Unity講座
～FPS的な～

3年　　海苔　威

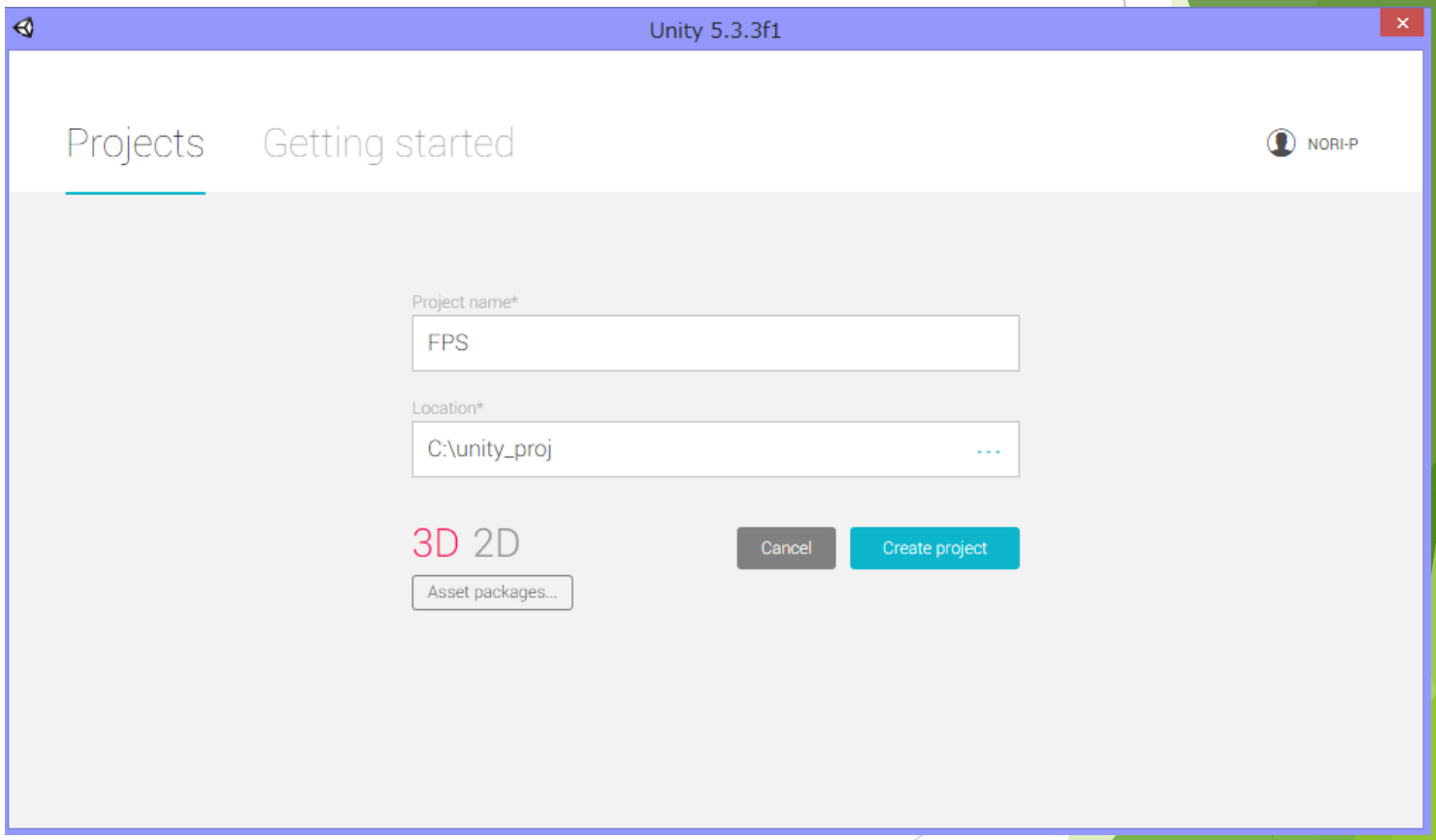
目次

1. プロジェクトの準備
2. プレイヤーを動かす
3. 射撃
4. 射撃にエフェクトをつける
5. 武器の追加

1. プロジェクトの準備

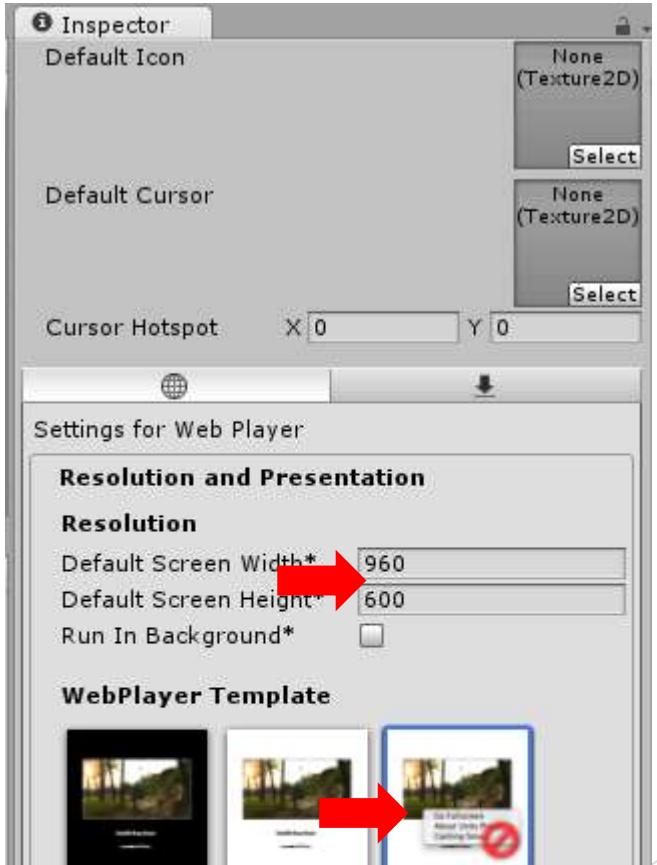
- まずは準備
- プロジェクト名を決めて

  3Dを選択し作成



- 「File」→「Build Settings」
- 「Web Player」を選択
- 左下の「Player Settings」

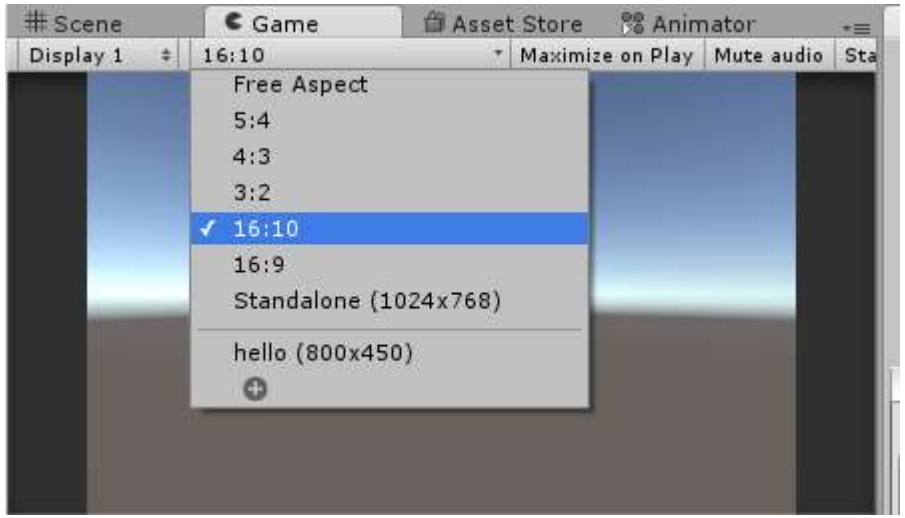
- するとUnityの画面、Inspectorに設定画面が出る



- 「Resolution and Presentation」のサイズを960、600にする
- 「WebPlayer Template」を一番右のやつにする

- Build Settingsは閉じていい

- Gameビューのアスペクト比を16:10にする



- 「File」→「Save Scene」を選択
  名前を「Scene_Stage」として保存する

- ファイルが増えると厄介なのでフォルダを作りましょう
- 「Assets」→「Create」→「Folder」
- 名前は「F00_Stage」
- 同様にあと6つフォルダを作る（名前は分かりやすいもので自由に）

