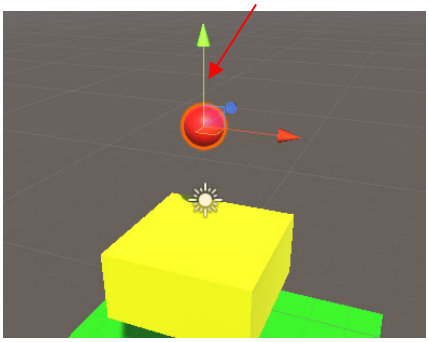


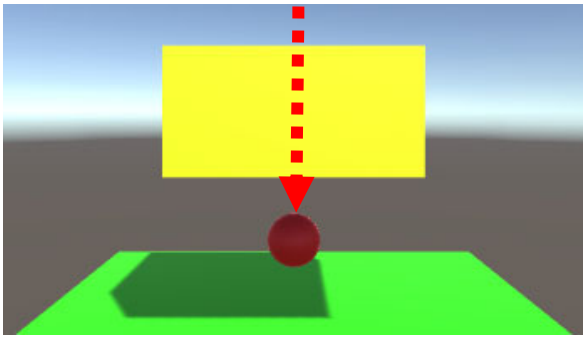
- ヒエラルキー欄の Cube を選択し、インスペクタの Add Component から Physics > Rigidbody を選択します。
  項目 Use Gravity をチェックオフにします。



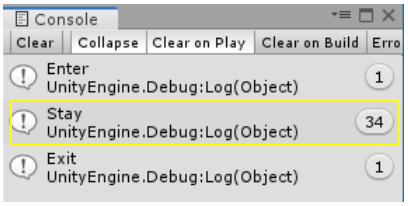
- プレイボタンを押下します。

  オブジェクト Ball を選択し、緑の Y 軸ギズモをドラッグし、Cube の下まで移動させます。

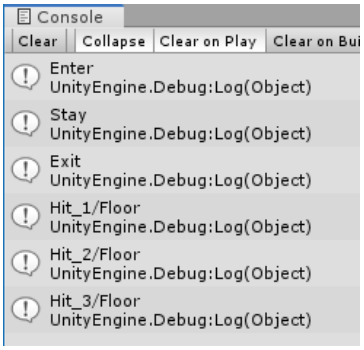




  コンソールを確認します。
  【STEP7】と同様の「侵入→ 滞在→ 脱出」が検出されています。衝突を検出するのに、Rigidbody はどちらが所持していても構わない訳です。



- 追加確認として、両方が Rigidbody を持てば、どうなるのでしょうか？
  Ball に再び Rigidbody コンポーネントを持たせてプレイボタンを押下してみます。
  【STEP7】と同様の結果となりました。



＜まとめコーナー＞ Unity の物理演算の特性

物理演算の仲間入りを宣言するコンポーネント Rigidbody は、**接触の当事者2名のどちらかが持っていれば接触を検出できる**ことが判りました。

物体 A	物体 B	結果
Rigidbody を持たない	Rigidbody を持たない	（×）接触検出は出来ない
Rigidbody を所有	Rigidbody を持たない	（○）接触検出が行われる
Rigidbody を持たない	Rigidbody を所有	（○）接触検出が行われる
Rigidbody を所有	Rigidbody を所有	（○）接触検出が行われる

また、もう一つ興味深い現象として、Ball は IsTrigger オフで反射する気まんまん！であるのに対して、Cube は IsTrigger オンなので、Ball は反射せずに侵入しました。反射は、Floor と Ball の関係のように、両方が IsTrigger オフでないと成立せず、どちらか一方が IsTrigger オンなら、結果は侵入になってしまう事も観察できます。

物体 A	物体 B	結果
IsTrigger がオフ	IsTrigger がオフ	反射
IsTrigger がオン	IsTrigger がオフ	侵入
IsTrigger がオフ	IsTrigger がオン	侵入
IsTrigger がオン	IsTrigger がオン	侵入

