

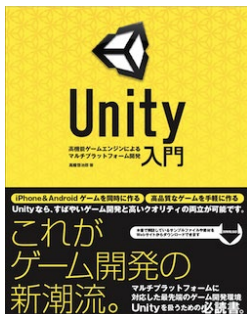
## BlockShooter (ブロックシューター)



画面の平面的なクリック位置(2次元座標)を、Unity のカメラから見た視野内での座標(3次元座標)に変換し、方向を求めて、弾丸を射出する3Dシューティングの仕組みを紹介します。

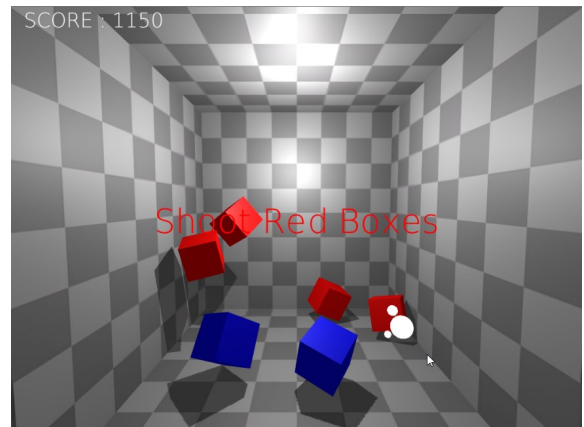
ゲームの内容は、赤と青の箱が次々と降ってくるので、現在のターゲットが赤／青のどちらかであるか？を元に、画面タッチ位置から弾丸を射出・命中させて得点するゲームです。

赤と青を間違えると減点になります。ターゲット表示は数秒毎に切り替わり、制限時間内で高得点を競う内容です。



このゲームは BallMaze と同様、ソフトバンククリエイティブ社刊:Unity 入門で紹介されていたゲームを元に再構成しています。

省いたゲーム演出もありますが、Javascript での内容を C#で書き換えたこと、パーティクルの仕組みを新式に置き換えたこと、インターフェースが UnityGUI に変わったこと、サウンド鳴動させた改変などを行っています。

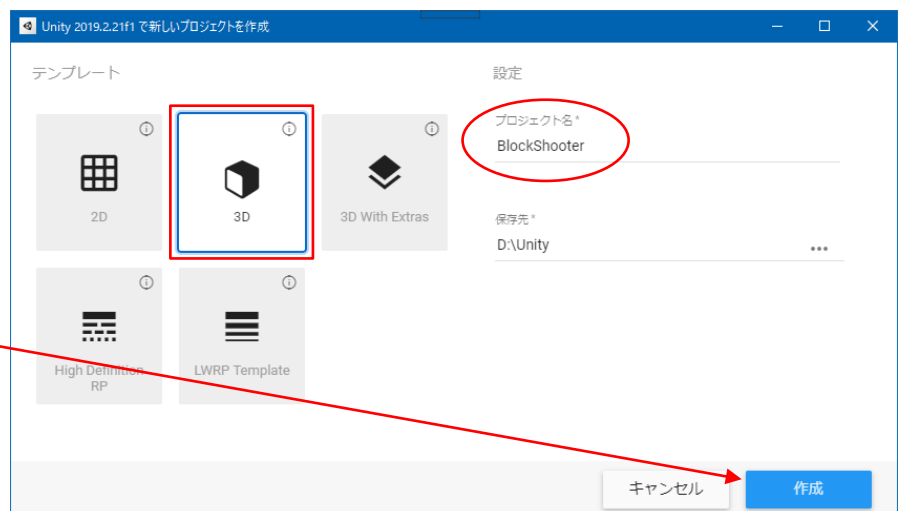


### シーンの構築

#### 【STEP1】プロジェクトの準備

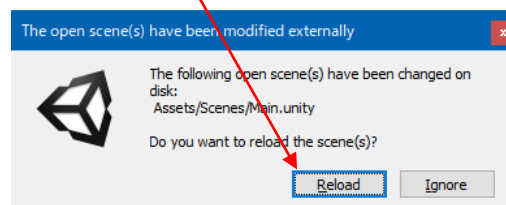
- Unity を起動し、新規プロジェクト **BlockShooter** をテンプレート:3Dで作成します。

ボタン**作成**を押下します。



- 現在のシーン名が暫定の **SampleScene** なので、プロジェクト欄の **SampleScene** を右クリック >Rename を選択し、**Main** と命名します。

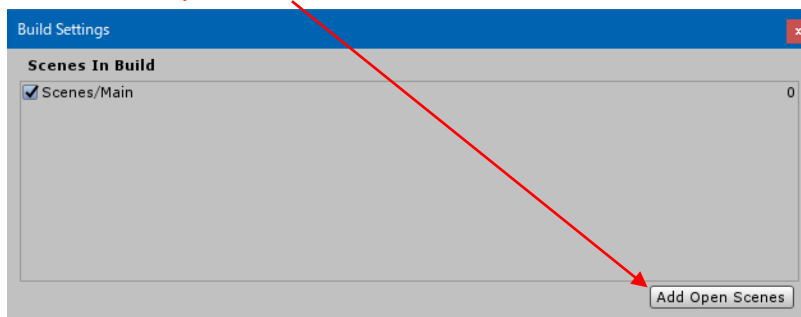
シーンをリロードするか聞かれるので、**Reload** を選択します。



- メニューFile の **Build Settings...**を選び、ボタン **Add Open Scenes**を押下します。

ビルド一覧(Scenes In Build)にこのシーン Main が追加されます。

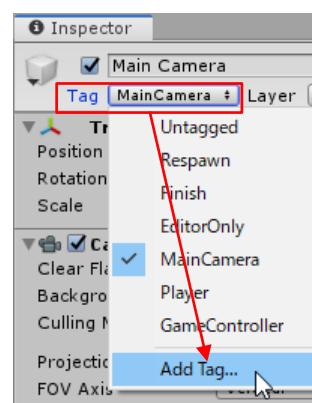
このパネルは閉じておきます。



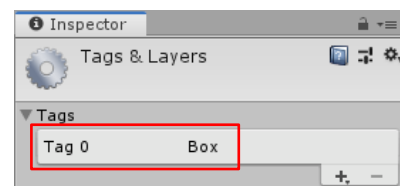
- Game 画面のサイズを **Standalone (1024x768)**に設定します。



- ヒエラルキー欄の **Main Camera** を選択し、インスペクタの Tag から **Add Tag...**を押下します。

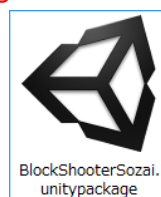


- タグ欄の**プラス(+)**ボタンを押下して、タグ **Box** を作成します。

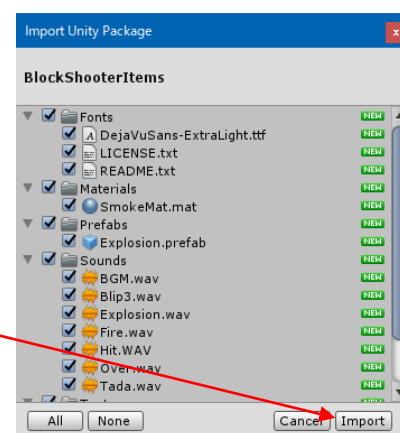


## 【STEP2】 外部素材を取得

- メニューAssets から Import Package > **Custom Package** と進み、今回のフォルダ内にある **BlockShooterSozai.unitypackage** を指定します。

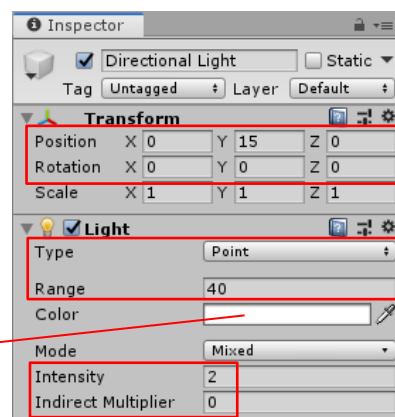


- 内容物が表示されたら、**Import** を押下し、それら全てをインポートします。

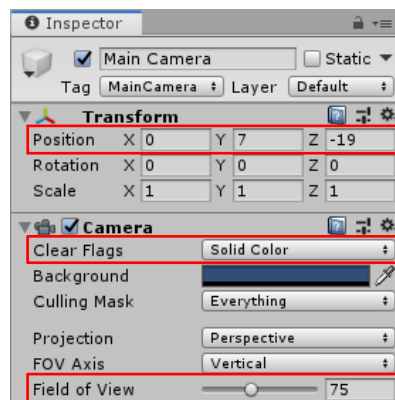


### 【STEP3】 光源とカメラの設定

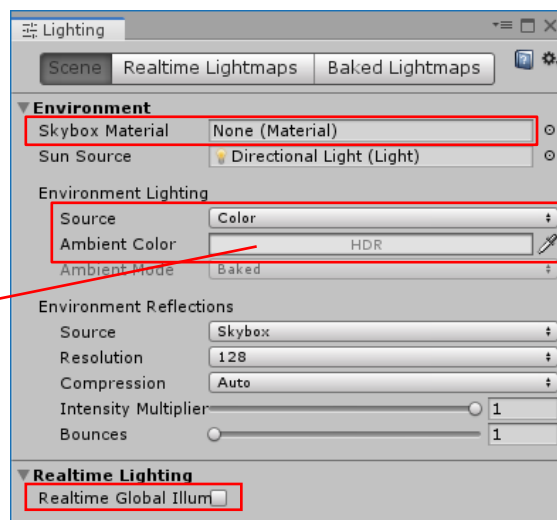
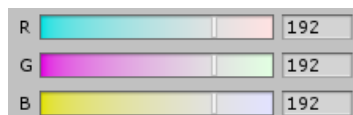
- ヒエラルキー欄の **Directional Light** を選択し、インスペクタでパラメータを設定します。



- ヒエラルキー欄の **Main Camera** を選択し、インスペクタでパラメータを設定します。



- メニューWindow > Rendering > **Lighting Settings** を選択し、インスペクタでパラメータを修正します。

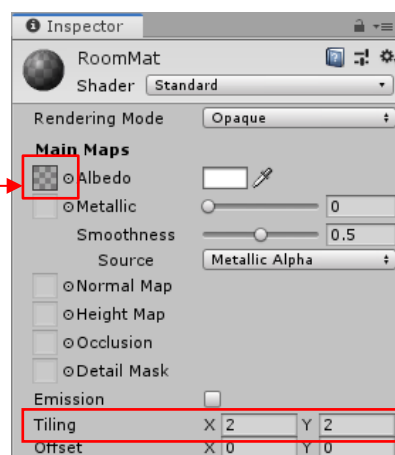


このパネルは閉じておきます。

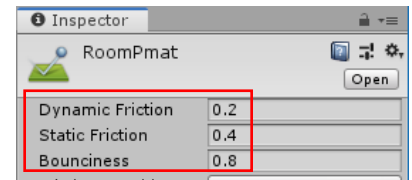
### 【STEP4】 部屋の構築

- プロジェクト欄の Create からマテリアル **RoomMat** を作成します。  
インスペクタでパラメータを設定します。

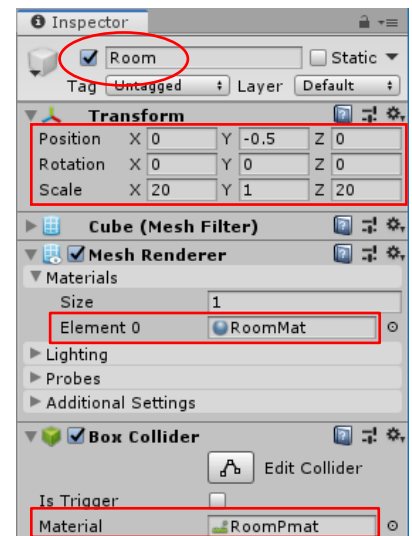
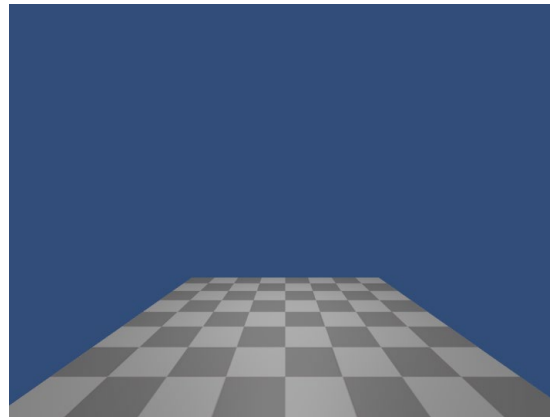
画像 **Checker** を指定します。



- プロジェクト欄の Create から **Physic Material**(物理マテリアル)を選択し、名称を **RoomPmat** とします。インスペクタでパラメータを設定します。

