小学生から始める

プログラミング教室

Kid's programmer -master course-

アドバンスコース

～Assetsとライセンス～

サンプル

1回目授業　　年　　月　　日

2回目授業

名前

KID's プログラミングラボ

ICT活用教育アドバイザー

◆文科省　ICT活用教育アドバイザー事務局掲載
学校ICT化サポート事業者

◆長期コースによる、プログラミングの普及

0 目標

・UnityAssetsStoreとIT関連の法務について知る
・Assetsを利用する
・Prefabと Raycastについて知る

1 完成ファイルの確認

今回作成するゲームはFPS（ファーストパーソンシューター）ゲームです。シューティングゲームの一種でキャラクターの本人視点ゲームが進行され、画面に表示されるのはキャラクターの一部と武器や道具のみです。

世界的に人気があるジャンルであり、高いリアリティを求められることがあります。そのため、先端CG技術が実験的に使われることもあります。

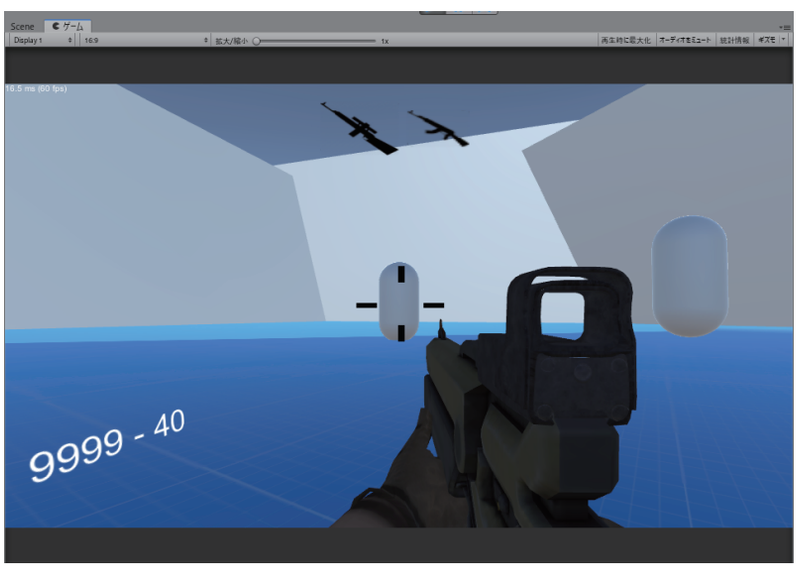
1.0 動作の確認

実行すると2つのゲームモードを選択することができます。「射的モード」と「迷路脱出モード」です。射的モードは限られた時間と弾でターゲットを撃ち、ハイスコアを目指します。「迷路脱出モード」は巨大迷路を進み、脱出を目指します。

途中には襲い掛かるターゲットがいるのでターゲットを倒しながら進みます。

1.0.0 動作説明

WASD：移動	左クリック：銃を撃つ	Shit：ダッシュ	マウスホイール：武器の切り替え
Space：ジャンプ	右クリック：サイトを覗く	R：リロード	



1.1 Assets Store

Assets Store(アセットストア) とは、Unity で使用できる 3D モデルの素材や画像などを購入できるショップです。ゲーム素材だけではなく、開発効率を上げるためのプラグインも扱われています。

利用するだけではなく、作成した作品を Assets Store に公開することもできます。

今回作成するゲームの FPS のシステムは Assets Store からダウンロードした Assets が含まれています。

1.1.0 今回利用する Assets

今回のゲームには「Easy FPS」という Assets を利用しました。FPS ゲームを作ることはとても大変なことですが、この Assets を使うことにより簡単に FPS ゲームを作ることができます。

1.1.1 ライセンス

Assets Store の利用にはライセンス（利用規約）に則る必要があります。利用できる範囲や、利用にあたっての有償、無償、再配布の可否などのライセンスは Assets によって異なることがあるのでライセンスをよく読んでから利用する必要があります。

基本的に Assets が簡単に取り出せる状態での再配布（ビルドせずにプロジェクトデータを配布するなど）は許可されていません。

しかし、今回は特別に作成者から許可を得て「Easy FPS」を配布しています。

Unity が最も人気のあるゲーム開発のソフトウェアである理由の一つとして Assets Store の存在があります。初心者でも Assets を利用することによって簡単にゲームを作ることができるからです。

Unity に限らず、日頃利用している高品質なサービスはライセンスが守られているからこそ利用できているのです。サービスを利用するためだけではなく、トラブルに巻き込まない、巻き込まれないようにするためにもライセンスはしっかり守りましょう。

2 練習ファイルの確認

ゲームを作成していく前に「Easy FPS」の機能を確認しましょう。

2.0 動作の確認

1日目練習フォルダを開いて実行しましょう。今回はAssetsを利用していますので初めからある程度ゲームが出来る状態ですが、足りない機能や表示、気づいたことを書き出してみましょう。

実行するとマウスポインターの表示が消えます。

マウスポインターを再表示させるには「Esc」キーを押しましょう

2.1 ディレクトリとファイルの確認

「Easy FPS」のディレクトリと使用するファイルを確認しましょう。Easy FPSで使用するファイルはProjectウインドウのEasy FPSフォルダに格納されています。

3 弾数の変更と保存

画面左下に表示されている数字は総弾数と装備中の武器のマガジン内の弾数を表しています。つまり、総弾数は9999発、マガジン内は40発あるということです。

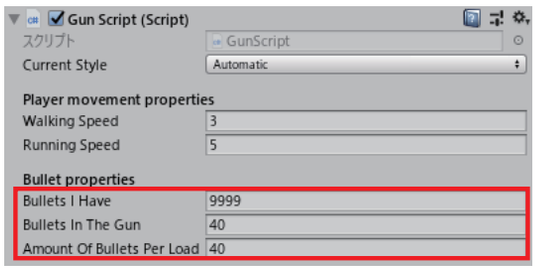
現在は武器の種類が変わっても総弾数とマガジン内の弾数は変わりません。また、武器を切り替えると減った弾数がリセットされてしまいます。

3.0 仕様の確認

弾数の管理方法と武器を切り替えるとリセットされてしまう原因を探します。

3.0.0 弾数の管理方法

弾数は Resources フォルダ内の「NewGun_auto」、「NewGun_semi」にアタッチされている GunScript の以下の赤枠部分で設定されています。



bulletsIHave : 総弾数
bulletsIntheGun : 初期のマガジン内の弾数
amountOfBulletsPerLoad : リロード時にマガジンに装填される弾数

「NewGun_auto」と「NewGun_semi」の変数の値を変更して弾数が変わるか確かめてみよう！

3.0.1 弾数がリセットされる原因

弾数がリセットされる原因は武器の切替方法にあります。シーンを実行してヒエラルキーに注目しながら武器を切り替えてみましょう。

武器を切り替えることによってオブジェクトの削除と生成が行われていますね。弾数がリセットされてしまうのはこの動作が原因です。オブジェクトの削除、生成が行われるとどうして弾数がリセットされてしまうのか考えてみましょう。

考えてみよう！

3.1 弾数がリセットされないようにする

弾数がリセットされてしまうのは武器を切り替える度にオブジェクトの削除と生成をすることによってオブジェクトにアタッチされている「GunScript」も削除されてしまっているためです。

3.1.0 弾数がリセットされない仕組み

弾数がリセットされないようにするには「GunScrpt」ではないスクリプトで管理することによって武器切替によってGunScrpt が削除されても弾数がリセットされません。

3.1.1 スクリプトの作成とアタッチ

弾数を管理する「Player」スクリプトを作成しましょう。また、作成したスクリプトはシーン内の「Player」オブジェクトにアタッチしてください。

3.1.2 スクリプトの編集

Player スクリプトを以下のように変更しましょう。

Player.cs

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

public class Player : MonoBehaviour
{
    public float[,] Bullet = new float[,] { {50,10,10},{100,40,40} };
    void Start()
    {
```


弾を２次配列で管理しています。一つ目の列はNewGun_semi、二つ目の列はNewGun_autoの弾数です。要素は左からbulletsIHave、bulletsIntheGun、amountOfBulletsPerLoadとなるようにします。

3.1.3 現在の武器の取得

現在の装備している武器によって読み込む配列を変更する必要があります。現在装備している武器は「GunInventory」スクリプトの変数「currentGunCounter」で管理しています。

この変数はアクセス修飾子がprivateになっています。GunScrpt から値が取得できるようにpublicに変更しましょう。

GunInventory.cs

```
public class GunInventory : MonoBehaviour {
      [Tooltip("Current weapon gameObject.")]
      public GameObject currentGun;
      private Animator currentHAndsAnimator;
      public int currentGunCounter = 0;
```

GunScript を以下のように編集して現在装備している武器を取得できるようにしましょう。変更後、武器を切り替えるとコンソールにどのような値が出力されるか確認しましょう。

GunScript.cs

```
private GunInventory gis;
void Awake(){
    gis = GameObject.Find("Player").GetComponent<GunInventory>();
    Debug.Log(gis.currentGunCounter);
    mls = GameObject.FindGameObjectWithTag("Player").GetComponent<MouseLookScript>();
```

NewGun_semi のとき出力された値 ＿＿＿＿　NewGun_auto のとき出力された値 ＿＿＿＿

3.1.4 Player スクリプトの弾数を取得と反映

GunScrpt が実行されたときは Player スクリプトの変数 Bullet を参照し、クリックによって弾を消費したときは残弾を Player スクリプトの変数 Bullet に反映します。

以下のように変更しましょう。

GunScript.cs