但し、IsTrigger が指定されていないオブジェクト（例：Floor）に対し、侵入系の処理（OnTriggerEnter などのトリガー系）を記述しても、当然、トリガーオブジェクトではないので実行はされません。

反射系の処理（OnCollisionEnter などのコリジョン系）は動きます。

＜特性のまとめ＞

- 物理演算の計算対象に加えたい物体には、コンポーネント Rigidbody を付加する。
- 接触後は反射と侵入の2系統。その2つを分けたのは IsTrigger のチェック項目である。
- 接触後の反射は、両方の物体が IsTrigger オフの時だけ発生する。
- 接触は、動く側に Rigidbody を付加する決まりはない。どちらか一方が持てば、反射や侵入の検知は可能。
- 反射は3つの事象を検知できる。 ＜接触開始＞→＜接触中＞→＜離れた瞬間＞
- 侵入は3つの事象を検知できる。 ＜侵入開始＞→＜侵入中＞→＜脱出の瞬間＞
- 接触物体は反射なら Collision 型、侵入なら Collider 型として情報（名前、タグ、各種パラメータ）を認識可能。

リスポーンについて

【STEP9】リスポーンの実現

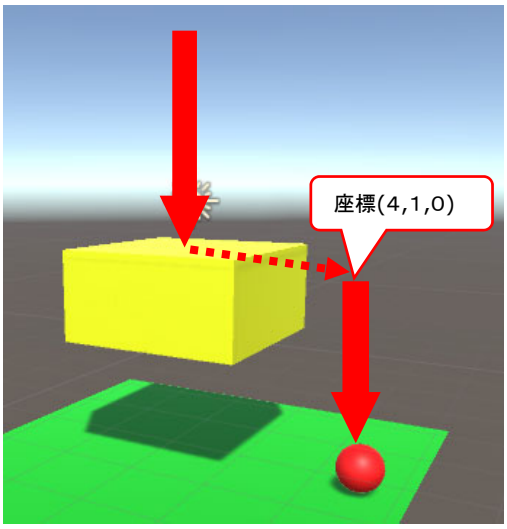
接触したオブジェクトを一瞬にして他の場所へ移動させ、ワープしたように見せる演出を、ゲーム業界ではリスポーンと呼んでいます。ここでは、黄色い箱に侵入したオブジェクトに対して、外部の座標を与えてやることで、箱への侵入はせずに、一瞬のうちにその外部の座標へリスポーンさせます。

- スクリプト CubeAction を次のように修正します。

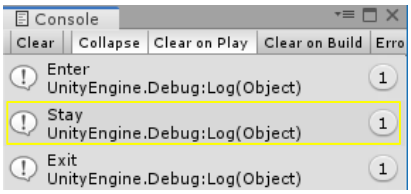
```
// 前略
    void OnTriggerEnter(Collider other) {
        Debug.Log("Enter");
        other.gameObject.transform.position = new Vector3(4, 1, 0);
    }
// 後略
```

- プレイボタンを押下します。

黄色い箱に接触した瞬間に座標(4,1,0)を与えられ、一瞬にして移動します。これがリスポーン処理の典型例です。



興味深い情報として、黄色い箱には、侵入1回は当然としても、滞在が1回と、脱出1回が記録されています。特性として知っておくべきでしょう。



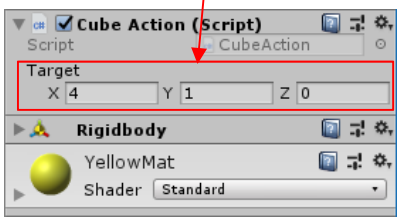
【STEP10】リスポーン先をパブリックで指定する

リスポーン先の Target 座標をプログラムで与えるのではなく、パブリック定義すると、インスペクタに出しておくことができ、数値入力で指定することができます。

- スクリプト CubeAction を次のように修正します。(同じ座標です。)

```
// 前略
public class CubeAction : MonoBehaviour {

    public Vector3 Target = new Vector3(4, 1, 0);

    void OnTriggerEnter(Collider other) {
        Debug.Log("Enter");
        other.gameObject.transform.position = Target;
    }
// 後略
```