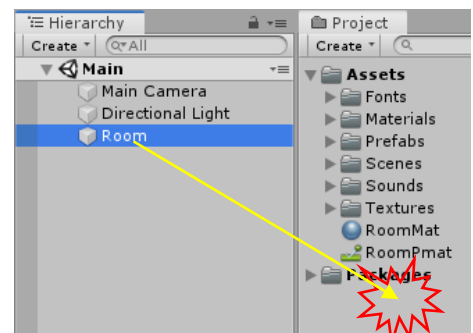
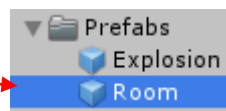


- ヒエラルキー欄の Create から3D Object > **Cube**を選び、名称を **Room** とします。インスペクタでパラメータを設定します。



- ヒエラルキー欄の Room をプロジェクト欄にドラッグ & ドロップして**プレハブ化**します。

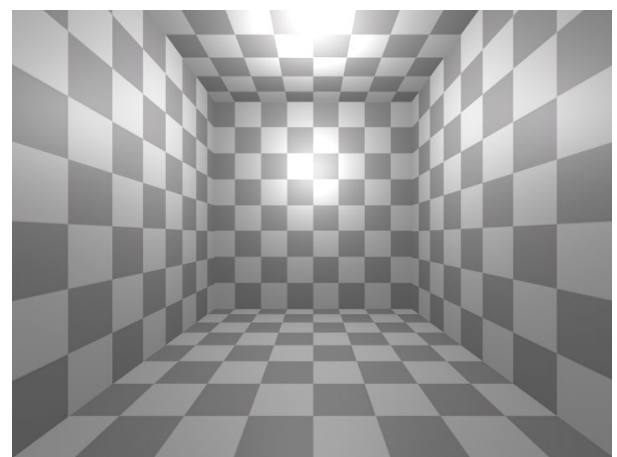
プレハブ化されました。Prefabs フォルダに移動させておきます。



- **ヒエラルキー欄の Room** を4回複製 (Ctrl + D)して計5個とし、インスペクタでパラメータを設定します。

	Position			Rotation		
	X	Y	Z	X	Y	Z
Room	0	-0.5	0	0	0	0
Room1	0	20.5	0	0	0	0
Room2	-10	10	0	0	0	90
Room3	10	10	0	0	0	90
Room4	0	10	10	90	0	0

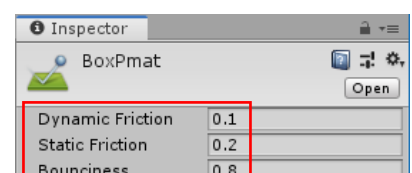
部屋が組みあがります。



## ターゲット関連の構築

### 【STEP5】 落下してくる箱の準備

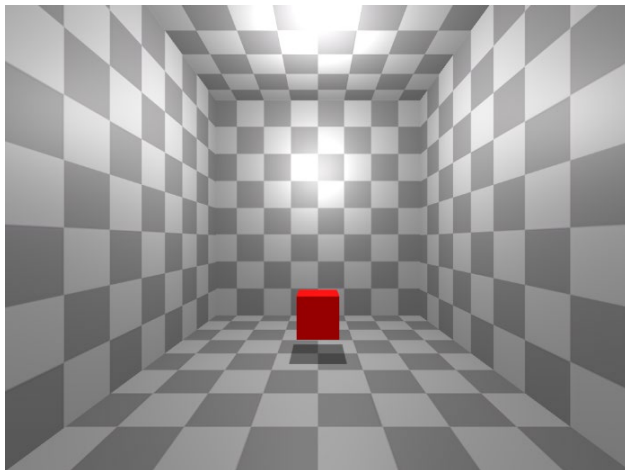
- プロジェクト欄の Create から **Physic Material** を選択し、**BoxPmat** と命名します。インスペクタでパラメータを設定します。



- プロジェクト欄の Create からマテリアルを作成し、RedBoxMat と命名します。インスペクタでパラメータを設定します。

- ヒエラルキー欄の Create から 3D Object > Cube を選び、名称を RedBox とします。

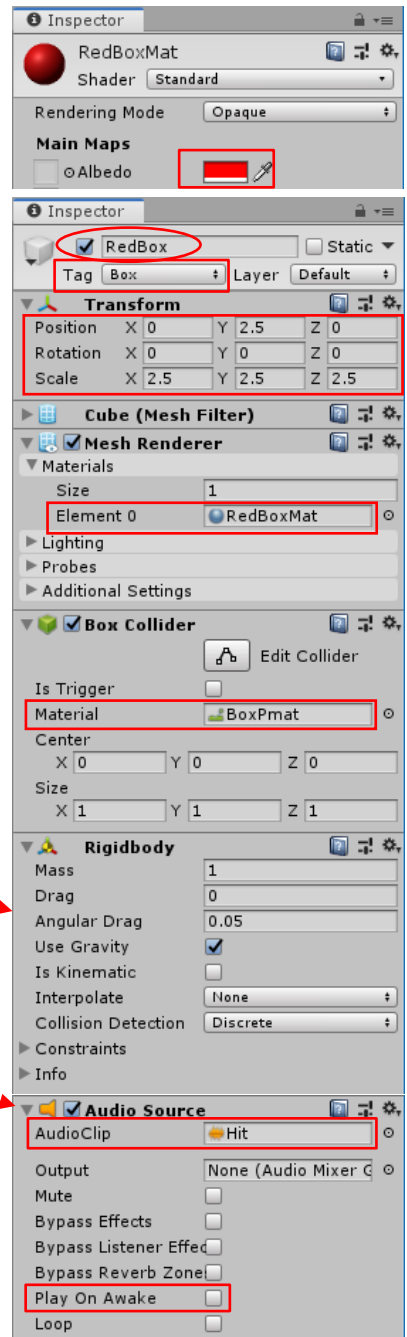
インスペクタでパラメータを設定します。



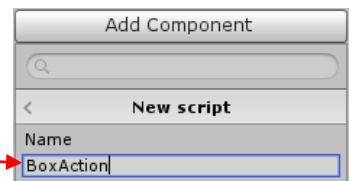
- インスペクタの Add Component から Physics > Rigidbody を選択し追加します。(物理挙動の反映に必要！)

- インスペクタの Add Component から Audio > AudioSource を選択し追加します。

パラメータを設定します。

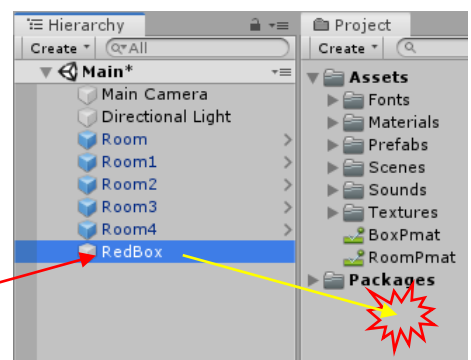


- 実際にプログラムを記述するのは後ですが、先に取り付けのみ、行います。インスペクタの Add Component から New script を選びます。スクリプトの名称を BoxAction とします。



- ヒエラルキー欄の RedBox をプロジェクト欄にドラッグ＆ドロップしてプレハブ化します。

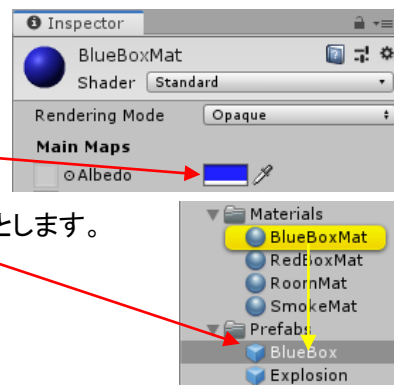
ヒエラルキー欄に残った RedBox は削除しておきます。



## 【STEP6】 青いは複製で作成

- プロジェクト欄の **RedBoxMat** を複製 (Ctrl + D) し、名称を **BlueBoxMat** と命名します。

Albedo (アルベド) カラーに青を設定します。



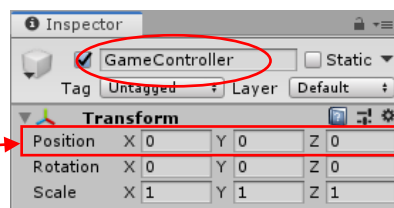
- プロジェクト欄のプレハブ **RedBox** を複製 (Ctrl + D) し、名称を **BlueBox** とします。

- 先ほどの **BlueBoxMat** を **BlueBox** にドラッグ & ドロップします。

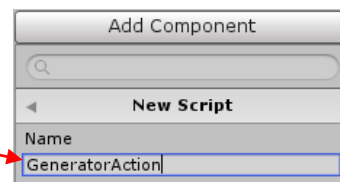
## 【STEP7】 箱の発生

- 空 (から) のオブジェクトを準備し、ゲームの責任者を作ります。ヒエラルキー欄の Create から **Create Empty** を選択し、**GameController** と命名します。

原点にあるか？を確認します。



- この **GameController** を選択し、インスペクタの **Add Component** から **New script** を選びます。スクリプトの名称を **GeneratorAction** と命名します。



- スクリプト **GeneratorAction** を次のように編集します。

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

public class GeneratorAction : MonoBehaviour {

    public float Interval = 1.0f; //発生間隔(秒)
    public GameObject BlueBoxPrefab; //青箱のプレハブ
    public GameObject RedBoxPrefab; //赤箱のプレハブ
    bool NextIsRed; //次は赤箱か？の真偽
    float Elapsed; //経過時間