```
private GunInventory gis;
private Player ps;
void Awake(){
    gis = GameObject.Find("Player").GetComponent<GunInventory>();
    ps = GameObject.Find("Player").GetComponent<Player>();
    Debug.Log(gis.currentGunCounter);
    //弾数の取得
    bulletsIHave = ps.Bullet[gis.currentGunCounter, 0];
    bulletsInTheGun = ps.Bullet[gis.currentGunCounter, 1];
    amountOfBulletsPerLoad = ps.Bullet[gis.currentGunCounter, 2];
～～～～～～～～～～～～～～～省略～～～～～～～～～～～～～～～
private void ShootMethod(){
     if(waitTillNextFire <= 0 && !reloading && pmS.maxSpeed < 5){

          if(bulletsInTheGun > 0){

～～～～～～～～～～～～～～～省略～～～～～～～～～～～～～～～
          else{
               //if(!aiming)
               StartCoroutine("Reload_Animation");
               //if(emptyClip_sound_source)
               //    emptyClip_sound_source.Play();
          }
     }
    //残弾の反映
    ps.Bullet[gis.currentGunCounter, 0] = bulletsIHave;
    ps.Bullet[gis.currentGunCounter, 1] = bulletsInTheGun;

}
```

Check! できているかチェックをしよう！

- ☐ 武器を切り替えても弾数がリセットされない

4 エフェクトの作成

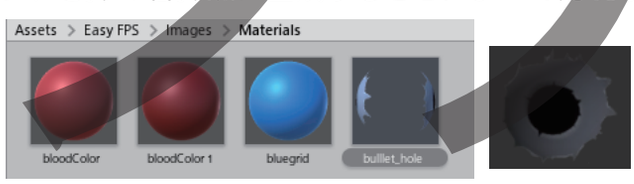
弾オブジェクトと発射したオブジェクトの弾が特定のオブジェクトに衝突した際のエフェクトを作成します。

4.0 エフェクトの作成

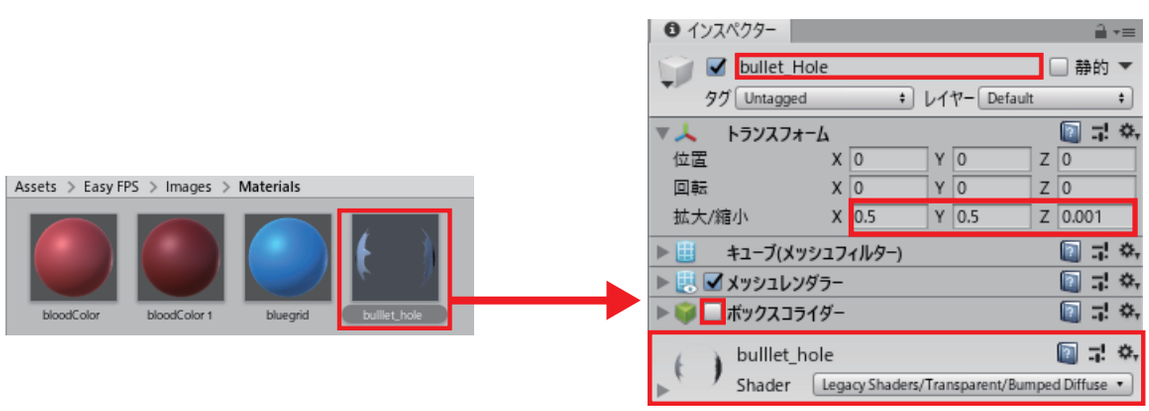
弾が着弾したときのエフェクトから作成していきます。弾は壁に当たると左下図のように壁に穴があき、シーン内のカプセルオブジェクトに当たると複数の光の粒が出現し消滅します。

4.0.0 穴があくエフェクトの作成