  「F01_Script」

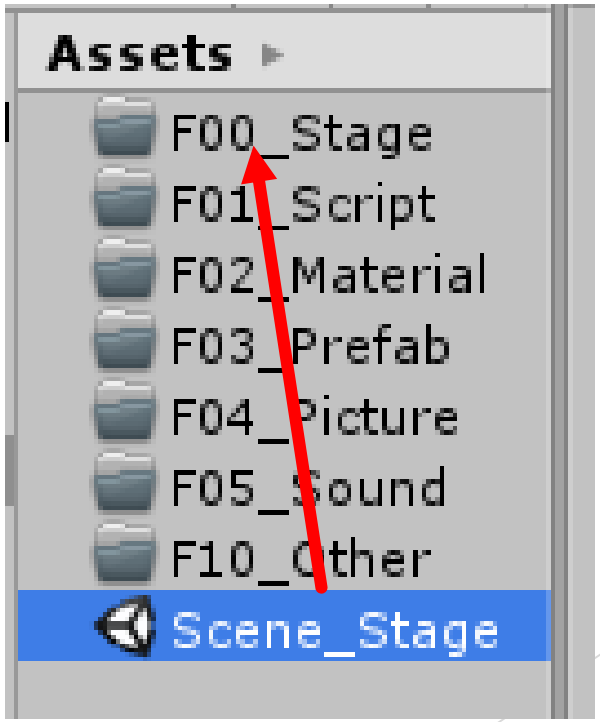
  「F02_Material」

  「F03_Prefab」

  「F04_Picture」

  「F05_Sound」

  「F10_Other」



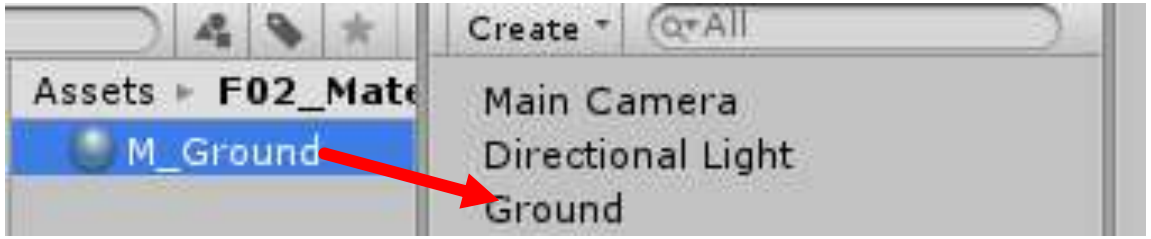
- 「Scene_Stage」を「F00_Stage」に入れる

- これで準備完了

2. プレイヤーを動かす

- まずは地面を用意
- 「GameObject」→「3D Object」→「Cube」
- 名前は「Ground」、Positionは（0, -0.5, 0）、Scaleは（50, 1, 50）