    void Start () {
        NextIsRed = true; //赤から開始
        Elapsed = 0.0f;
    }

    void Update () {
        Elapsed -= Time.deltaTime;
        if (Elapsed <= 0.0f) { //発生までの時間がゼロになったら
            Vector3 Pos = new Vector3( 0, 10.0f, 0 ); //箱を生成するランダム位置Pos
            Pos.x = Random.Range( -8.0f, 8.0f );
            Pos.z = Random.Range( -4.0f, 4.0f );
            //生成するプレハブを割り当てる
            GameObject SpawnBox = NextIsRed ? RedBoxPrefab : BlueBoxPrefab;
            //箱のプレハブをインスタンス生成する
            Instantiate(SpawnBox, Pos, Random.rotation);
            Elapsed = Interval; //次の発生までの時間を設定
            NextIsRed = !NextIsRed; //次の発生の為に赤青を反転
        }
    }
}
```

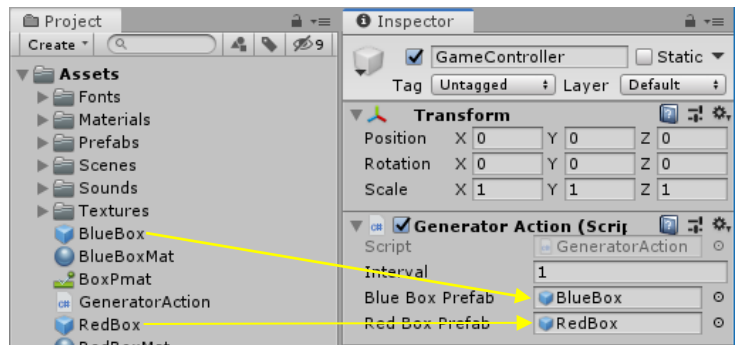
### 【確認項目】

- ・ 3項演算子の利用
- ・ ランダムな回転
- ・ トグル式反転



- ヒエラルキー欄の **GameController** を選択します。

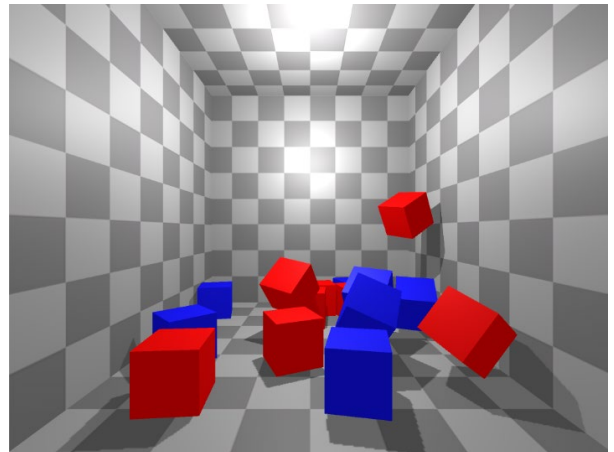
インスペクタの欄で赤箱、青箱のプレハブを指定する欄が表示されるので、それぞれをドラッグして指定します。



- プレイボタンを押下します。



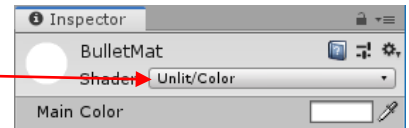
実際に箱が生成されるか、動作確認します。



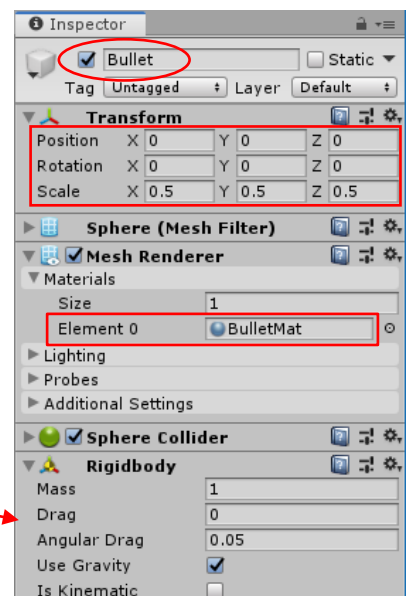
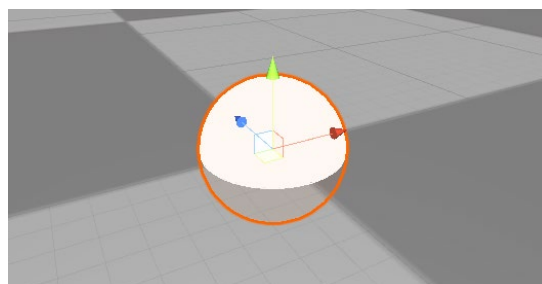
## 弾丸射出の実現

### 【STEP8】 弾丸の作成

- プロジェクト欄の Create からマテリアルを作成し **BulletMat** と命名します。  
シェーダーには **Unlit > Color** を選んでおきます。

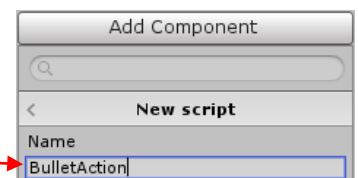


- ヒエラルキー欄の Create から 3D Object > **Sphere** を作成し、名称を **Bullet** とします。  
インスペクタでパラメータを設定します。

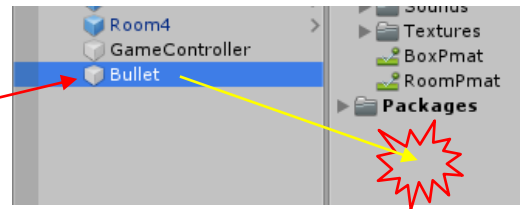


- インスペクタの Add Component から Physics > **Rigidbody** を選択して追加します。

- 実際にプログラムを記述するのは後ですが、先に取り付けのみ、します。  
インスペクタの **Add Component** から **New Script** を選びます。スクリプトの名称を **BulletAction** とします。

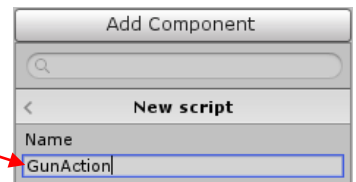


- ヒエラルキー欄の Bullet をプロジェクト欄にドラッグ & ドロップして  
プレハブ化します。
- ヒエラルキー欄に残った Bullet は削除しておきます。



### 【STEP9】 弾丸の射出

- ヒエラルキー欄の Main Camera を選択し、インスペクタの Add Component から New script を選び、名称を GunAction と命名します。
- スクリプト GunAction を次のように編集します。



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

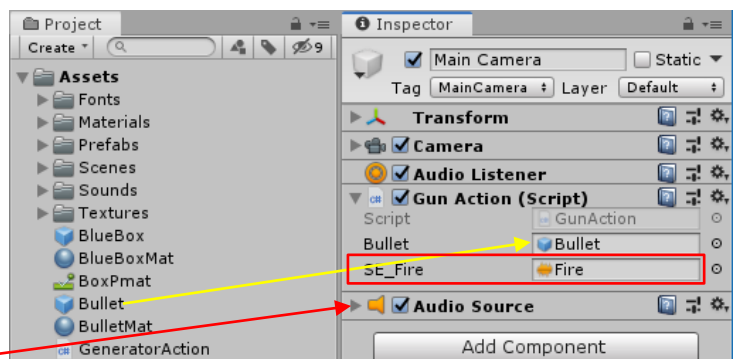
public class GunAction : MonoBehaviour {

    public GameObject Bullet; //弾丸のプレハブ
    public AudioClip SE_Fire; //射出音
    AudioSource myAudio;

    void Start() {
        myAudio = GetComponent<AudioSource>();
    }

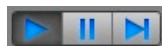
    void Update() {
        if ( Input.GetMouseButtonDown( 0 ) ) {
            Instantiate( Bullet, transform.position, transform.rotation );
            myAudio.PlayOneShot( SE_Fire ); //射出音鳴動
        }
    }
}
```

- ヒエラルキー欄の Main Camera を選択し、インスペクタに登場したパラメータを設定します。



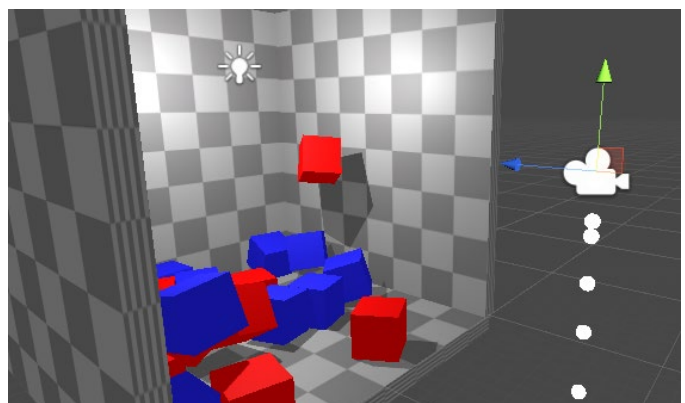
- インスペクタの Add Component から Audio > Audio Source を選択します。

- プレイボタンを押下します。



画面をクリックすると、射出音はするのですが、カメラに弾丸が設置され、飛ばずに落ちて行ってしまいます。初速度を持たせるのを忘れているのです。

弾丸プレハブ bulletPrefab をカメラ位置にインスタンス生成するだけでなく、動き(奥へ飛ぶ初速度)を与える為にゲームオブジェクトとして生成し、直後に速度を与える手法とします。





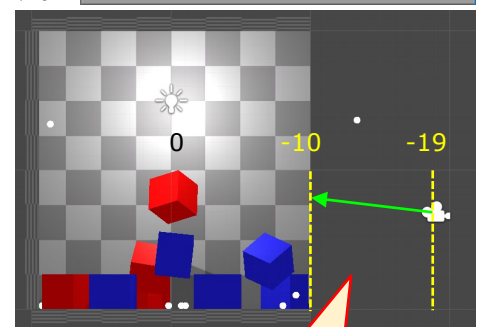
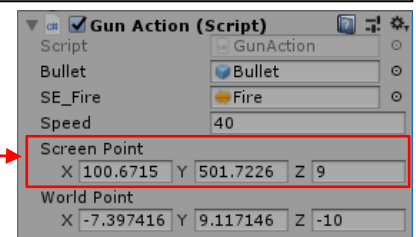
● スクリプト **GunAction** を次のように編集します。

```
//前略
public float Speed = 40.0f; //初速度
public Vector3 screenPoint; //スクリーン座標
public Vector3 worldPoint; //ワールド座標

void Start() {
    myAudio = GetComponent<AudioSource>();
}

void Update () {
    if (Input.GetMouseButtonDown(0)) {
        //クリック位置の座標
        screenPoint = Input.mousePosition; //XとYはスクリーン座標
        screenPoint.z = 9.0f; //Zはワールド座標で、カメラからの距離
        //クリック座標をカメラから見たUnity空間での座標worldPointに変換する
        worldPoint = GetComponent<Camera>().ScreenToWorldPoint( screenPoint );
        Debug.DrawLine( transform.position, worldPoint ,Color.green);
        //その座標とカメラ位置との2つで向きを求める
        Vector3 Dir = ( worldPoint - transform.position ).normalized;
        //カメラ位置に弾丸をインスタンス生成しつつ、Bへ代入する
        GameObject B = Instantiate( Bullet, transform.position, transform.rotation ) as GameObject;
        B.GetComponent<Rigidbody>().velocity = Dir * Speed; //弾丸に初速度を与える
        myAudio.PlayOneShot( SE_Fire ); //射出音鳴動
    }
}
}
```

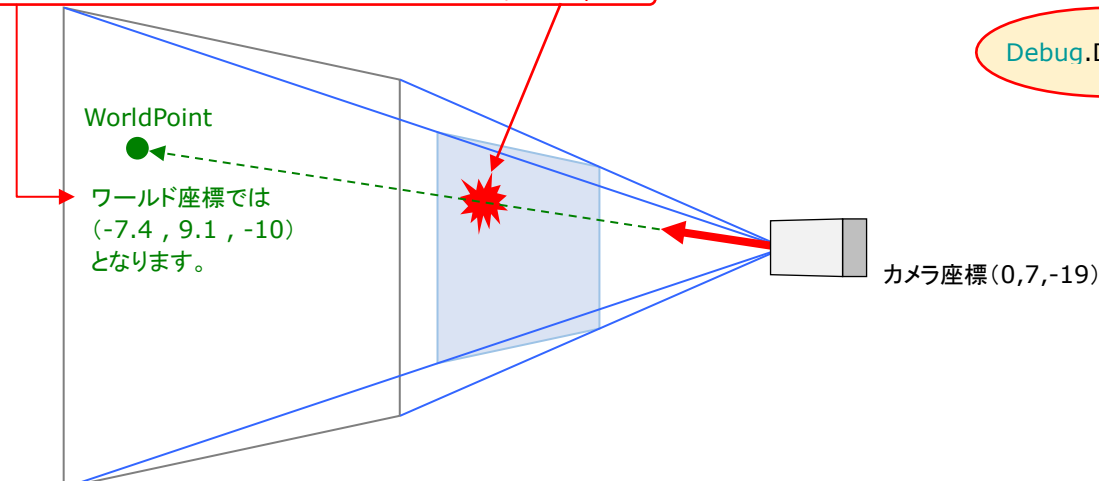
1. サンプルのクリックでは、スクリーン座標が約(100,500)であることが判ります。クリックでは奥行き(Z座標)が無いので、Roomまでの距離をワールド座標系で9として代入しています。
2. その地点は、ワールドでは約(-7.4 , 9.1 , -10)の位置が求まります。
3. カメラの座標は(0,11,-19)なので、この2つを用いてベクトルを求めます。  
緑点線の矢印となります。
4. 緑点線の長さは不要です。向きだけが欲しい上に、長さを1とし、赤い矢印を求めます。
5. これに初速度 40 を乗算し、結果的に、あたかもクリックした向きに弾を撃ち出すような現象が実現できるわけです。



**ScreenPoint**

カメラからの距離を9とする。

座標(100,500)

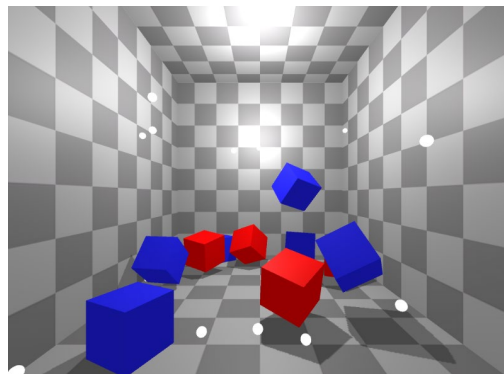


Debug.DrawLine

- プレイボタンを押下します。



画面をクリックした位置から弾丸が射出される動作確認をします。



## 【STEP10】 弾丸からの挙動発信

- スクリプト **BulletAction** を次のように編集します。

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

public class BulletAction : MonoBehaviour {

    void OnCollisionEnter(Collision other) {
        if (other.gameObject.tag == "Box") {
            //相手が箱だったらOnDamageをメッセージ通告する
            other.gameObject.SendMessage("OnDamage", SendMessageOptions.DontRequireReceiver);
        }
        Destroy(gameObject); //何かに当たれば自身(弾丸)を撤去
    }
}
```

- 弾丸が赤箱や青箱に向かってメッセージ「OnDamage」を送信します。赤箱や青箱はそれを受信し、対応する処理を実行します。

スクリプト **BoxAction** を次のように編集します。

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

public class BoxAction : MonoBehaviour {

    void Start () {

    }

    void OnDamage () {
        Destroy(gameObject);
    }

    void Update () {

    }
}
```

- プレイボタンを押下します。



マウスクリックで射出した弾が、壁や箱に当たると消えること、また、箱は撃たれたら消えることを確認します。

## 【STEP11】 爆発エフェクトの追加

撃たれた箱は、上方向に跳ね上げられる挙動を付加し、0.4 秒ほど滞空してから爆発プレハブを生成し、その後、自身を撤去する、という効果を演出します。

- スクリプト **BoxAction** を次のように編集します。(と言うか、**ほとんど書き替え**です。)