穴が開くエフェクトは [Images] の bullet_hole という穴のあいたような画像が適用されたマテリアルを使用したオブジェクトを弾の着弾点に生成することによって再現します。



まずはマテリアルをアタッチするオブジェクトを作成します。ヒエラルキーから [作成] > [3D オブジェクト] > [キューブ] を選択し、以下のように変更してください。



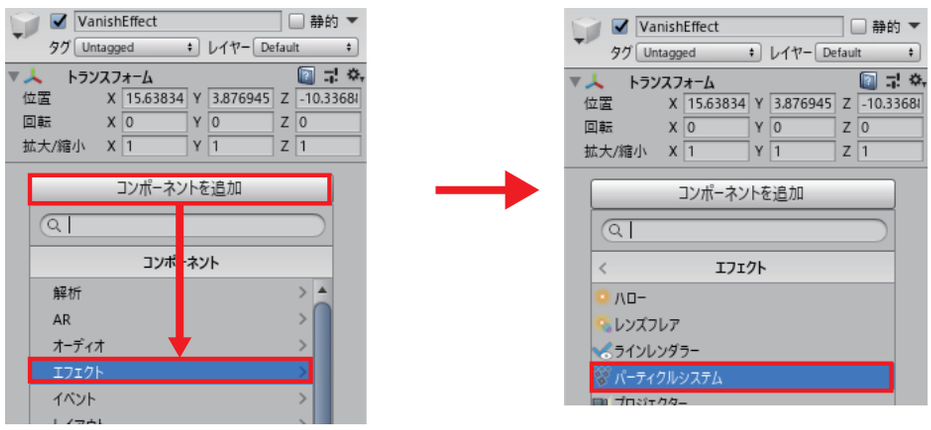
マテリアルは [Images] > [Materials] にあります。ドラッグ＆ドロップでアタッチしましょう。

4.1 消滅エフェクトの作成

消滅のエフェクトはパーティクルシステムというコンポーネントを使用して作成します。

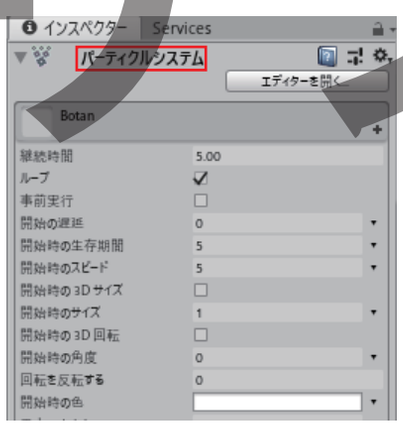
4.1.0 オブジェクトの作成とコンポーネントの追加

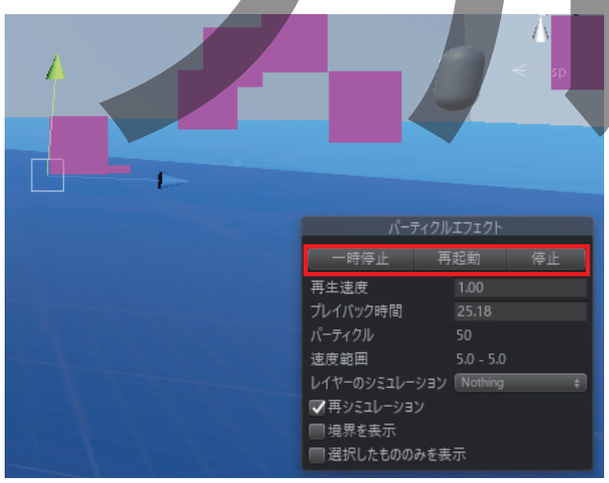
パーティクルは小さな画像をたくさん発生させるシステムです。これをオブジェクトに追加します。空のオブジェクト「VanishEffect」を作成し、以下の手順でパーティクルを追加してください。



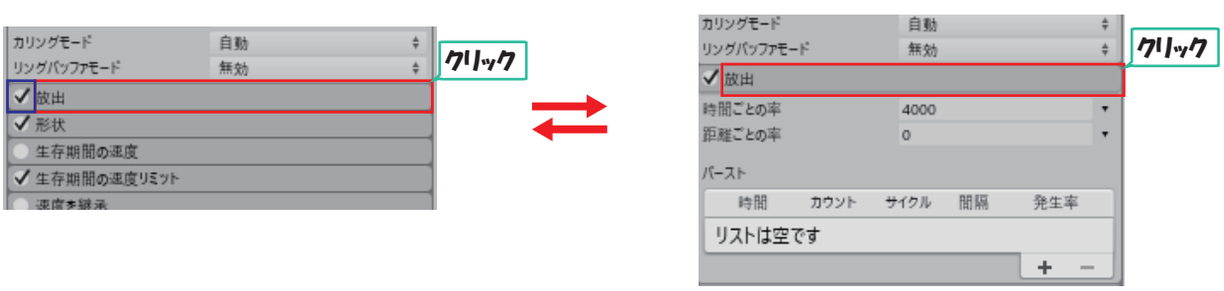
「VanishEffect」オブジェクトを選択し、インスペクターを選択するとパーティクルが追加されたことが確認できます。

ゲーム、シーンビューではポコポコと放出されている紫色のものがありますね。これがパーティクルです。また、シーンビューではパーティクルの再生、停止が行えるようになっています。





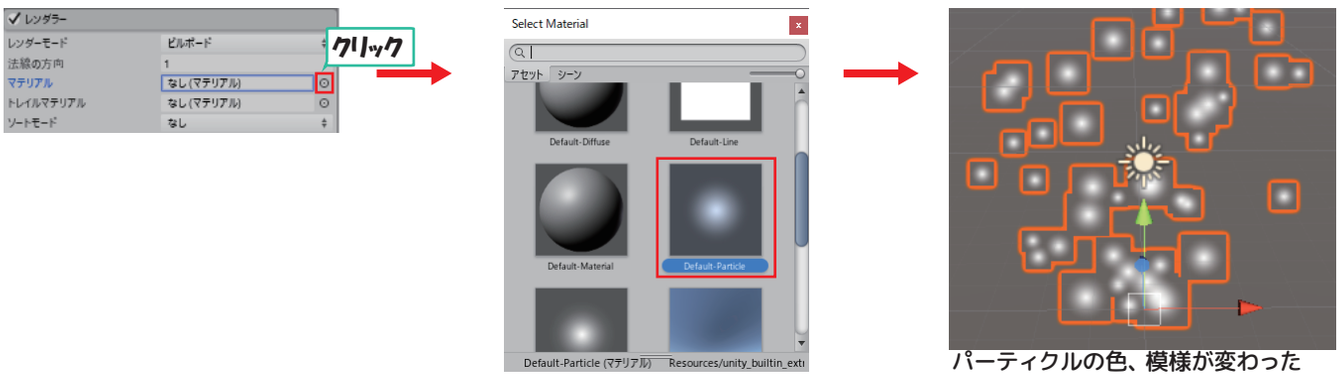
パーティクルシステムコンポーネントは多くのプロパティを有しており、折り畳み式のセクション、「モジュール」へとまとめられています。モジュール名をクリックすることにより開閉することができます。



各モジュールの有効にするにはモジュール名横（青枠）にチェックをいれる必要があります。

4.1.1 レンダラーモジュールの設定

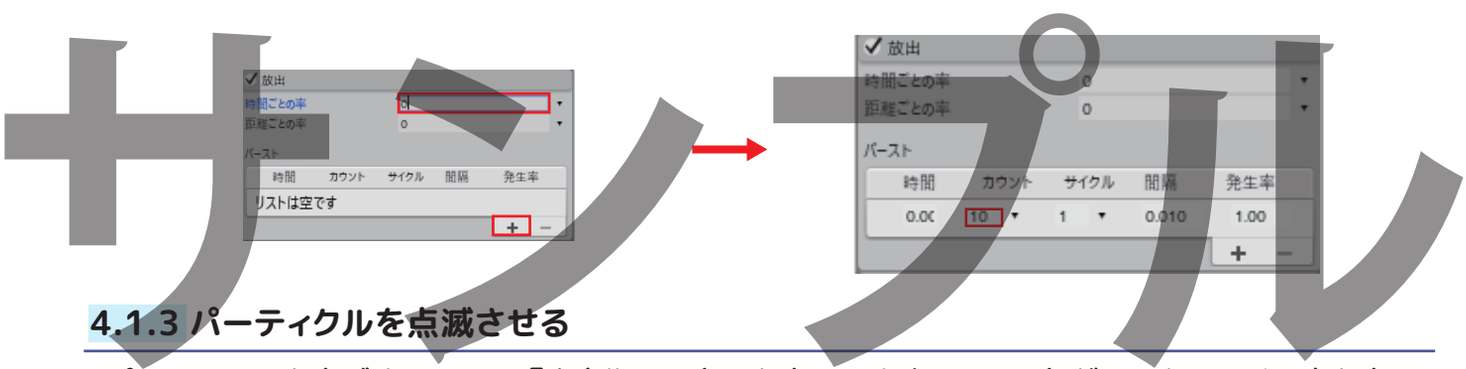
パーティクルが紫色になっているのはパーティクルのマテリアルがセットされていないためです。レンダラーモジュールで設定できます。以下のように変更しましょう。



パーティクルの色、模様が変わった

4.1.2 放出モジュールの設定

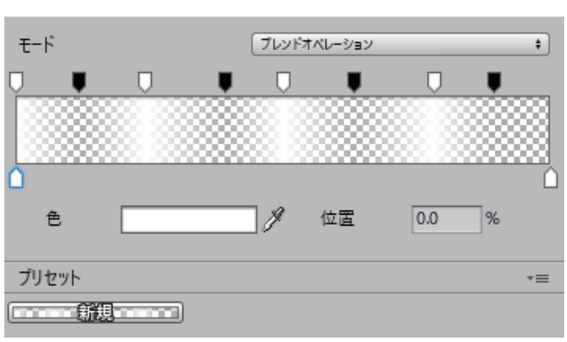
パーティクルが絶え間なく放出されていますね。生成された瞬間に特定の数だけ発生させるために以下のように変更しましょう。

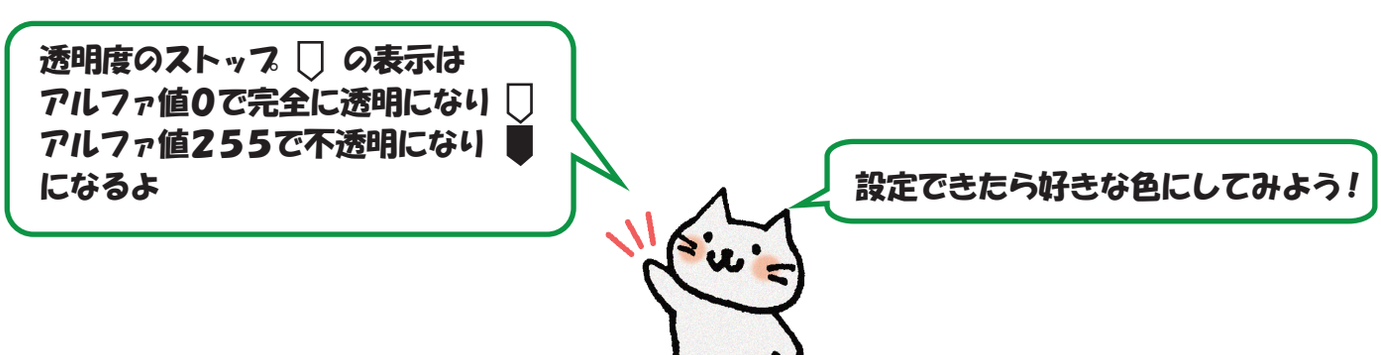


4.1.3 パーティクルを点滅させる