ヒエラルキー欄の Cube を選択すると、インスペクタに項目が登場していますので、値を変更することが可能です。

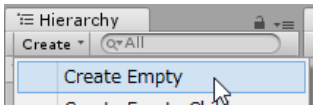


- プレイボタンを押下します。

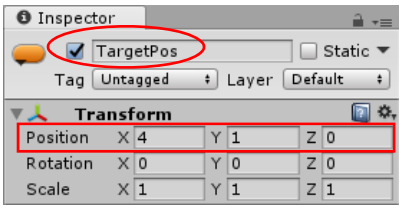
少し値を変更してみて、異なる場所にリスポーンされるか？を確認します。

【STEP11】リスポーン先をオブジェクトで与える

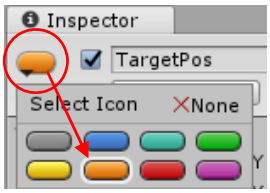
リスポーン先が数値のままだと直感的ではないことから、座標だけを持つダミーオブジェクトを作成し、そのオブジェクトを丸ごと与えてリスポーン先とする考え方に変更します。



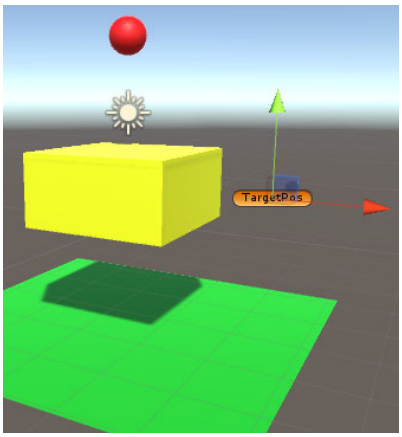
- ヒエラルキー欄の Create から Create Empty を選択し、空（から）のオブジェクトを制作します。
- 名称を TargetPos とし、インスペクタでパラメータを設定します。



- 空（から）のオブジェクトなので視覚的な存在がありません。目印をアイコンで設けるべく、見易い色を選んでおきます。



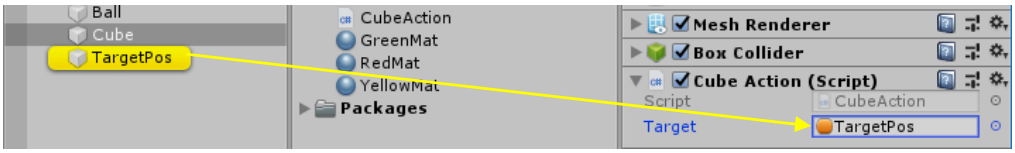
こんな位置関係になります。



- プログラム CubeAction を次のように修正します。

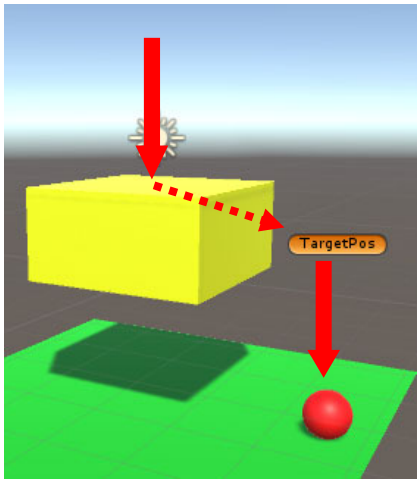
```
// 前略
public class CubeAction : MonoBehaviour {

    public GameObject Target;

    void OnTriggerEnter(Collider other) {
        Debug.Log("Enter");
        other.gameObject.transform.position = Target.transform.position;
    }
// 後略
```