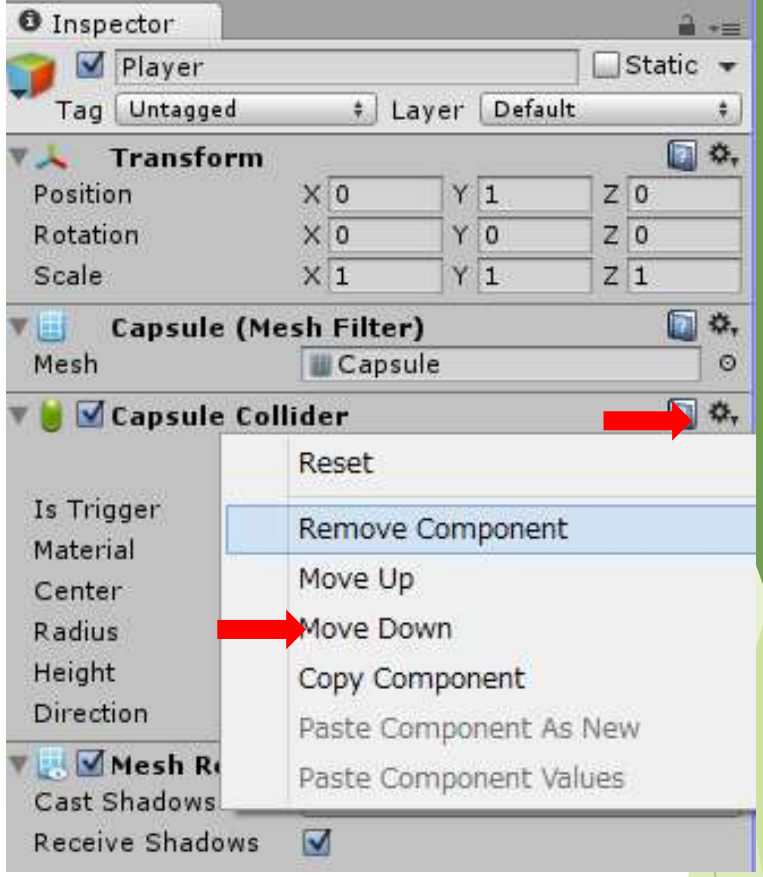
- フォルダF02_Materialで右クリック
  「Create」→「Material」、名前は「M_Ground」
  色を適当に決めてヒエラルキの「Ground」にドラッグ&ドラッグ



- 「GameObject」→「3D Object」→「Capsule」
  名前は「Player」、Positionは（0, 1.0, 0）

- 「Player」のCapsule Colliderから
  「歯車マーク」→「Remove Component」で削除

- 代わりにAdd Componentから
  「Physics」→「Character Controller」を追加



- 「Character Controller」を使ってプレイヤーをいろいろ動かします

- フォルダ「F01_Script」で右クリック「Create」→「C# Script」
- 名前は「C01_Player」

```
using UnityEngine;
using System.Collections;

public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;
    private Vector3 move = Vector3.zero;
    private float speed = 5.0f;

    void Start()
    {
        charaCon = GetComponent<CharacterController>();
    }

    void Update()
    {
        move = new Vector3(Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));
        charaCon.Move(move * speed * Time.deltaTime);
    }
}
```

- スクリプトが書けたらヒエラルキの「Player」にドラッグ&ドロップ
- ゲームを実行
- 上下左右のカーソルキーでプレイヤーが動くはず！

- 「Character Controller」の情報を扱う変数として「charaCon」を用意
- Update関数で受けた入力に対して動く方向が決まる
- 入力はジョイスティック入力

→スティックの傾きで-1.0～1.0の範囲で変化する。滑らかな値変化。

「Horizontal」・・・左右キー（A, Dでも反応）

「Vertical」・・・上下キー（W, Sでも反応）

をそれぞれ表す

- 次はプレイヤーをジャンプさせよう！

- 「C01_Player」に
  ・GRAVITYという重力を扱う変数
  ・Update内に4行
  を追加

- ほんとに重力が働いているか気になる人はPlayerのy座標を5とかにすると分かる

```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;
    private Vector3 move = Vector3.zero;
    private float speed = 5.0f;
    private const float GRAVITY = 9.8f;

    void Start()
    {
        charaCon = GetComponent<CharacterController
    }

    void Update()
    {
        float y = move.y;
        move = new Vector3(Input.GetAxis("Horizonta
        move *= speed;
        move.y += y;
        move.y -= GRAVITY * Time.deltaTime;
        charaCon.Move(move * Time.deltaTime);
    }
}
```