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

public class BoxAction : MonoBehaviour {

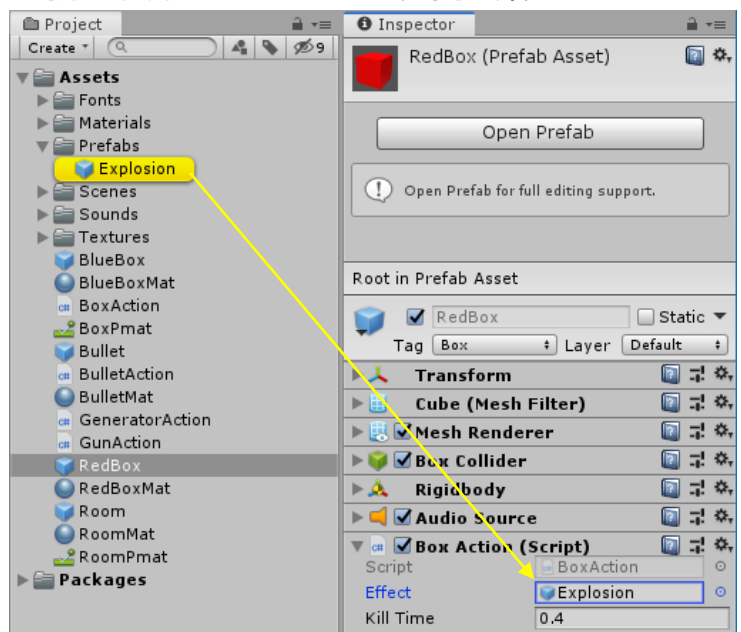
    float Elapsed = 0.0f;
    bool Damaged = false; //撃たれたか
    public GameObject Effect; //爆発エフェクトのプレハブ
    public float KillTime = 0.4f; //撃たれてから爆発するまでの遅延時間
    AudioSource myAudio; //自身の音源

    void Start() {
        myAudio = GetComponent<AudioSource>(); //自身の音源を取得
    }

    void OnDamage() {
        //Destroy(gameObject); //コメントアウトします
        if (!Damaged) {
            Damaged = true;
            GetComponent<Rigidbody>().AddForce(Vector3.up * 15.0f, ForceMode.Impulse);
            myAudio.Play(); //命中音鳴動
        }
    }

    void Update () {
        if (Damaged) {
            Elapsed += Time.deltaTime; //経過時間を計測
            if (Elapsed > KillTime) {
                //爆発エフェクトを設置
                GameObject Fx = Instantiate( Effect, transform.position,
                    transform.rotation ) as GameObject;
                Destroy( Fx, 2.0f ); //爆発エフェクトを撤去
                Destroy( gameObject ); //自身(箱)を撤去
            }
        }
    }
}
```

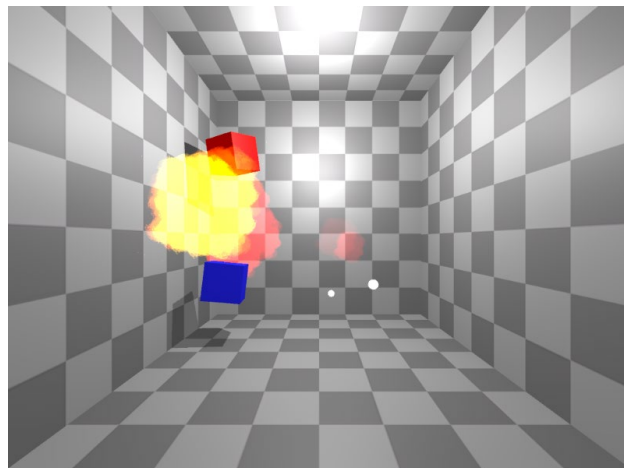
- このスクリプトはプレハブ **BlueBox** と **RedBox** の両方に取り付けられているので、両方で爆発エフェクトプレハブ **Explosion** を割り付けます。



- プレイボタンを押下します。



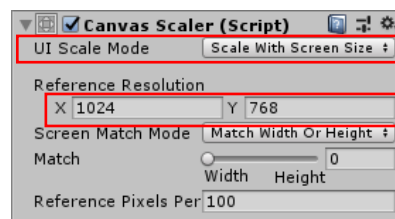
マウスクリックで弾丸を射出し、箱は弾が当たった時の上方向に跳ね上げられ、少ししてから消えると同時に爆発プレハブが生成されます。



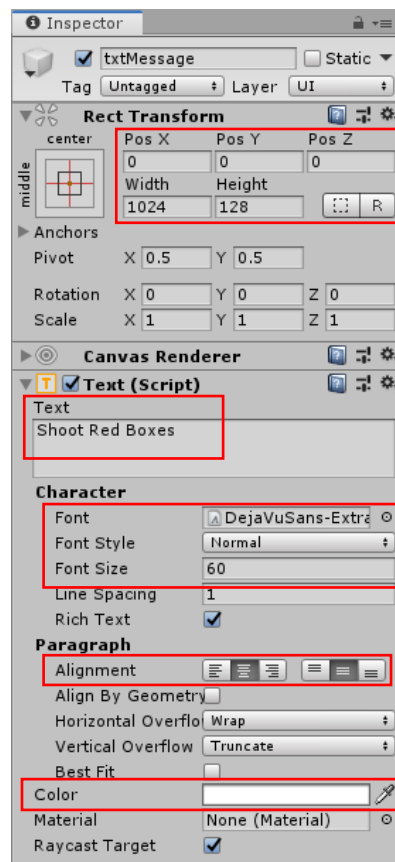
## スコアと時間の計測

### 【STEP12】 スコアの管理

- ヒエラルキー欄の Create から UI > **Text** を選びます。
- 同時に生成されている、ヒエラルキー欄の **Canvas** をインスペクタで設定します。



- ヒエラルキー欄の **Text** を選択し、**txtMessage** と命名します。インスペクタでパラメータを設定します。

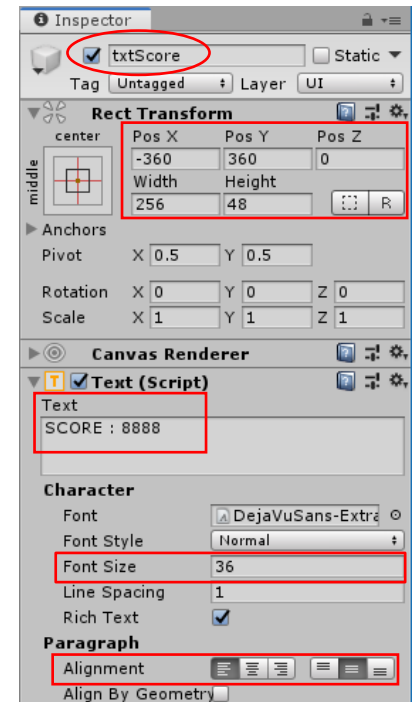


### 【学習項目】

フォントファイルを指定する場合は、そのフォントの使用規約を必ず確認します。多くの場合、アプリなどで埋め込んで配布してしまうと、無断配布・無断複製の罪となってしまいます。

フリーフォントの場合でも、ゲームやアプリへの使用方法についてOKか？NGか？が明記されていることがあり、OKの場合でもクレジットの記述や同梱ファイル指定などが指定されていますので、必ず確認します。

- **txtMessage** を複製 (Ctrl + D) して名称を **txtScore** と命名します。  
インスペクタでパラメータを指定します。



- スクリプト **BoxAction** を次のように編集します。

```
//～前略～

public string ColorName; //自身の色名
GameObject Manager; //マネージャー

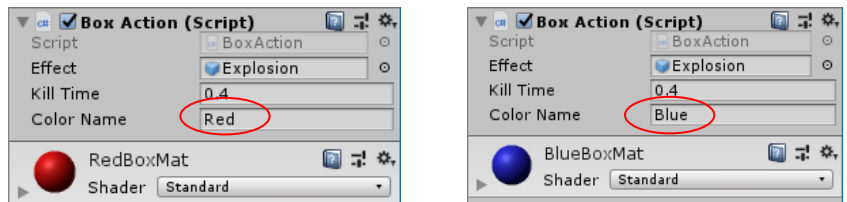
void Start() {
    myAudio = GetComponent<AudioSource>(); //自身の音源を取得
    Manager = GameObject.Find("GameController"); //マネージャーを探して取得する
}

void OnDamage() {
    if (!Damaged) {
        Damaged = true;
        GetComponent<Rigidbody>().AddForce(Vector3.up * 15.0f, ForceMode.Impulse);
        myAudio.Play(); //命中音鳴動
    }
}

void Update () {
    if (Damaged) {
        Elapsed += Time.deltaTime; //経過時間を計測
        if (Elapsed > KillTime) {
            //撤去直前に自身の色名を添えてOnDestroyBoxをマネージャーに通知。
            Manager.SendMessage( "OnDestroyBox", ColorName,
                                SendMessageOptions.DontRequireReceiver );

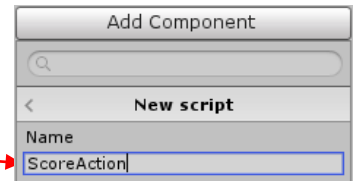
            //爆発エフェクトを設置
            GameObject Fx = Instantiate( Effect, transform.position,
                                         transform.rotation ) as GameObject;
            Destroy( Fx, 2.0f ); //爆発エフェクトを撤去
            Destroy( gameObject ); //自身(箱)を撤去
        }
    }
}
}
```

- 赤・青の両方のプレハブにあるスクリプト欄に、色名を設定する欄が登場するので **Red**、**Blue** と入力します。



ボールは自身を撤去する直前に、自身の色名(文字列)を引数に用い、メッセージ「OnDestroyBox」を送信しました。受信するのは GameController です。メッセージに含まれた色名を聞いて、現在が青／赤のどちらのタイミングか？を判定して、一致なら加点、不一致なら減点を行う**スコア管理専用のスクリプト**を追加します。

- ヒエラルキー欄の **GameController** を選択し、インスペクタの Add Component から **New script** を選びます。
- スクリプトの名称を **ScoreAction** と命名します。
- スクリプト **ScoreAction** を次のように編集します。



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを用いるのに必要

public class ScoreAction : MonoBehaviour {

    float Elapsed; //経過時間
    bool TargetIsRed; //ターゲットは赤か？
    public int Score; //スコア
    public int Reward = 100; //加点ポイント
    public int Penalty = -25; //減点ポイント
    public float Interval = 10; //赤青の切り替え間隔(既定値10秒)
    public Text txtMessage; //メッセージ表示
    public Text txtScore; //スコア表示

    void Start () {
        txtScore.text = "SCORE : 0";
        TargetIsRed = true; //赤から開始
        Score = 0; //スコア
    }

    void OnDestroyBox(string boxColorName) {
        //破壊された箱の色名を元に判定して得点計算・表示する
        Score += (boxColorName == GetTargetColorName()) ? Reward : Penalty;
        txtScore.text = "SCORE : " + Score;
    }

    string GetTargetColorName() { //真偽値をもとに文字列を返す関数
        return TargetIsRed ? "Red" : "Blue";
    }

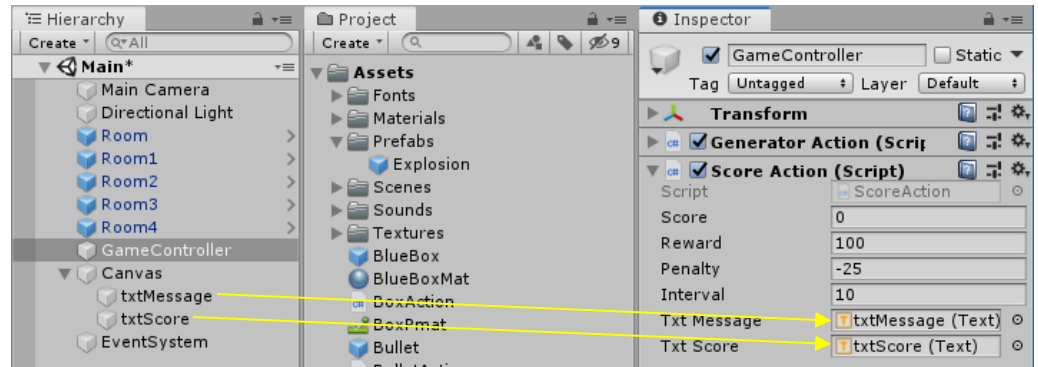
    void Update () {
        Elapsed += Time.deltaTime;
        if (Elapsed > Interval) {
            TargetIsRed = !TargetIsRed; //赤青切り替え
            Elapsed = 0.0f;
        }
        //ターゲットを字と色で指示
        txtMessage.text = "Shoot " + GetTargetColorName() + " Boxes";
        txtMessage.color = TargetIsRed ? Color.red : Color.blue;
    }
}
```