- ヒエラルキー欄の Cube を選択し、インスペクタに登場した項目に、ヒエラルキー欄の TargetPos をドラッグ＆ドロップします。



- プレイボタンを押下します。

黄色の箱に接触した途端、TargetPos の座標をもらう考え方です。



【STEP12】 リスポーン先を名前（name）で探して与える

ターゲットを自分で探させます。手掛かりはヒエラルキー欄での名前（name）となります。

【構文】 GameObject.Find(“オブジェクト名”)

名前でオブジェクトを特定する方法です、覚えておきましょう。

- プログラム CubeAction を次のように修正します。

```
// 前略
public class CubeAction : MonoBehaviour {

    GameObject Target;

    void OnTriggerEnter(Collider other) {
        Debug.Log("Enter");
        Target = GameObject.Find("TargetPos");
        other.gameObject.transform.position = Target.transform.position;
    }
// 後略
```

- プレイボタンを押下します。 

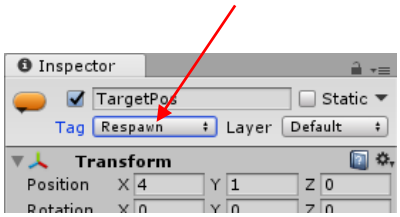
  見た目には変化はありませんが、ターゲットを渡してあげるのではなく、名前で探し出して、その座標をボールに与えています。

【STEP13】 リスポーン先をタグ（tag）で探して与える

ターゲットを自分で探させます。手掛かりをオブジェクトのタグ（tag）に変更します。

【構文】 GameObject.FindGameObjectWithTag(“タグ名”)

タグでオブジェクトを特定する方法です、覚えておきましょう。



- ヒエラルキー欄の TargetPos を選択し、タグを Respawn に設定します。

  タグ名 Respawn はあまりにも有名な処理なので、最初から存在しています。もちろん自身で新しく追加作成することも可能です。

- プログラム CubeAction を次のように修正します。

```
// 前略
public class CubeAction: MonoBehaviour {

    GameObject Target;

    void OnTriggerEnter (Collider other) {
        Debug.Log ("Enter");
        Target = GameObject. FindGameObjectWithTag ("Respawn");
        other.gameObject.transform.position = Target.transform.position;
    }
// 後略
```

- プレイボタンを押下します。 

  見た目には変化はありませんが、ターゲットを名前ではなく、タグ名で探し当てて、その座標をボールに与えています。

【考察:物体を探す】

＜いつ探すか？＞

　この**【STEP12】**と**【STEP13】**は、リスポーンするタイミングでオブジェクトを名前やタグで探しています。

　つまり、ヒットしたタイミングで探しているので、特に STEP12 の名前で探す場合などは、対象物が非常に数多く存在して、忙しくなってしまいます。音楽に合わせたプレイ操作のリズムゲームでは、処理が遅れたりする原因にもなります。

　そこで、Start()処理などで記述することで、事前に余裕を持って探して所有しておき、ヒットしたタイミングでは位置座標を与えるのみ！の作業量にしておくと、素早い処理消化が見込めます。

＜探す対象者の数は？＞

　仮に、ヒエラルキー内に非常に多くのゲームオブジェクトがあった場合、名前で特定しようとすると、非常に時間がかかりゲーム全体の負荷となります。シーンの全てから探さずに、特定の配下構造の中などの、狭い領域から探す工夫も考慮します。

＜複数が見つかるかも？＞

　今回は対象物を探す場合、その物体が1つに定まるのでエラーにならずに済んだ事実も見逃せません。同じタグや同じ名前を持つ物体が複数存在することは許されているので、複数が見つかってしまうと、どのオブジェクトか？が不明です。それ以前に、単数形の物体を見つけようとして複数が見つかると、その時点でエラーとなります。

　複数個を探す命令も別途ありますので、探す相手が1つに定まるのか？複数個になりそうか？を明確に意識することも必要です。

以上