パーティクルを点滅させるには「生存期間の色」を変更します。これは色だけでなく、透明度を変更できるので透明、不透明を繰り返すことによって点滅しているように見せます。

以下に設定例と次のページで操作説明を記載するので操作説明を読んでから設定してみましょう。



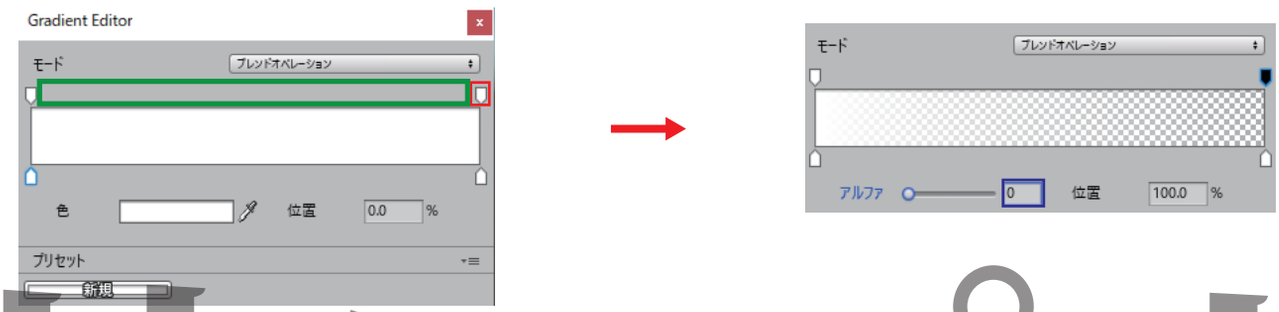


操作、設定方法

色をクリックすると以下のような画面が現れます。これは「Gradient Editor」といい、色や透明度を徐々に変更することができます。

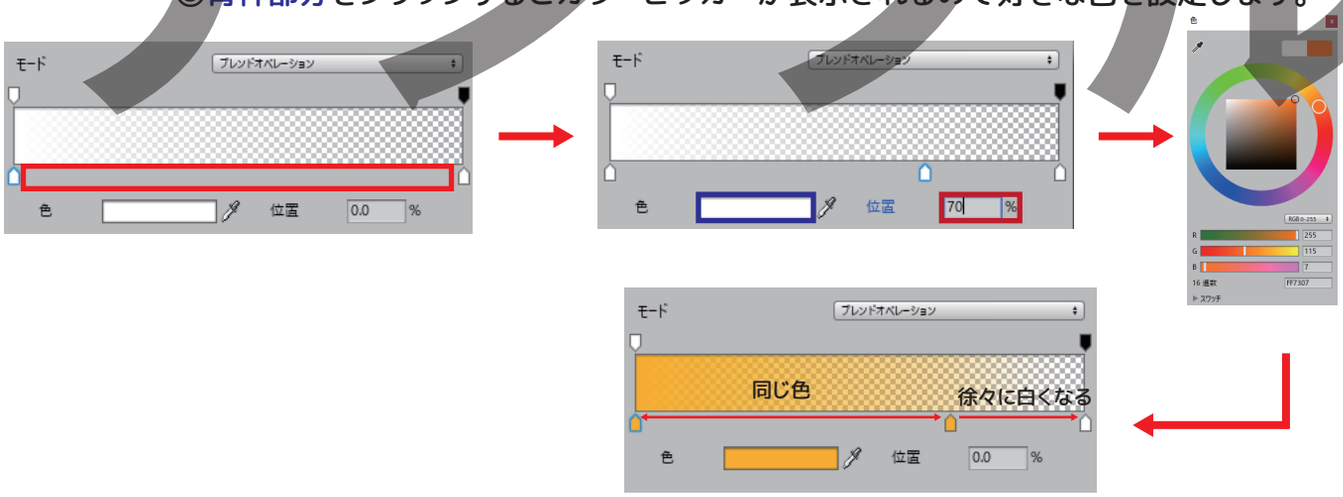
透明度の変更方法

①左下赤枠部分をクリックします。これはストップと呼ばれるものです。
②右下青枠部分を０にします。これは透明度を表しており、０～２５５の間で指定します。
　値が小さいほど透明度は高くなり、０で完全に透明になります。
③新しい透明度のストップを追加する場合は緑色部分をクリックします



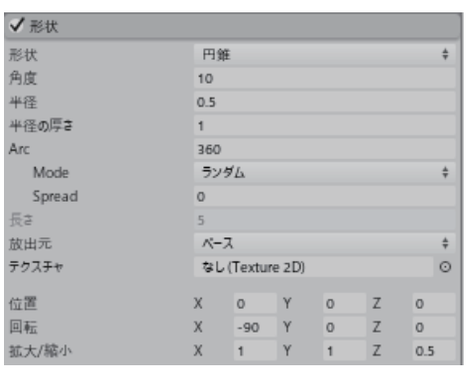
色の変更方法

①左下赤枠部分をクリックし色のストップを追加します。
②新しく追加したストップを選択し、ドラッグで移動できます。
③青枠部分をクリックするとカラーピッカーが表示されるので好きな色を設定します。



4.1.4 放出時の形状の変化

パーティクルを放出する際の形状を変更します。以下のように変更しましょう。

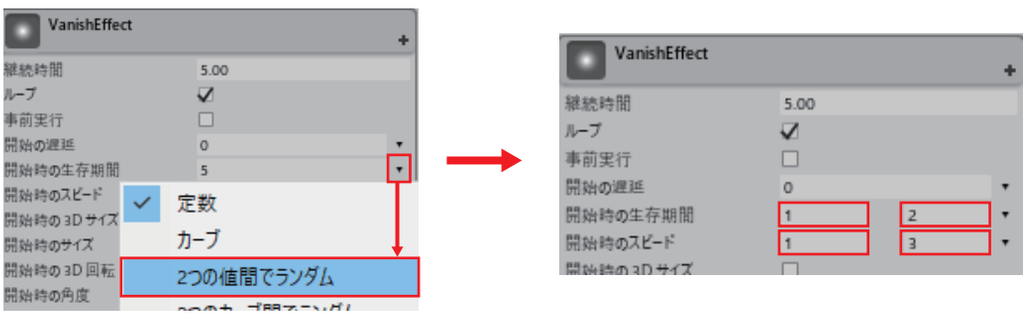


4.1.5 エフェクトに無作為性を持たせる

パーティクルが一塊で放出されていますね。それぞれのパーティクルの生存期間と速度に無作為性を持たせます。

項目名横のドロップダウンボタンをクリックし、[2つの値間でランダム]を選択すると指定した値間でランダムに動作させることができます。

右下図の赤枠と同じ値になるように変更しましょう。



4.2 エフェクトを削除する

エフェクトを削除するスクリプトを作成します。シーンに登場するエフェクトは再生終了後、削除する必要があります。

考えよう! 削除が必要な理由

例えばシューティングゲームではキーを押すたびに弾が増えていきます。すると、増えすぎた弾はコンピューターに負荷をかけてしまい、アプリケーションが異常終了する原因になったりします。

いくらプログラムといえど、使えるリソース(資源)は有限です。また、リソースもPCや環境により様々なため、幅広いパターンを想定して開発を行うようにしましょう。

4.2.0 スクリプトの作成とアタッチ

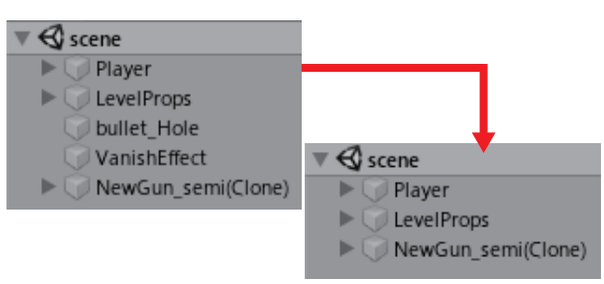
オブジェクトを削除するスクリプト「DestroyAfterTime」を作成し、以下のように変更しましょう。また、変更後はbullet_HoleとVanishEffectオブジェクトにアタッチします。

DestroyAfterTime.cs