#### 【学習項目】

文字列型関数



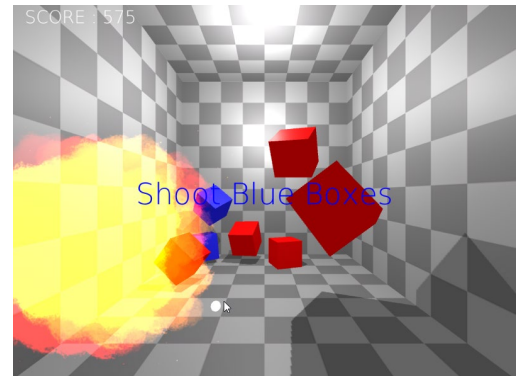
- ヒエラルキー欄の GameController を選択し、インスペクタに登場したパブリック項目に、**txtScore** と **txtMessage** をドラッグ & ドロップします。



- プレイボタンを押下します。



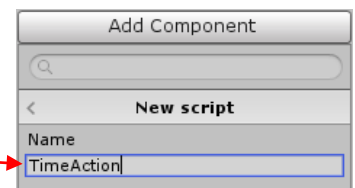
プレイ中に標的の指示が出て、青と赤が切り替わること、スコアが加点減点に応じて増減することを確認します。



### 【STEP13】 時間の管理

制限時間の減少は表示しない仕組みとしています。(もちろん、表示も出来ますよ。)

- ヒエラルキー欄の GameController を選択し、インスペクタの Add Component から **New Script** を選びます。  
スクリプトの名称を **TimeAction** とします。
- スクリプト **TimeAction** を次のように編集します。



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

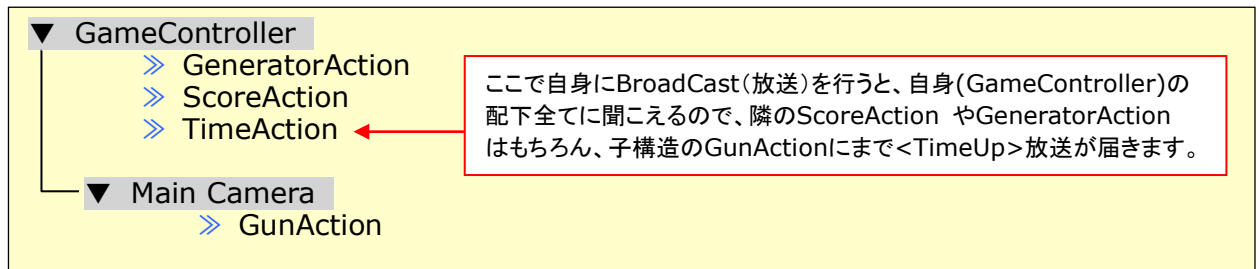
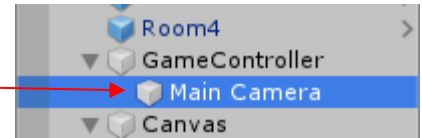
public class TimeAction : MonoBehaviour {

    public float TimeLimit = 60.0f; //制限時間
    float Elapsed; //経過時間
    bool isPlaying; //プレイ中か? の真偽値

    void Start() {
        Elapsed = 0.0f;
        isPlaying = true;
    }

    void Update () {
        Elapsed += Time.deltaTime;
        if (isPlaying) {
            if (Elapsed >= TimeLimit) {
                //制限時間を超えたらTimeUpを放送する
                BroadcastMessage( "TimeUp",SendMessageOptions.DontRequireReceiver );
                isPlaying = false; //プレイ中を偽にする
            }
        }
    }
}
```

- ヒエラルキー欄の **MainCamera** を GameController の配下構造になるように組み替えます。



- プレイ終了が把握できることで、音源(Audio Source)を持つ MainCamera は BGM の鳴動期間とタイムアップサウンドの管理が可能となります。  
スクリプト **GunAction** を次のように編集します。

//～前略～

```
public AudioClip SE_Over; //タイムアップ音
AudioSource myAudio;
public float Speed = 40.0f; //初速度
public Vector3 screenPoint; //スクリーン座標
public Vector3 worldPoint; //ワールド座標

void Start() {
    myAudio = GetComponent<AudioSource>();
    myAudio.Play(); //BGMを鳴動
}

void TimeUp() {
    enabled = false; //スクリプトを停止
    myAudio.Stop(); //BGMを停止
    myAudio.PlayOneShot( SE_Over ); //タイムアップ鳴動
}
```

//～後略～

- ヒエラルキー欄の **MainCamera** を選択し、インスペクタに登場した項目を設定します。

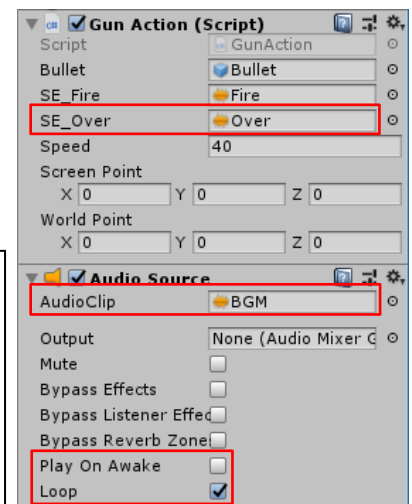
- TimeAction が放送した TimeUp を聞く側の以下のスクリプトに、次の処理を追加します。

- GeneratorAction
- ScoreAction

//～前略～

```
void TimeUp() {
    enabled = false; //スクリプトを停止
}
```

//～後略～

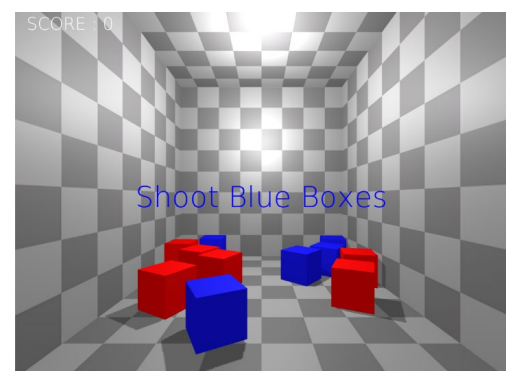


- プレイボタンを押下します。  
制限時間が経過したら、



- ①箱の生成と、
- ②弾丸生成と、
- ③ターゲット切り替えの 全てが止まります。

インスペクタで制限時間を、15 秒などの数字で実験し、BGMやタイムアップ鳴動などを確認します。

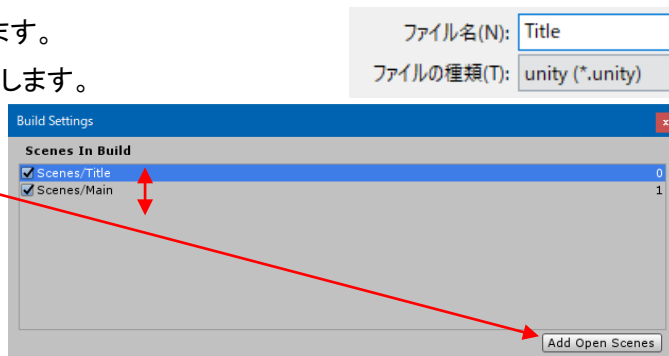


## シーケンス運営

### 【STEP14】複製してタイトル画面を作る

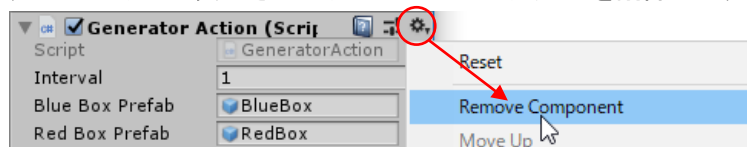
- 今まで作ってきたシーン Main を保存 (Ctrl + S) します。
- メニューFile から **Save As...** を選び、名称を **Title** とします。
- メニューFile から **Build Settings...** を選び、現在のシーン Title を **追加 (AddOpenScenes)** します。

このリストの順序でゲームが進むことを考慮し、Title が上になるように位置を入れ替えます。

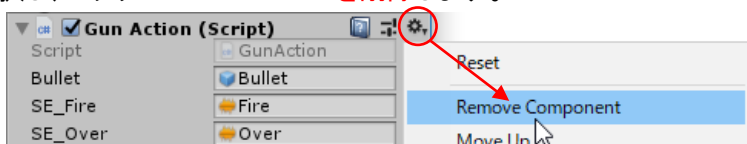


- ヒエラルキー欄の **GameController** を選択し、インスペクタに表示される以下の3つのスクリプトを削除します。

- **GeneratorAction**
- **TimeAction**
- **ScoreAction**

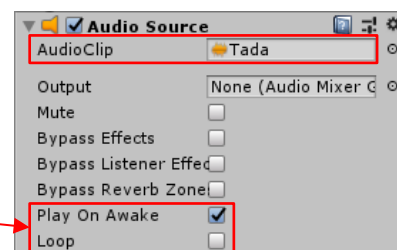


- 同様にヒエラルキー欄の **MainCamera** を選択し、スクリプト **GunAction** を削除します。

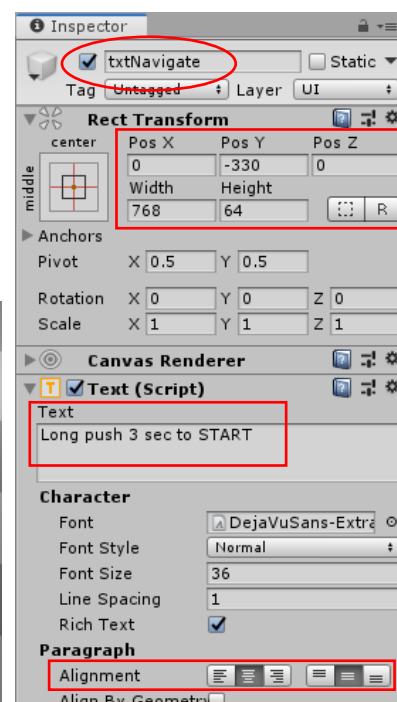


- この MainCamera の AudioSource に起動音サウンドを指定します。

ゲームが動き出したら、いきなり鳴動させ、ループはしないようにします。

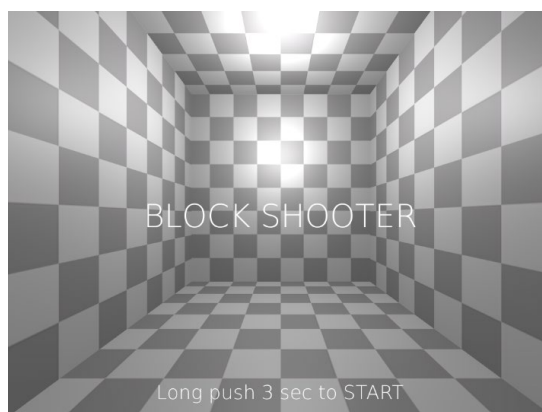


- ヒエラルキー欄の **txtScore** を選択し、名称を **txtNavigate** にします。インスペクタのテキスト内容を設定します。

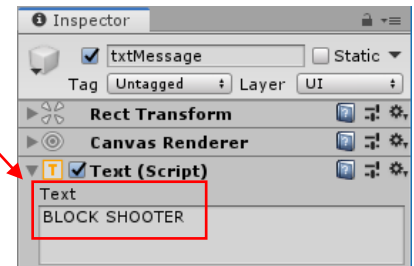


### 【学習項目】

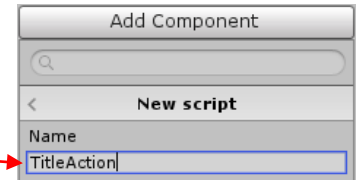
ゲームの画面は静止画になってはイケない！という UI 理論があります。  
ハングアップして何を押しても動かない故障状態？と思われるのを防ぐ為と言われていますが、ユーザーは動くものに目を取られるという心理的な効果も狙っていて、次に何をすべきか？を誘導する効果もあると言われています。



- ヒエラルキー欄の **txtMessage** を選択し、インスペクタのテキスト内容を設定します。



- ヒエラルキー欄の **GameController** を選択し、インスペクタの Add Component から **New script** を選びます。  
スクリプトの名称を **TitleAction** とします。



- スクリプト **TitleAction** を次のように編集します。

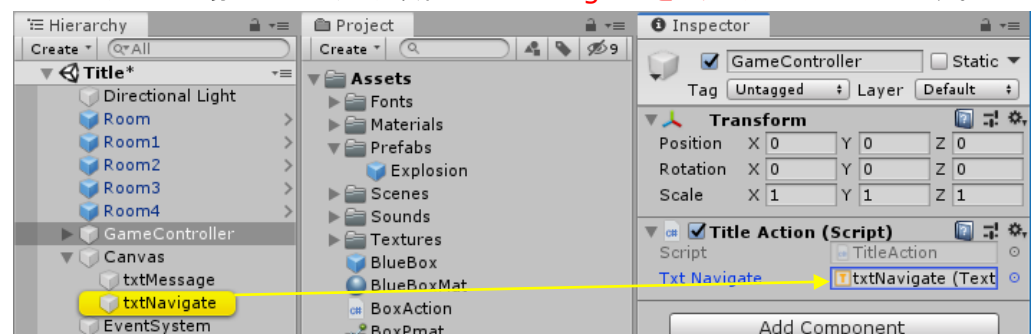
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを用いるのに必要
using UnityEngine.SceneManagement; //シーン切り替えに必要

public class TitleAction : MonoBehaviour {

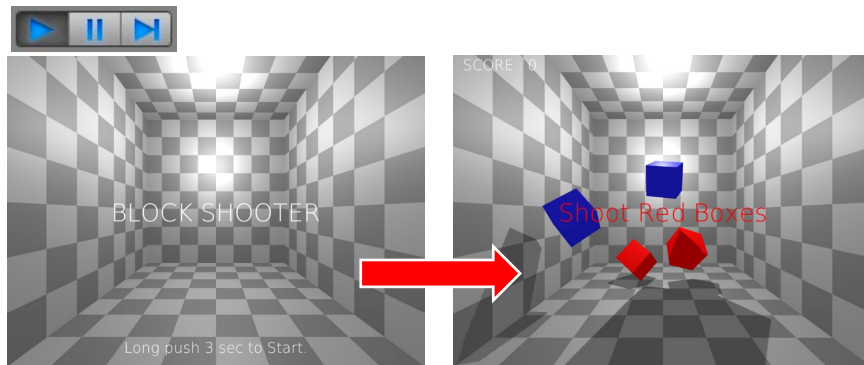
    float LongPush = 0.0f;
    float Elapsed = 0.0f;
    public Text txtNavigate;

    void Update () {
        //メッセージの点滅運営
        Elapsed += Time.deltaTime;
        Elapsed %= 1.0f;
        txtNavigate.text = ( Elapsed < 0.8f ) ? "Long push 3 sec to START" : "";
        //長押し3秒の検出
        if ( Input.GetMouseButton( 0 ) ) {
            LongPush += Time.deltaTime;
            if ( LongPush > 3.0f ) {
                SceneManager.LoadScene( "Main" );
            }
        } else {
            LongPush = 0.0f;
        }
    }
}
```

- GameController のインスペクタに登場したパブリック項目に **txtNavigate** をドラッグ & ドロップします。



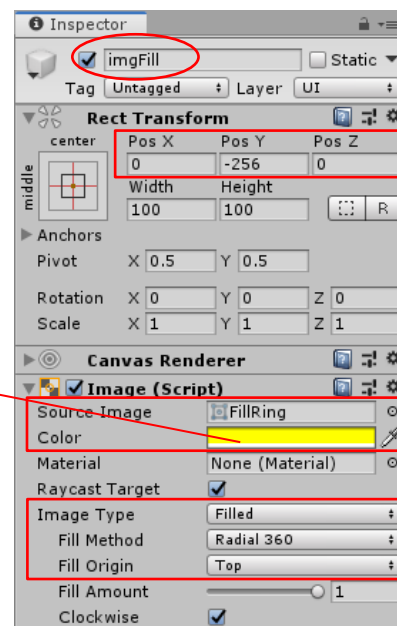
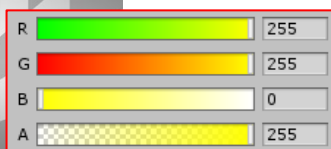
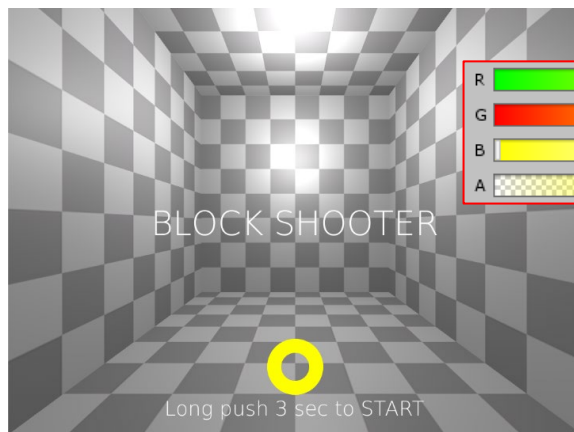
- プレイボタンを押下します。  
起動音と共にタイトル画面が表示され、メッセージが点滅します。画面を長押し3秒し、Main 画面へ遷移してゲームが始まります。



### 【STEP15】 長押し3秒の可視化

ユーザーにとって、画面の長押し3秒が可視化されていない為に、本当に計測してくれているかが不安な側面があります。3秒間を所要して増えていくグラフィック要素で、進捗を表すプログレスバーなどが基本的なインターフェースとなります。

- ヒエラルキー欄の Create から UI > **Image** を選択し、名称を **imgFill** と命名します。  
インスペクタでパラメータを設定します。



- スクリプト **TitleAction** を次のように編集します。

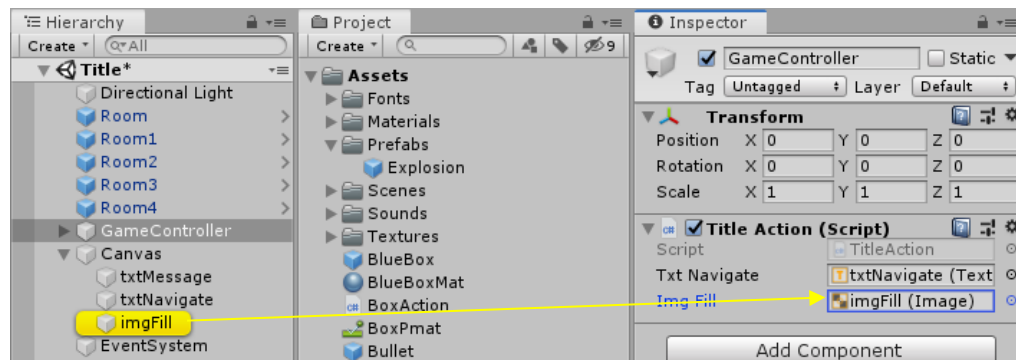
//～前略～

```
public Image imgFill;
```

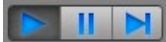
```
void Update () {  
    imgFill.fillAmount = LongPush / 3.0f;
```

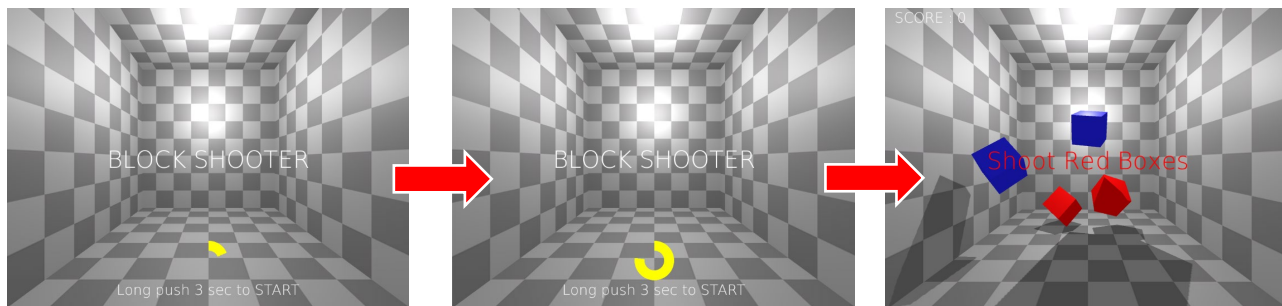
//～後略～

- ヒエラルキー欄の **GameController** を選択し、インスペクタに登場した項目を設定します。





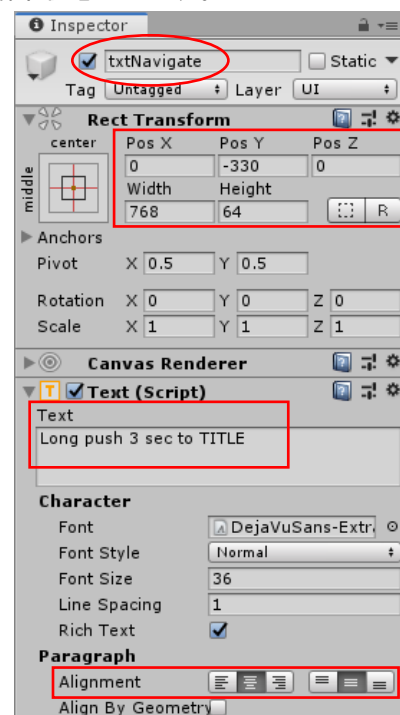
- プレイボタンを押下します。
- 長押し3秒が円グラフで可視化されています。



## 【STEP16】 タイムアップの判定

制限時間が尽きたら、単に停止するだけでなく、「TIME UP」を明示後、一瞬メッセージが消え、すぐに「Your score is XXX」がされ、タイトルへ戻る押下指示が画面下部に表示されるようにします。

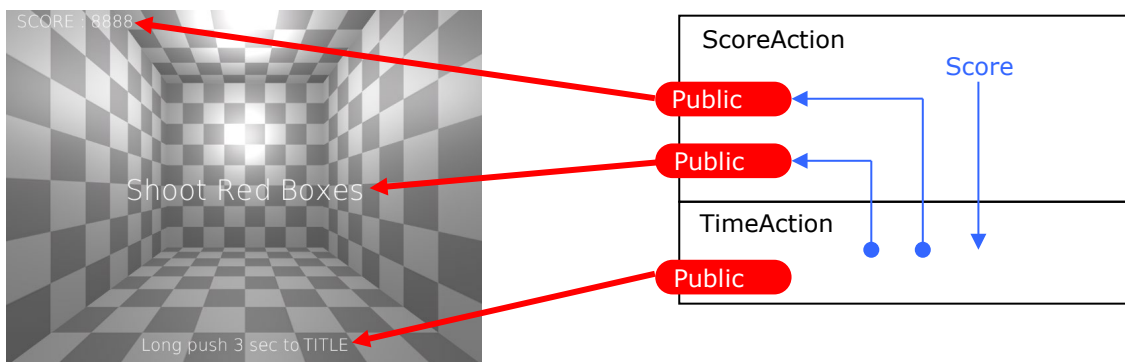
- 今まで作ってきたシーン Title を保存し、シーン **Main** をダブルクリックして編集状態にします。
  - ヒエラルキー欄の Canvas > **txtScore** を複製 (Ctrl + D) し、名称を **txtNavigate** とします。
- インスペクタでパラメータを設定します。



## 【学習項目】

隣のスクリプトの変数を参照したい時があります。様々な手法がある中から、今回はスクリプトと同じ型の入れ物を用意し、本当に取得して入れてしまい、そのスクリプトが扱う変数をそのまま使う手法を採用しました。

また、1つの文字表示エリアを複数のスクリプトで扱うと、どこに不具合があるのか？ が判りにくなる為、できるだけ文字表示の経路は一つにすることに貢献できています。





- スクリプト **TimeAction** を次のように編集します。

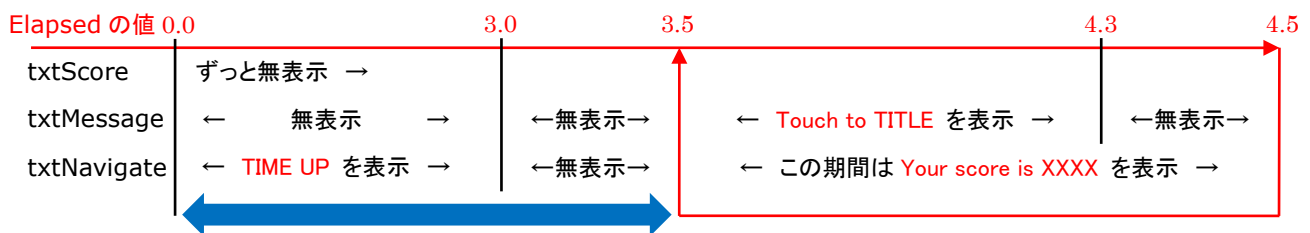
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを用いるのに必要

public class TimeAction : MonoBehaviour {

    public float TimeLimit = 60.0f; //制限時間
    float Elapsed; //経過時間
    bool isPlaying; //プレイ中か？の真偽値
    ScoreAction SA; //隣のスクリプトScoreAction
    bool CanLoad = false; //タイトルをロードできるかどうか
    public Text txtNavigate;