```
using System.Collections;
using UnityEngine;

public class DestroyAfterTime : MonoBehaviour {
      public float timeToDestroy = 2f;
      void Start () {
            Destroy (gameObject, timeToDestroy);
      }
```

Check! できているかチェックをしよう!

- [ ] 実行すると2秒後にオブジェクトが削除される



4.3 火花エフェクトの作成

壁に着弾した際に黄色の火花も一緒に生成されていたのに気が付いたでしょうか。この火花のエフェクトもパーティクルで作られています。

4.3.0 オブジェクトの作成とコンポーネントの追加

空のオブジェクト「SparkEffect」を作成し、パーティクルシステムコンポーネントを追加しましょう。

4.3.1 パーティクルの基本設定

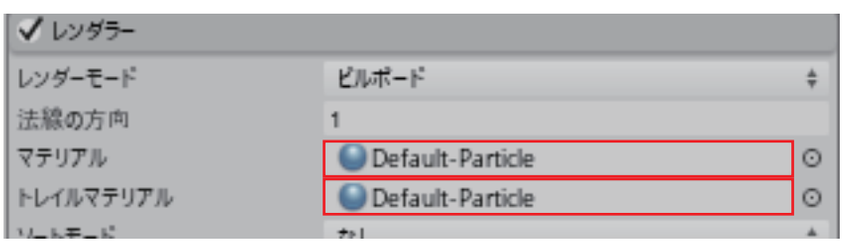
追加したパーティクルシステムの設定を以下のように変更しましょう。

4.3.2 放出モジュールの設定

消滅のエフェクトと同様にパーティクルは生成時に生成された瞬間に特定の数だけ発生させます。

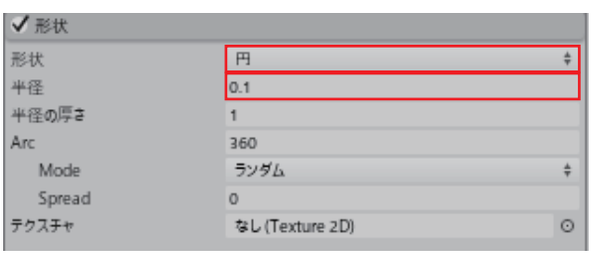
4.3.3 レンダラーモジュールの設定

マテリアルを設定します。今回はトレイルモジュールを使用するのでトレイルマテリアルも設定します。



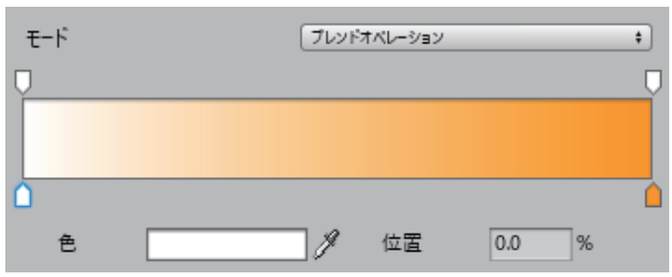
4.3.4 形状モジュールの設定

パーティクルを放出する形状を変更します。形状、角度などを変更することによってパーティクル放出時の方向を変更できます。



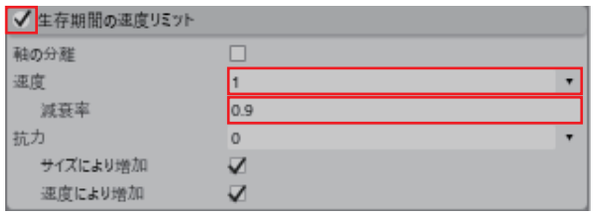
4.3.5 生存期間の色モジュールの設定

生存期間の色を変更します。右下図になるように変更しましょう。



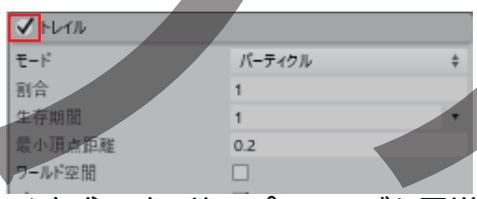
4.3.6 生存期間の速度リミット

パーティクルの移動速度の減衰率を変更します。速度１以上のとき１割のスピードまで落とします。これによりパーティクル発生時は速く動きますが、徐々に速度が遅くなります。



4.3.7 トレイルモジュールを有効にする

トレイルモジュールを有効にすることによって尾を引くようなエフェクトを作成できます。レンダラーでトレイルマテリアルを設定したのはこのためです。

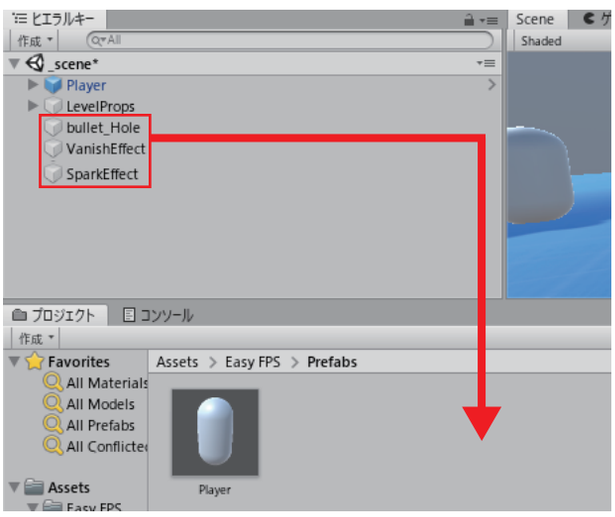


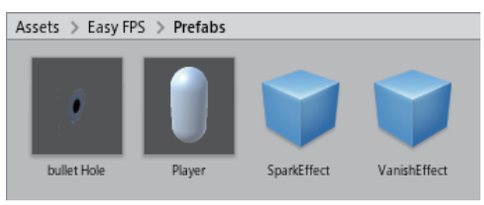
これで火花のエフェクトは完成です。他のプレファブと同様に DestroyAfterTime をアタッチしましょう。

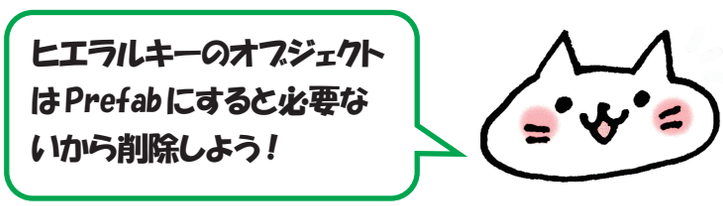
4.3.8 エフェクトを Prefab にする

オブジェクトを Prefab にする方法は簡単です。Prefab にしたいオブジェクトをプロジェクトウィンドウにドラッグ&ドロップで移動するだけです。

VanishEffect と bullet_Hole を [Assets] > [Easy FPS] > [Prefab] ドラッグ&ドロップで Prefab にしましょう。







5 弾オブジェクトの作成

弾オブジェクトは実際に弾丸となるオブジェクトは発射せずにRaycast（レイキャスト）という機能を利用します。

5.0 レイキャスト

レイキャストとはある地点から透明な線を特定の方向に引いて、そのRay上のどこかでコライダがあるか検知をするものです。例えば、ゲームであれば銃から弾を発射して敵に攻撃する場合などに用います。

Unityでは特にオブジェクトにタッチなどを行なった場合の当たり判定の実装として使われています。レイキャスターはキャンバス上にある全てのグラフィックを監視し、透明な線がどのオブジェクトにヒットしたのかを決定します。また、判定を行いたくない、つまりレイキャストをブロックするオブジェクトのタイプやマスクを設定することもできます。

5.0.0 Raycast の書き方

要素が多いですが、特に重要なのは開始地点と向きです。ここが間違っていると当たり判定は意図したものとまったく異なるものになってしまいます。

構文
```
Physics.Raycast(Vector3 origin, Vector3 direction,RaycastHit hitInfo,
                                                  float distance, int layerMask);
```

引数 1:rayの開始地点　2:rayの向き　3:当たったオブジェクトの情報を格納　3:rayの発射距離　4:レイヤマスクの設定

5.1 Raycast を使ってオブジェクトの情報を取得してみる

実際にRaycastを使ってRayを発射することにより着弾地点のオブジェクトの情報を取得してみましょう。

5.1.0 オブジェクトとスクリプトの作成、アタッチ

空のオブジェクト「Bullet」を作成しましょう。

スクリプト「BulletScript」を作成し、「Bullet」にアタッチしましょう。

5.1.1 BulletScript の編集

BulletScriptを以下のように変更しましょう。

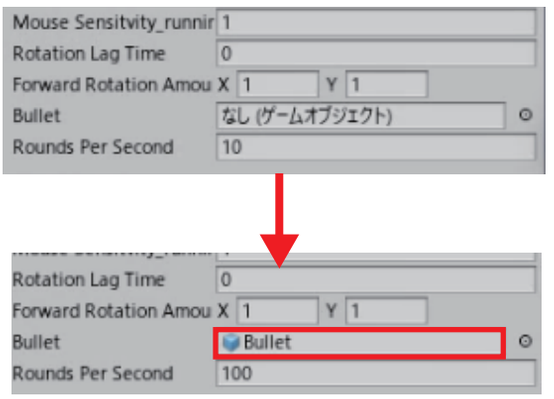
BulletScript.cs