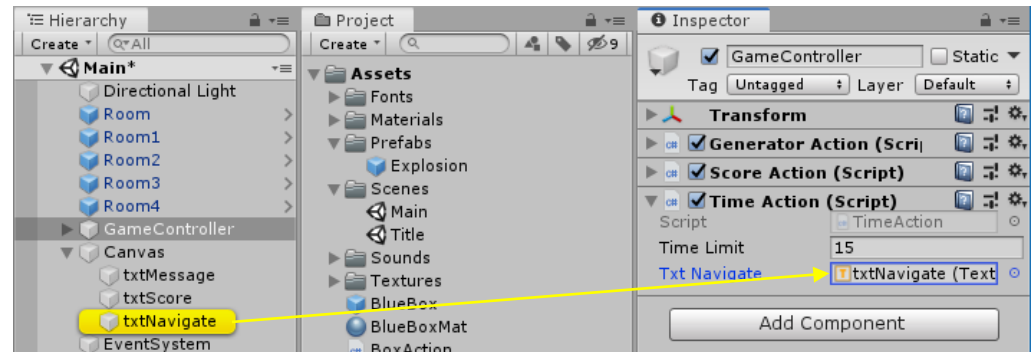
    void Start () {
        Elapsed = 0.0f;
        isPlaying = true;
        //隣のスクリプトScoreActionを取得する
        SA = GetComponent<ScoreAction>() as ScoreAction;
        txtNavigate.text = "";
    }

    void Update() {
        Elapsed += Time.deltaTime;
        if (isPlaying) {
            if (Elapsed >= TimeLimit) {
                //制限時間を超えたらTimeUpを放送する
                BroadcastMessage( "TimeUp", SendMessageOptions.DontRequireReceiver );
                isPlaying = false; //プレイ中を偽にする
                Elapsed = 0.0f;
                SA.txtMessage.color = Color.white;
                SA.txtMessage.text = "TIME UP";
                SA.txtScore.enabled = false;
            }
            else {
                if (Elapsed > 4.5f) { // > 4.5
                    Elapsed = 3.5f;
                }
                else if (Elapsed > 4.3f) { // 4.3---4.5
                    txtNavigate.text = "";
                }
                else if (Elapsed > 3.5f) { //3.5---4.3
                    CanLoad = true; //3.5秒以降はタイトルシーンをロード可能にする
                    txtNavigate.text = "Long push 3 sec to TITLE";
                    SA.txtMessage.text = "Your score is " + SA.Score;
                }
                else if (Elapsed > 3.0f) { // 3.0---3.5
                    SA.txtMessage.text = "";
                }
            }
        }
    }
}
```



【学習項目】プレイヤーはゲーム終了時に画面をタッチしまくっています。この操作が何かの YES の意思にならない配慮が必要です。操作受付不可となる期間を設けて、プレイヤーをクールダウン(熱気を冷却)しています。

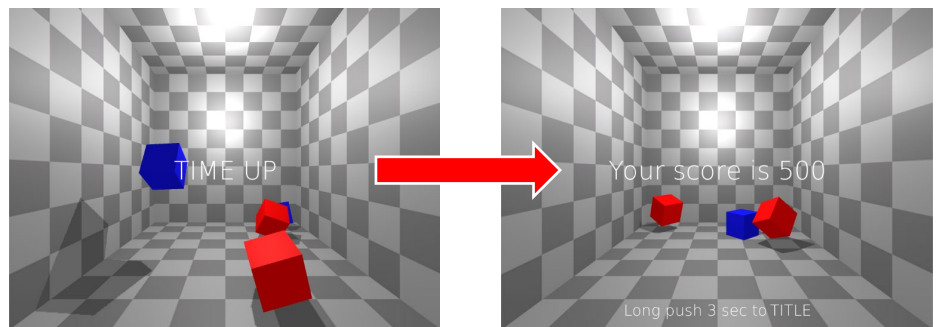
- ヒエラルキー欄の GameController をクリックし、インスペクタの TimeAction に現れた項目に **txtNavigate** をドラッグ & ドロップします。



- プレイボタンを押下します。



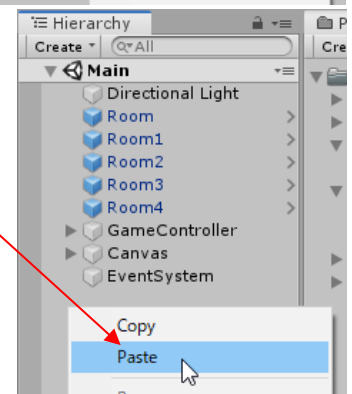
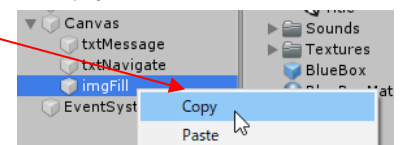
制限時間が尽きた時、**タイムアップ**の表示 → **無表示** → **スコア表示 & 下部メッセージ表示** となることを確認します。



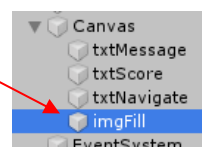
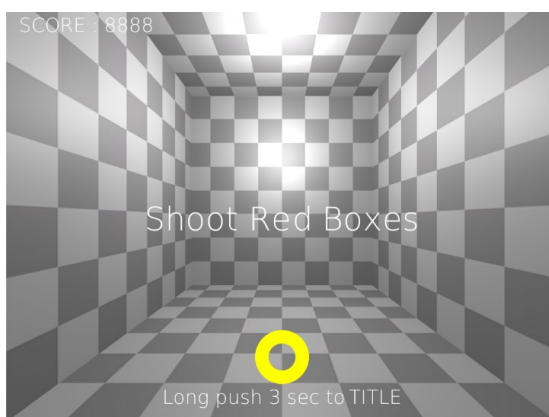
## 【STEP17】 タイトルへ戻る

タイムアップ後の画面で長押し3秒を受け付けて、シーン Title に切り替える運営を実装します。

- 現在のシーン Main を**保存 (Ctrl + S)**します。
- 黄色いリングをもらって来るので、シーン **Title** をダブルクリックして編集状態にします。
- ヒエラルキー欄の Canvas > **imgFill** を右クリックし、**Copy** を選択します。
- シーン **Main** をダブルクリックして編集状態にします。
- ヒエラルキー欄の空白部で右クリックし、**Paste** を選択します。



- 貼り付けた imgFill が **Canvas の配下構造**になるように組み変えます。



- 変数 CanLoad を持つスクリプトが適任なので、**TimeAction** を次のように編集します。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを用いるのに必要
using UnityEngine.SceneManagement; //シーン切り替えに必要

public class TimeAction : MonoBehaviour {

    float LongPush = 0.0f;
    public Image imgFill;

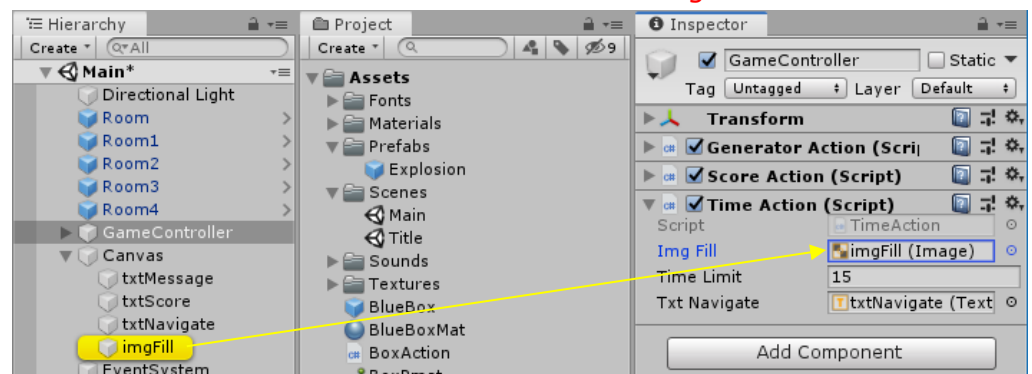
    //～中略～

    void Update() {

        //長押し3秒の検出
        if (Input.GetMouseButton( 0 ) && CanLoad) {
            LongPush += Time.deltaTime;
            if (LongPush > 3.0f) {
                //シーンTitleをロードする
                SceneManager.LoadScene( "Title" );
            }
        } else {
            LongPush = 0.0f;
        }
        imgFill.fillAmount = LongPush / 3.0f;
    }

    //～後略～
}
```

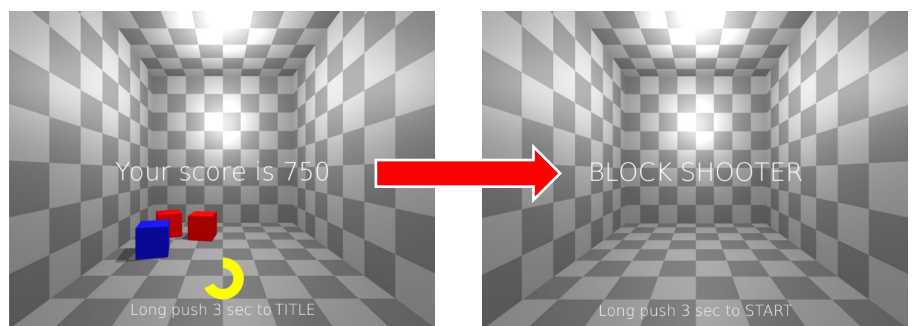
- ヒエラルキー欄の **GameController** を選択し、インスペクタに登場した項目に **imgFill** をドラッグ & ドロップします。



- プレイボタンを押下します。



タイムアップ後の画面で、3秒長押しをした時に、タイトル画面に戻れるか確認します。

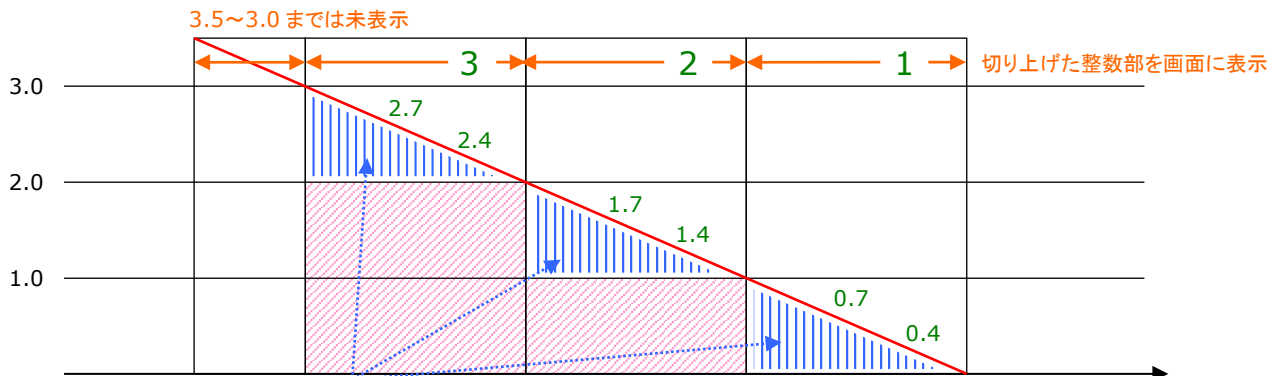


これでシーケンスが通った運営となりました。

## 【STEP18】 ゲーム開始時の演出

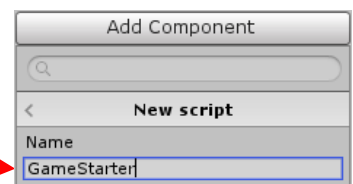
ゲームの開始時に、画面中央に 3 → 2 → 1 と、カウントダウンして始まる演出を加えます。

3秒間が減る表示ですが、最初は 0.5 秒間だけ未表示の期間とし、3.5 秒から時間が減っていく運営とします。その時間が減って行く様子をグラフにしたものが以下の通りです。



切り捨てた値(整数)を減算すると、三角形部分が残る。この減り具合をアルファ値(不透明度 0~1)として扱えば、徐々に消える効果となる。

- ヒエラルキー欄の GameController を選択し、インスペクタの Add Component から **New script** を選びます。  
スクリプトの名称を **GameStarter** とします。



- スクリプト **GameStarter** を次のように編集します。

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

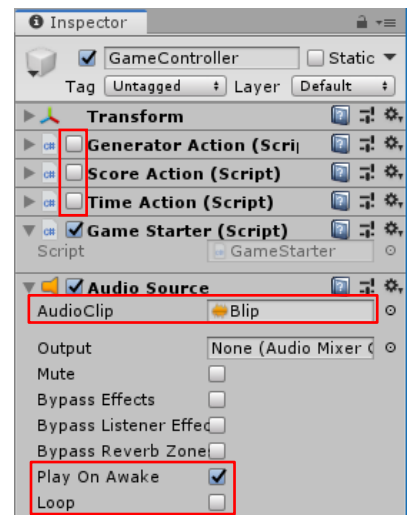
public class GameStarter : MonoBehaviour {

    ScoreAction SA; //隣のスクリプトScoreAction
    TimeAction TA; //隣のスクリプトTimeAction
    float Elapsed = 3.5f; //3.5 → 3.0の間は何も表示しない
    string Countdown = "";

    void Start() {
        SA = GetComponent<ScoreAction>() as ScoreAction;
        TA = GetComponent<TimeAction>() as TimeAction;
        SA.txtMessage.text = "";
        SA.txtScore.text = "";
        TA.txtNavigate.text = "";
        TA.imgFill.fillAmount = 0.0f;
    }

    void Update() {
        Elapsed -= Time.deltaTime;
        if (Elapsed <= 0.0f) {
            BroadcastMessage("GameStart",
                SendMessageOptions.DontRequireReceiver); //ゲームの開始を放送する
            enabled = false; //このスクリプトは停止
        } else if (Elapsed < 3.0f) { //0.5秒経過したらカウントダウン表示を行う
            Countdown = "" + Mathf.Ceil(Elapsed); //切り上げた整数部を表示する
            SA.txtMessage.text = Countdown;
            //経過時間 - 切り捨てた整数部 = 小数部(1.0~0.0)これを不透明度に使う
            SA.txtMessage.color = new Color(1, 1, 1, Elapsed - Mathf.Floor(Elapsed));
            //文字の大きさにも使っちゃいました。だんだん小さくなります。
            SA.txtMessage.fontSize = Mathf.FloorToInt((1 + (Elapsed - Mathf.Floor(Elapsed))) * 60);
        }
    }
}
```

- ゲーム開始はこのスクリプトに一任するので、他のスクリプトは停止します。  
ヒエラルキー欄の **GameController** を選択し、パラメータを設定します。



- Add Component から **Audio Source** を選択し、パラメータを設定します。

- 同様に MainCamera の持つスクリプト **GunAction** も停止します。



これらオフにしたスクリプト達は、カウントダウンによる **GameStart** という放送を聞いて、オンになる構造に改造します。

- それら停止したスクリプト全てに次の処理を加えます。チェックオフでも放送は聞こえており、処理は可能です。

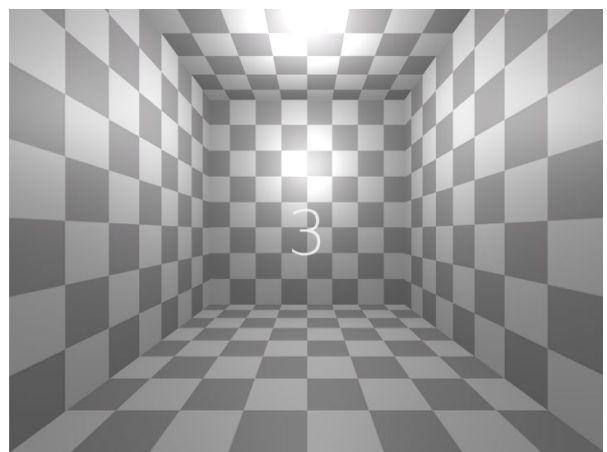
- **GeneratorAction**
- **ScoreAction**
- **TimeAction**
- **GunAction**

```
//～前略～

void GameStart () {
    enabled = true; //スクリプトを稼働させる
}

//～後略～
```

- プレイボタンを押下します。  
カウントダウンが行われてからゲームが始まることを確認します。

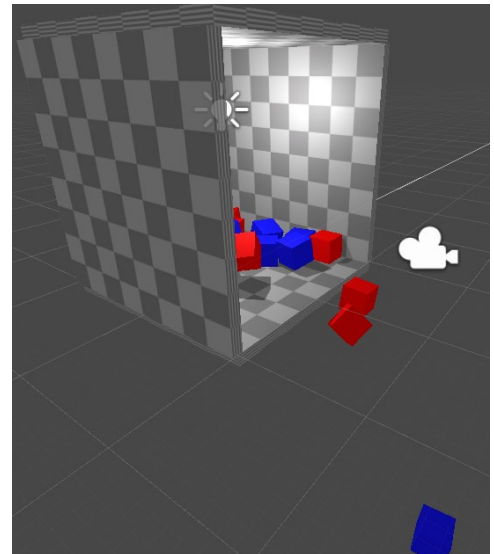


## 【STEP19】 レイヤーによる衝突のグループ分け

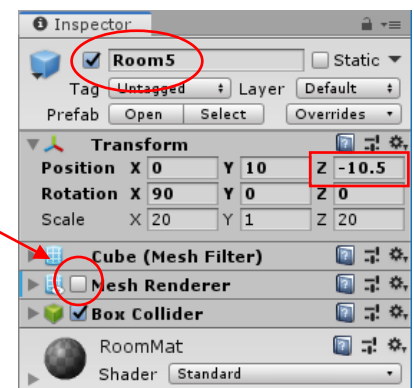
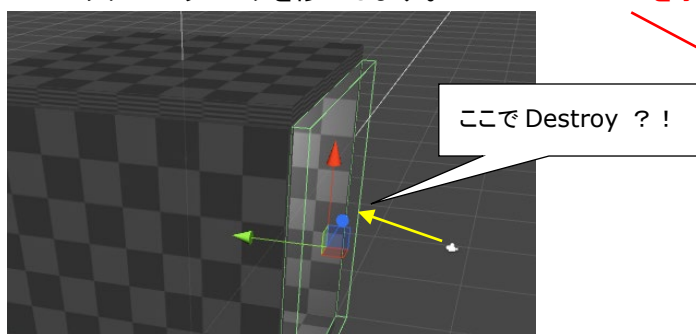
実は、箱は部屋からあふれ出ている、永遠の落下を続ける可能性があります。管理上はあまり良く無い事です。

対策として、①マイナス10くらいの高さに至ったら自身でデストロイする仕組み、②大きな受け皿の箱を置いて、落ちてその箱に当たったらデストロイする仕組み、③10秒経ったら箱が自身でデストロイするタイマーを持つなど、様々なアイデアが考えられます。

今回の施策では、そもそも箱が部屋からあふれない、という仕組みを実装します。ルールそのものが箱を消して得点していく内容なので、箱の破壊(デストロイ)はプレイの中で達成されるため、あふれない仕組みだけで構築できるはずです。これの実現の為に「レイヤーによる衝突のグループ分け」を理解する必要があります。

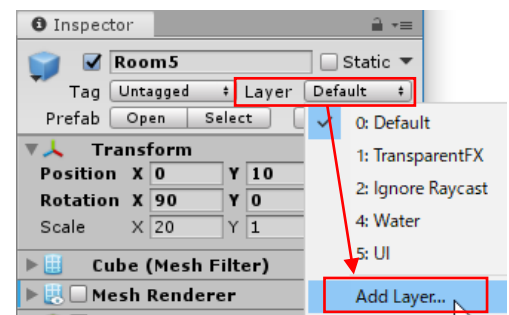


- ヒエラルキー欄の Room4 を複製(Ctrl + D)し、Room5 と命名します。インスペクタでパラメータを修正します。**MeshRenderer をオフ**にします。

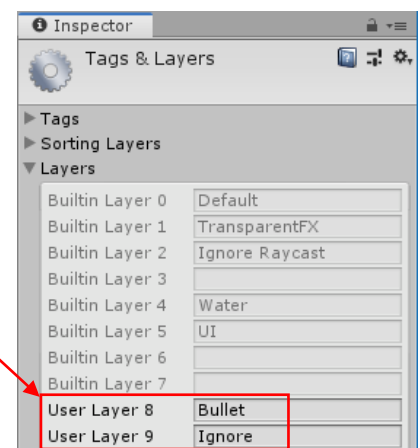


このままではカ Bullet が Room5 に当たった瞬間に破壊されて部屋の中に侵入出来ません。弾 Bullet は Room5 を無視するような物理判定ルールが必要になります。

- 衝突のグループ分けをするレイヤーを作ります。Layer(レイヤー)から **Add Layer...** を選択します。

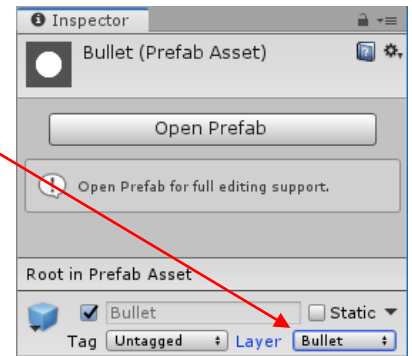


- 新しいユーザー作のレイヤーとして **Bullet, Ignore**(無視)を作成します。

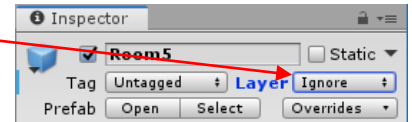




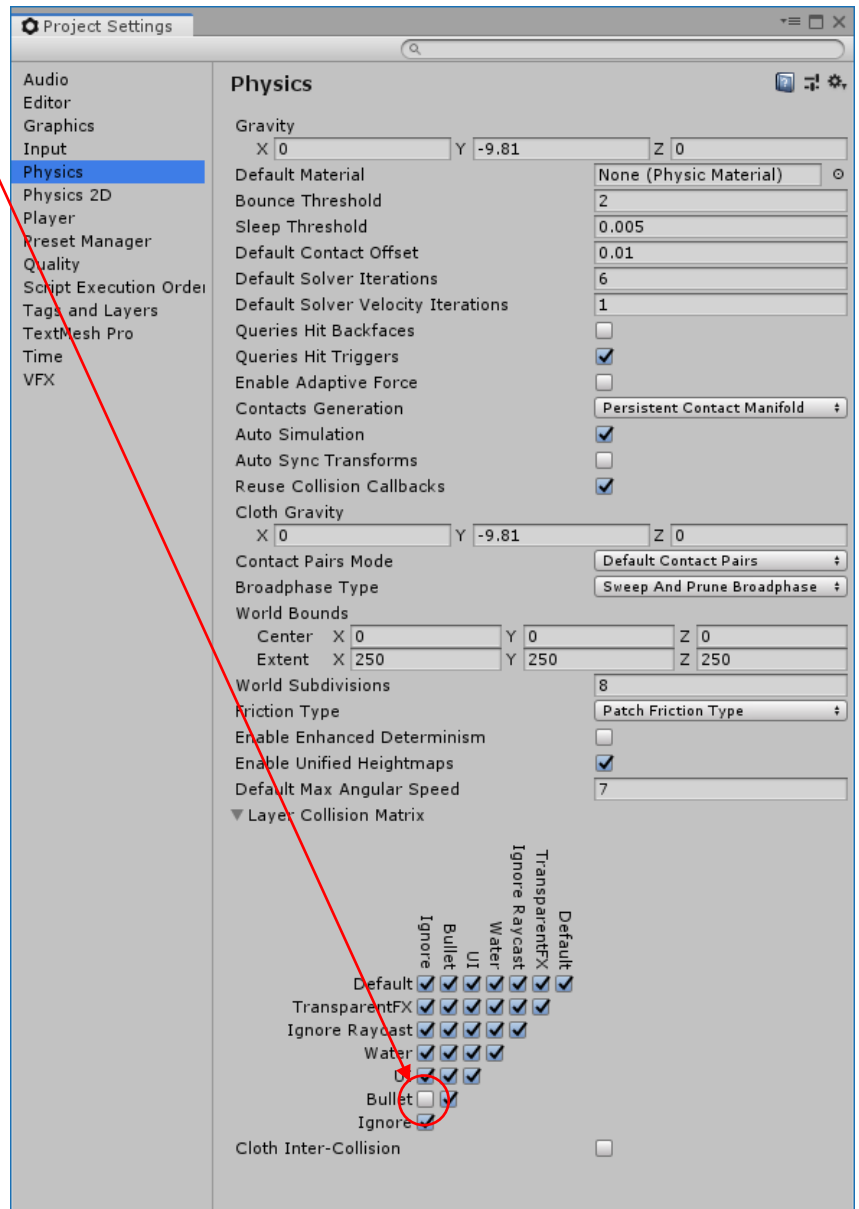
- プロジェクト欄のプレハブ **Bullet** を選択し、**レイヤー**を **Bullet** に指定します。



- ヒエラルキー欄の **Room5** を選択し、**レイヤー**を **Ignore** に指定します。  
Room5 は無視対象というわけです。



- メニューEdit > **Project Settings** を選択します。
- レイヤー衝突マトリックスで、Bulletレイヤーに属する物体が Ignoreレイヤーに属する物体に対して物理衝突演算を行わないよう、**チェックをはずします**。



これで弾はRoom5を無視して部屋に入るものの、赤と青の箱はRoom5を無視できずに出られない！という訳です。