```
private void Update () {

        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(transform.position, transform.forward,out hit, 100)){
              Debug.Log(hit.collider.gameObject.name);
              Destroy(gameObject);
        }

}
```

5.1.2 Prefab の作成

Bullet オブジェクトを Prefab にし、「NewGun_auto」、「NewGun_semi」のインスペクターの Bullet プロパティに Bullet オブジェクトの Prefab を設定しましょう。

他の Prefab と同様にヒエラルキーの Bullet は削除しましょう。



Check! できているかチェックをしよう！

- ☐ コンソールに Ray が当たったオブジェクト名が表示された

早く終わったら **チャレンジ問題**

今回は制作者に特別な許可を頂いて Assets Store にある「Easy FPS」を教材として使用していますが、冒頭で触れた通りでこのような素材は様々な規約が設けられており、使用する際は規約に違反していないかを十分に注意する必要があります。

IT 技術者はプログラムだけではなく、幅広い知識が必要です。そこで、今回は IT 企業に勤めたい社会人が受ける国家資格である「基本情報技術者」の試験にて IT 法務に関する実際に出題された過去の問題をピックアップしてみました。
次のページに解説を載せています。

問題 1（令和元年秋期に出題）

シュリンクラップ契約において、ソフトウェアの使用許諾が成立するのはどの時点か

ア：購入したソフトウェアの代金を支払った時点
イ：ソフトウェアの入った DVD-ROM を受け取った時点
ウ：ソフトウェアの入った DVD-ROM の包装を解いた時点
エ：ソフトウェアを PC にインストールした時点

問題 2（平成 21 年秋期に出題）

プログラム中のアイディアやアルゴリズムは保護しないが，プログラムのコード化された表現を保護する法律はどれか。

ア：意匠法　イ：商標法　ウ：著作権法　エ：特許法

問題３（平成 29 年春期に出題）

著作権法によるソフトウェアの保護範囲に関する記述のうち，適切なものはどれか。

ア：アプリケーションプログラムは著作権法によって保護されるが，OS などの基本プログラムは権利の対価がハードウェアの料金に含まれるので，保護されない。

イ：アルゴリズムやプログラム言語は，著作権法によって保護される。

ウ：アルゴリズムを記述した文書は著作権法で保護されるが，そのアルゴリズムを用いて作成されたプログラムは保護されない。

エ：ソースプログラムとオブジェクトプログラムの両方とも著作権法によって保護される。

問題１　正解　ウ

シュリンクラップ契約とは、ソフトウェアの購入者がパッケージを開封することで使用許諾契約に同意したとみなす契約方式のことです。パッケージの表面に使用許諾契約が印刷され、透明フィルムで包装された市販のパッケージソフトなどに適用されるケースがあります。
シュリンクラップ(Shrink-wrap)とは、製品の包装のことです。

したがって使用許諾契約が成立するのは「包装を解いた時点」となります。

問題2　正解　ウ

意匠法：意匠法は、製品の価値を高める形状やデザインを保護する法律です。
商標法：商標法は、商品の名称やロゴマークなどを保護する法律です。
著作権法：正しい。著作権法第十条の3にアルゴリズム (アイディア)、プログラム言語は、保護の対象である旨の記述があります。
特許法：特許法は、自然の法則や仕組みを利用した価値ある発明を保護する法律です。

問題3　正解　エ

ア：アプリケーションプログラムは著作権法によって保護されるが，OS などの基本プログラムは権利の対価がハードウェアの料金に含まれるので，保護されない。

OS も著作権法で保護されます。

イ：アルゴリズムやプログラム言語は，著作権法によって保護される。

著作権法に「著作物に対するこの法律による保護は、その著作物を作成するために用いるプログラム言語、規約及び解法に及ばない。(第 10 条 3 項)」と規定されているため、アルゴリズムやプログラム言語は保護対象外です。

ウ：アルゴリズムを記述した文書は著作権法で保護されるが，そのアルゴリズムを用いて作成されたプログラムは保護されない。

アルゴリズムは保護対象外ですが、それを記述した文書およびプログラムは著作権法で保護されます。

エ：ソースプログラムとオブジェクトプログラムの両方とも著作権法によって保護される。

正しい。プログラムは著作権法における保護対象です。

0 目標

・Prefabを複製する
・タグを使ったオブジェクトの判別ができる
・オリジナル要素を組み込むことができる

1 エフェクトの生成

Raycastを使ってオブジェクトの情報を取得することができました。これを利用してオブジェクトを判別し、エフェクトを生成します。

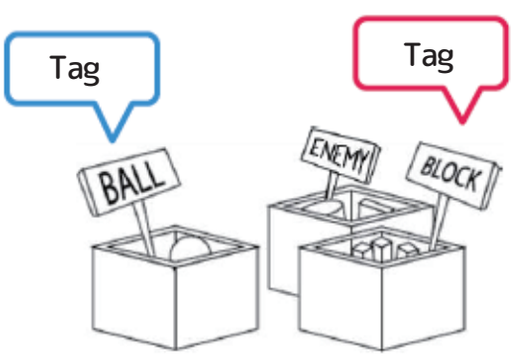
1.0 Tag(タグ) の作成と取得

オブジェクトの判別はTagを利用して行います。取得したオブジェクトのTagによって生成するエフェクトを変更します。

1.0.0 Tagとは

タグとはゲームゲームオブジェクトを区別するのに使用します。デフォルトでは[Untagged]（日本語でタグ付けされていないという意味）となっています。

たとえば、シューティングゲームの場合、自機が発射した弾は「Player」というタグを、敵が発射した弾には「Enemy」というタグを設定し、自機に当たった弾のタグが「Enemy」ならダメージを受けるという処理が分かりやすく簡単にできます。

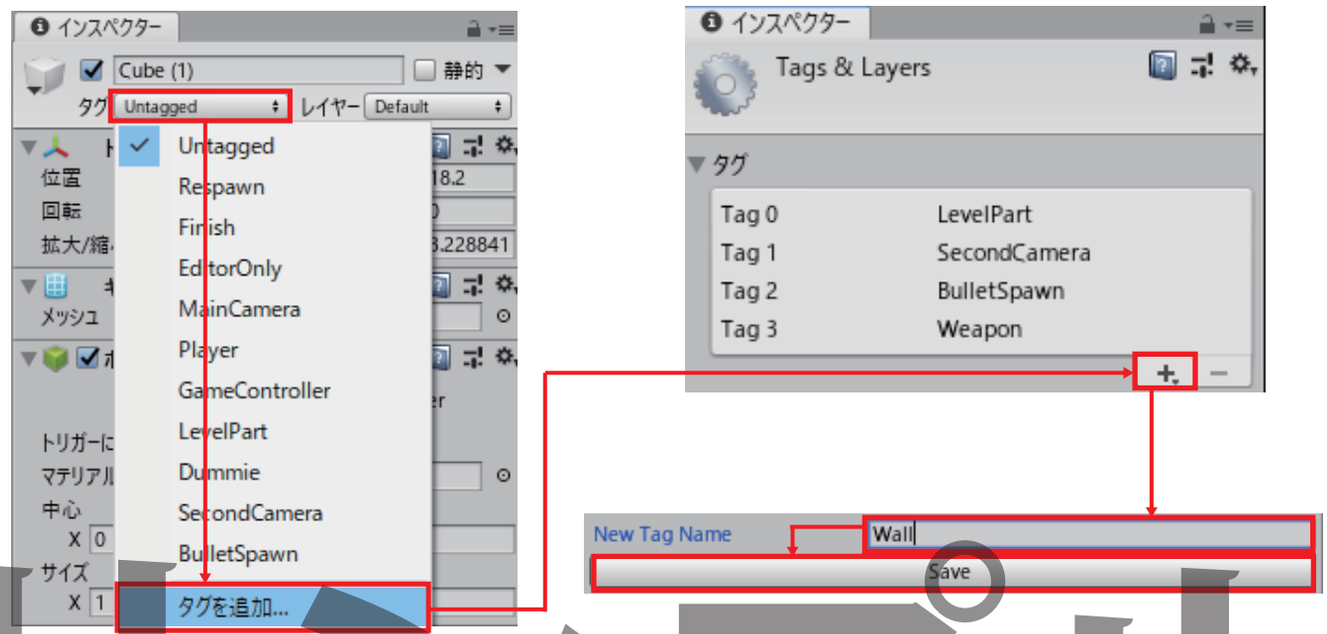


壁とターゲットのオブジェクトに別々のタグをつけて、壁のタグのときはbulletHoleとSparkEffect、ターゲットの場合はVanishEffectを生成するんだね！

1.0.1 タグの作成

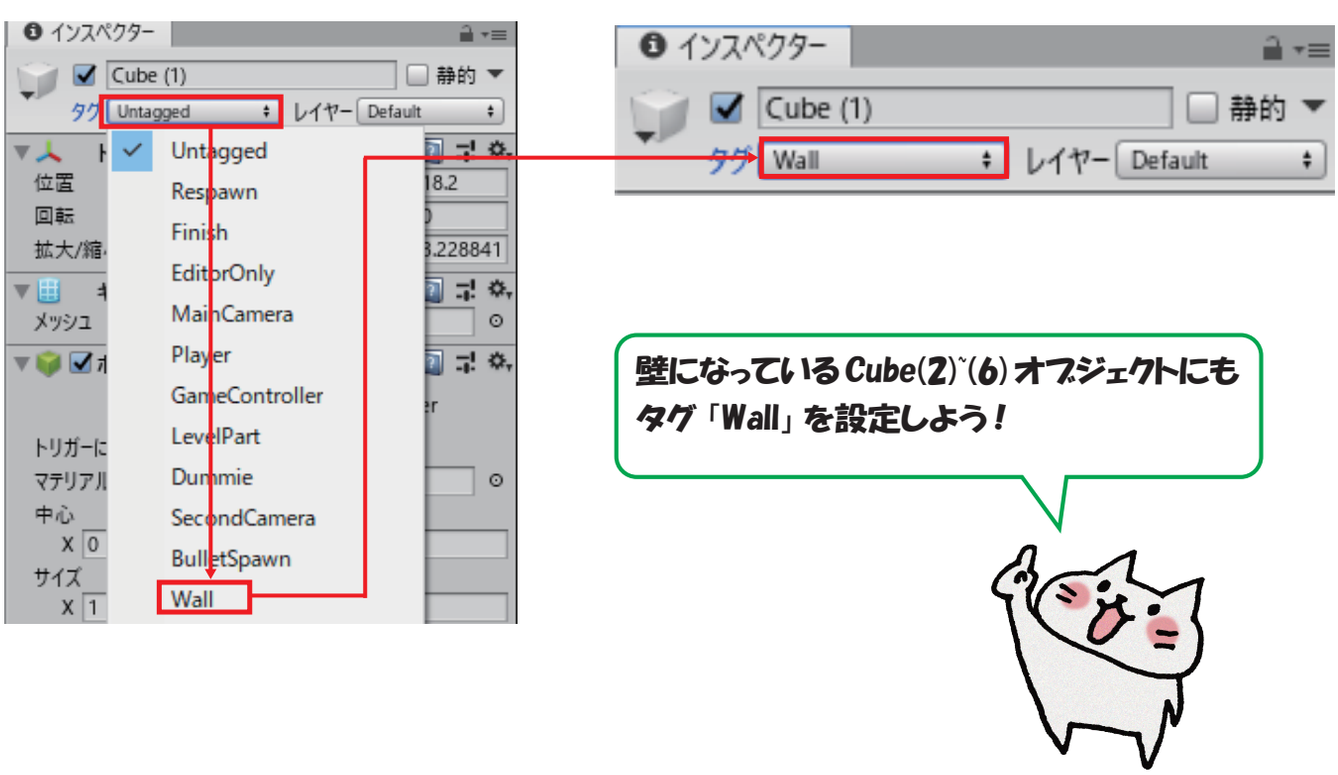
壁となるオブジェクトにタグをつけてみます。タグの名前は「Wall」とします。

タグを作成するにはインスペクターから [タグ]→[タグを追加] から作成できます。以下の手順で作成しましょう。



1.0.2 タグの適用

これでタグが作成されました。インスペクターのタグをクリックすると先ほど作成した [Wall] というタグが表示されているはずです。Cube(1) オブジェクトのタグを [Wall] に変更しましょう。



やってみよう！

タグ「Target」を作成し、的となる Dummie（1）～(3) オブジェクトに設定しましょう。

1.0.3 タグの取得

設定したタグをスクリプトで取得します。BulletScript を以下のように変更しましょう。

BulletScrpt.cs

```
private void Update () {

        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(transform.position, transform.forward,out hit, 100)){
              Debug.Log(hit.collider.gameObject.tag);
              Destroy(gameObject);
        }
}
```

Check! できているかチェックをしよう！

- ☐ Cube(1) ～ (6) に着弾するとコンソールに「Wall」と出力される
- ☐ Dummie(1) ～ (3) に着弾するとコンソールに「Target」と出力される

1.1 着弾点の取得とエフェクトの生成

タグを取得できました。これでオブジェクトが判別できるようになりましたね。次に着弾点を取得し、着弾点にエフェクトを生成します。

1.1.0 着弾点の取得

着弾点を取得してみます。以下のように変更しましょう。

BulletScrpt.cs

```
private void Update () {

        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(transform.position, transform.forward,out hit, 100)){
              Debug.Log(hit.point);
              Destroy(gameObject);
        }
}
```

Check! できているかチェックをしよう！

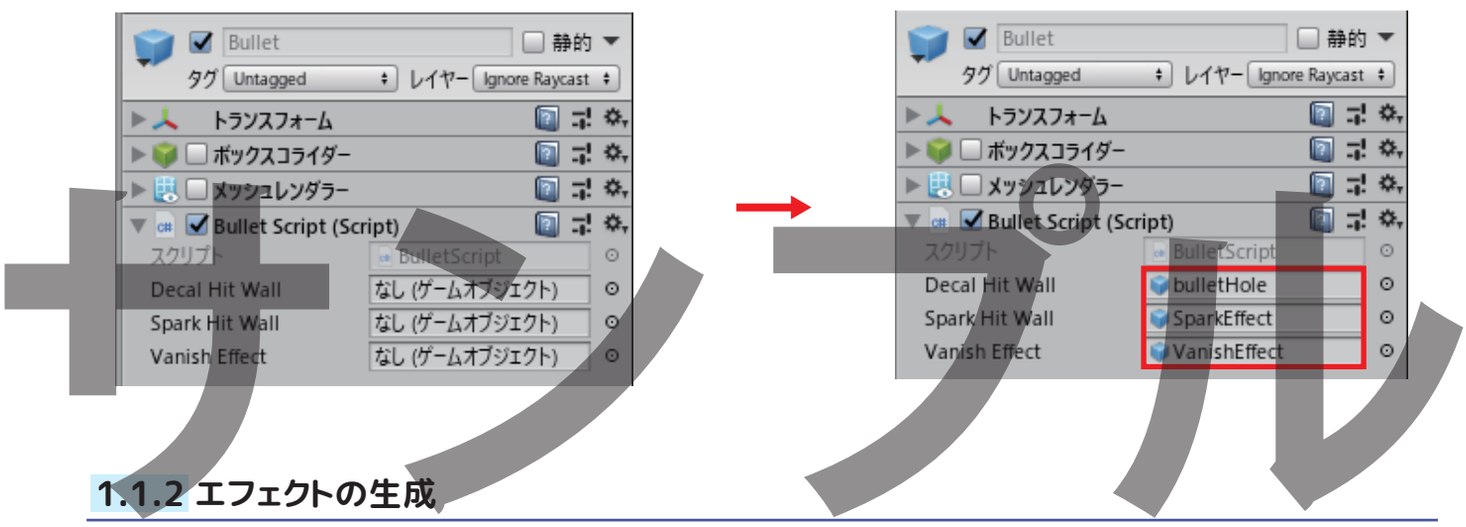
- ☐ コンソールに座標が出力された

1.1.1 プレファブの関連付け

プレファブをスクリプトから操作できるようにします。以下のように変更し、インスペクターでエフェクトを関連付けましょう。

BulletScrpt.cs

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

public class BulletScript : MonoBehaviour {
    public GameObject DecalHitWall;
    public GameObject SparkHitWall;
    public GameObject VanishEffect;
    void Start() {
```



1.1.2 エフェクトの生成

着弾点を取得してみます。以下のように変更し、インスペクターでエフェクトを関連付けましょう。

BulletScrpt.cs

```
void Update () {

        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(transform.position,transform.forward, out hit, 100)){
            if (hit.collider.gameObject.tag == "Wall") {
                //穴のエフェクトの生成
                Instantiate(DecalHitWall, hit.point + hit.normal * 0.1f,
                                        Quaternion.LookRotation(hit.normal));
                //火花のエフェクトの生成
                Instantiate(SparkHitWall, hit.point + hit.normal*0.1f,
                                        Quaternion.LookRotation(hit.normal));

            }
            Destroy(gameObject);
        }
}
```

衝突したオブジェクトのタグがWallの場合、穴があくエフェクトと火花のエフェクトを生成しています。生成するにあたって「Instantiate」というプログラムを使用しています。

オブジェクトを生成する

構文 Instantiate（出現させたい物, 生成位置, 回転角）	複製させたい物を、指定の位置、指定の回転角で複製させる
用例 `Instantiate(sparkHitWall, hit.point + hit.normal * 0.1f, Quaternion.LookRotation(hit.normal));`	

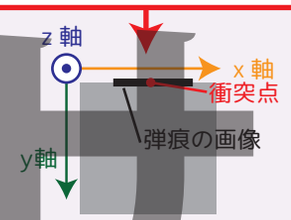
よく見ると生成位置が着弾点 + hit.normal * 0.1f となっていますね。これは生成したオブジェクトが壁に埋もれて見えなくならないようにするためです。壁から0.1だけ浮かして表示しています。

hit.normalとは衝突時の法線ベクトルを取得しています。簡単に言うと衝突したオブジェクトの角度です。壁の角度に合わせて生成位置をずらします。

「着弾点+hit.normal*0.01f」の命令を入れる理由について考える

考え方のステップ①

衝突点と弾痕が同じ座標



壁と弾痕が重なり弾痕が見えない可能性がある

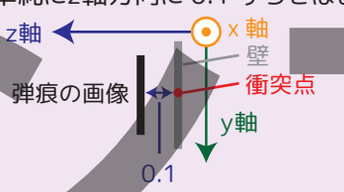
そこで…

壁から0.1だけ弾痕の座標をずらす

考え方のステップ②

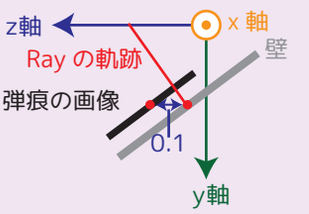
正面から弾が当たった場合

単純にz軸方向に 0.1 ずらせばよいが…



壁が斜めになっていた場合

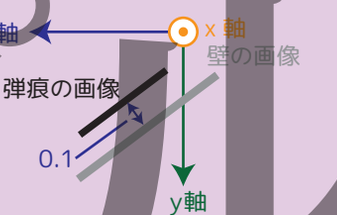
単純にz軸方向に 0.1 ずらすと…



中心がずれてしまう

考え方のステップ③

壁が斜めでも角度に合わせて弾痕が置かれる調整が必要



壁に対して
x,y,z の全ての成分を調整して
垂直に 0.1 ずらす命令が

hit.normal*0.1f

オブジェクトを特定の方向に向ける

構文 Quaternion.LookRotation (vector3);	オブジェクトを特定の方向に向けます。常に実行することによって常に特定のターゲットに視線を向けることもできます。
用例 `Quaternion.LookRotation(hit.normal));`	

やってみよう！

タグ「Target」に衝突した際に VanishEffect を生成し、Bullet だけでなく、衝突したオブジェクトも削除しましょう。

衝突したオブジェクトは削除するには
「 Destroy(hit.transform.gameObject); 」と記述するよ！

Check! できているかチェックをしよう！

- ☐ Cube(1) ～ (6) に着弾すると穴と火花のエフェクトが生成された
- ☐ Dummie(1) ～ (3) に着弾すると消滅エフェクトが生成され、オブジェクトが消えた

サンプル

2 的当てゲームにする

オブジェクトを的当て形式のゲームにしていきます。そのために終了のお知らせ、ポイント表示ができるようにします。

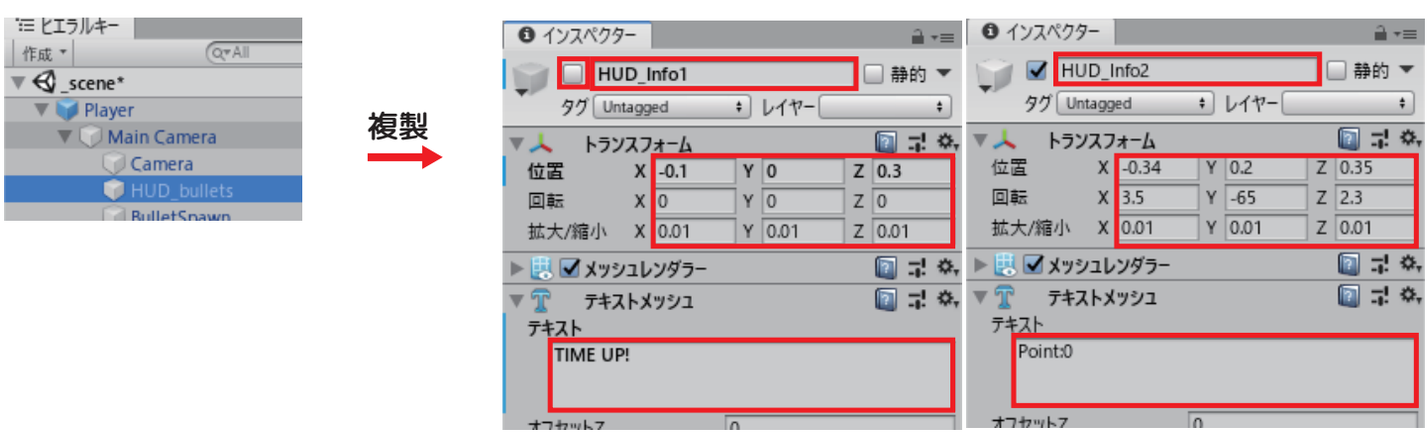
2.0 UI の作成

ゲーム終了をお知らせするオブジェクトとポイントを表示するオブジェクトを作成します。

2.0.0 オブジェクトの作成

UI は弾数を表示するオブジェクト「HUD_bulles」を複製して作成します。ゲーム終了をお知らせするオブジェクトを「HUD_Info1」、ポイントを表示するオブジェクトを「HUD_Info2」として作成します。

それぞれ以下のように変更しましょう。



2.1 ゲームを停止するプログラムの作成

ゲーム終了をお知らせするオブジェクトとポイントを表示するオブジェクトを作成します。

2.1.0 スクリプトの作成とアタッチ

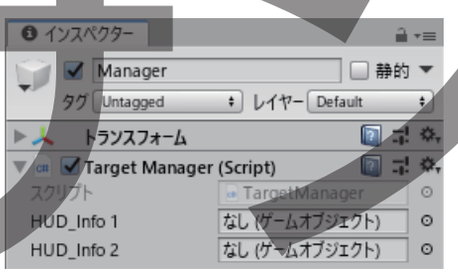
的当てゲームの流れを管理するスクリプト「TargetManager」スクリプトを作成しましょう。また、空のオブジェクト「Manager」を作成し、アタッチしましょう。

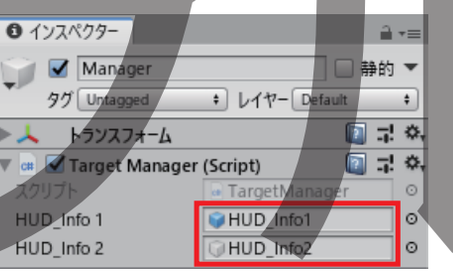
2.1.1 UI の関連付け

UI オブジェクトを扱えるように変数を宣言し、関連付けましょう。

TargetManager.cs

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

public class TargetManager : MonoBehaviour
{
    public GameObject HUD_Info1;
    public GameObject HUD_Info2;
```



2.1.2 ゲーム終了処理

一定時間を経過するとゲームが終了するようにします。今回は 20 秒にしますが、動作確認後、好きな値に設定しても構いません。

TargetManager.cs

```
void Update(){
    //20 秒経過したなら
    if(20.0f - Time.time < 0) {
        // オブジェクトを有効にする
        HUD_Info1.gameObject.SetActive(true);
        // ゲームを停止する
        UnityEditor.EditorApplication.isPaused = true;
    }
}
```

Check! できているかチェックをしよう！

- ☑ 20 秒経過するとゲームが停止した
- ☑ 20 秒経過すると「TIME UP!」と表示された

2.2 ポイントを加算するプログラムの作成

ポイントを加算する仕組みははTargetManagerにポイントを加算するメソッドを定義し、GunScrptから呼び出します。

2.2.0 TargetManagerの編集

TargetManagerにメソッドを定義します。以下のように変更しましょう。

TargetManager.cs

```
public class TargetManager : MonoBehaviour
{
    public GameObject HUD_Info1;
    public GameObject HUD_Info2;
    int Point = 0;
    void Start()
    {

    }

    public void AddPoint() {
        Point += 10;
        HUD_Info2.GetComponent<TextMesh>().text = "Point:" + Point;
    }
```


2.2.1 BulletScriptの編集

BulletScriptからTargetManagerのAddPointを呼び出します。以下のように変更しましょう。

BulletScrpt.cs

```
if (hit.collider.gameObject.tag == "Target") {
    Instantiate(VanishEffect, hit.point, Quaternion.LookRotation(hit.normal));
    Destroy(hit.transform.gameObject);
    TargetManager tms = GameObject.Find("Manager").GetComponent<TargetManager>();
    tms.AddPoint();
    Destroy(gameObject);
}
```

Check! できているかチェックをしよう！

- ☐ ターゲットにヒットするとポイントが加算された

2.3 動く的の作成

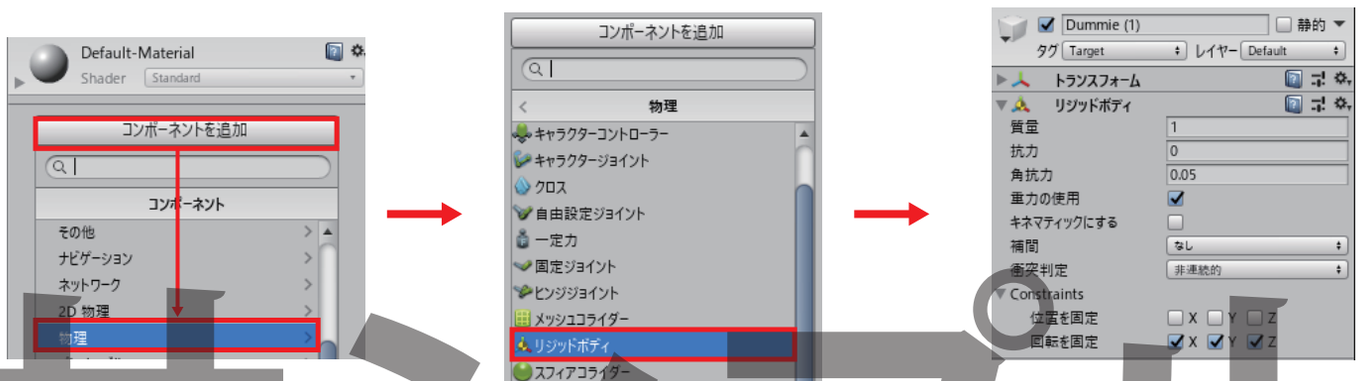
的となるオブジェクトを動くようにします。5秒経過ごとにランダムな方向に向き、壁に当たると反対を向くようにします。

Dummie(1) オブジェクトを対象にして作成していきます。

2.3.0 Rigidbody コンポーネントの追加と設定

オブジェクトの衝突を検知するには Rigidbody コンポーネントが必要です。的となるオブジェクトにインスペクターから [物理] > [リジッドボディ] を選択してコンポーネントを追加しましょう。

追加後、スクリプト以外の要素（オブジェクトに衝突など）で回転しないように回転を固定します。



2.3.1 スクリプトの作成とアタッチ

オブジェクトを動かすスクリプト「TargetController」を作成し、オブジェクトにアタッチしましょう。作成後、以下のように変更しましょう。

TargetController.cs

```
    void Update() {
        timeCount += Time.deltaTime;
        // 自動で前進する。
        transform.position += transform.forward * Time.deltaTime;
        // 5秒経過したら
        if (timeCount > 5.0f) {
            // 進路をランダムに変更する。
            Vector3 course = new Vector3(0, Random.Range(0, 180), 0);
            transform.localRotation = Quaternion.Euler(course);
            // タイムカウントを0に戻す。
            timeCount = 0;
        }
    }
    void OnCollisionEnter(Collision collision) {
        if(collision.gameObject.tag == "Wall") {
            // 壁に触れたら反対に向く
            Vector3 course = new Vector3(0, 180, 0);
            transform.localRotation = Quaternion.Euler(course);
        }
    }
}
```

Check! できているかチェックをしよう！

- ☐ 5秒経過ごとに進路が変わった
- ☐ 壁に衝突すると反対を向いた

早く終わったら チャレンジ問題

Challenge1

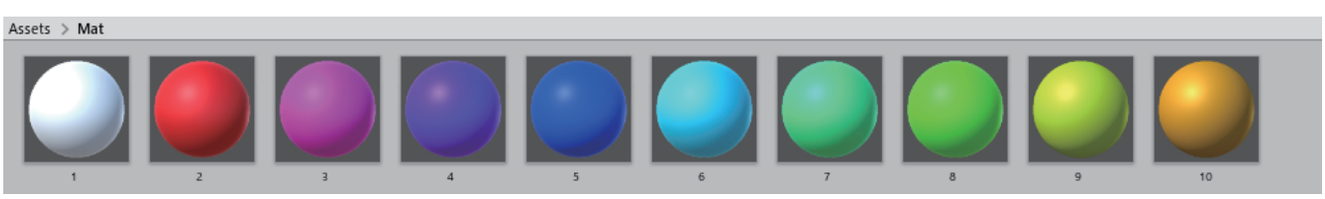
ステージや的を自由に作成し、オリジナルゲームを作成する

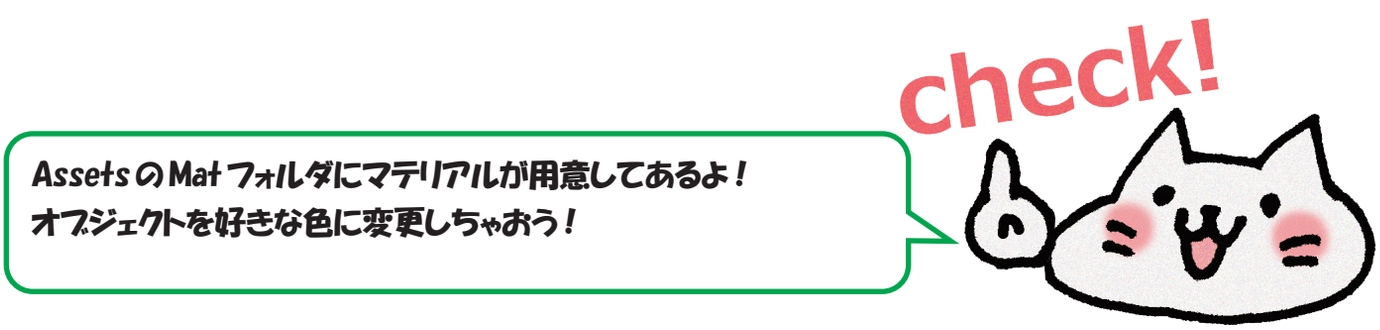
Challenge2

タグを追加してオブジェクトによって獲得できるポイントを変更する

Challenge3

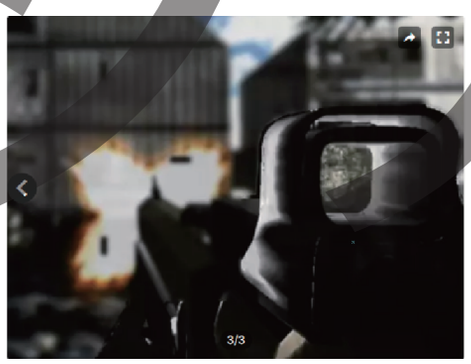
的となるオブジェクトに衝突した際にポイントが減るようにオブジェクトとスクリプトを作成する





アドバンスコース

サンプル



◆文科省　ICT活用教育アドバイザー事務局掲載
学校ICT化サポート事業者
◆長期コースによる、プログラミングの普及