- ジャンプ力を扱う変数とUpdate内を少しいじる

- ここまでやってゲーム実行
- さっきのプレイヤー移動に加えて
  「スペースキー」を押すとジャンプできる！

- GRAVITYやjumpPowerの値を変えるとジャンプの具合も変わるので自由に
  （GRAVITYの値は倍でいいかも）

```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;
    private Vector3 move = Vector3.zero;
    private float speed = 5.0f;
    private const float GRAVITY = 9.8f;
    private float jumpPower = 10.0f;

void Update()
{
    float y = move.y;
    move = new Vector3(Input.GetAxis("Horizonta
    move *= speed;
    move.y += y;
    if (Input.GetKeyDown(KeyCode.Space))
    {
        move.y = jumpPower;
    }
    move.y -= GRAVITY * Time.deltaTime;
    charaCon.Move(move * Time.deltaTime);
}
```

- このままだと空中ジャンプが無限に可能…
- 地面にいる時のみジャンプ可能にしたい

- 「C01_Player」のUpdate内
  ジャンプの条件を書いた部分を→の
  if文で囲う

```
if (charaCon.isGrounded)
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        move.y = jumpPower;
    }
}
```

- 空中ジャンプは回避！

- isGroundedはオブジェクトが接地しているかをtrue/falseで返してくれる

- ここまでをいったんまとめます
- →の感じで部分ごとに
  コードをまとめると見やすい

```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;   // キャラクターコンポーネント用の変数
    private Vector3 move = Vector3.zero;    // キャラ移動量.
    private float speed = 5.0f;             // 移動速度
    private float jumpPower = 10.0f;        // 跳躍力.
    private const float GRAVITY = 9.8f * 2; // 重力

    void Start()
    {
        charaCon = GetComponent<CharacterController>();
    }

    void Update()
    {
        //▼▼▼移動量取得▼▼▼
        float y = move.y;
        move = new Vector3(Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));
        move *= speed;

        //▼▼▼ジャンプ処理▼▼▼
        move.y += y;
        if (charaCon.isGrounded)
        {
            if (Input.GetKeyDown(KeyCode.Space))
            {
                move.y = jumpPower;
            }
        }
        move.y -= GRAVITY * Time.deltaTime;

        //▼▼▼移動処理▼▼▼
        charaCon.Move(move * Time.deltaTime);
    }
```

- さっきのを一部かえて
  よりFPSっぽい動きにします
- ジャンプの部分は変化なし
- ジャンプと移動処理の間に
  ↓の向き変更部分を追加

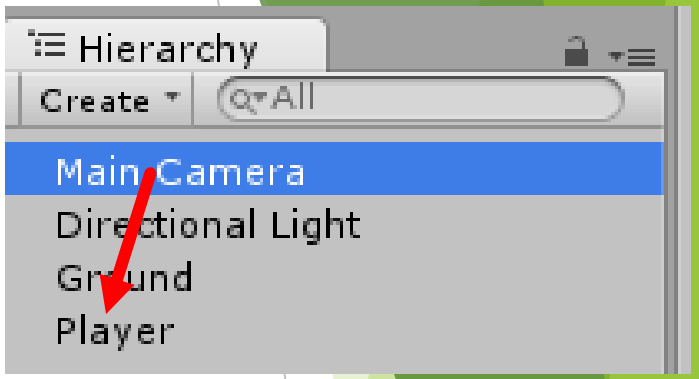
```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;    // キャラクターコンポーネント用の変数
    private Vector3 move = Vector3.zero;     // キャラ移動量.
    private float speed = 5.0f;              // 移動速度
    private float jumpPower = 10.0f;         // 跳躍力.
    private const float GRAVITY = 9.8f * 2; // 重力
    private float rotationSpeed = 180.0f;    // プレイヤーの回転速度

    void Start()
    {
        charaCon = GetComponent<CharacterController>();
    }

    void Update()
    {
        //▼▼▼移動量取得▼▼▼
        float y = move.y;
        move = new Vector3(0.0f, 0.0f, Input.GetAxis("Vertical"));
        move = transform.TransformDirection(move);
        move *= speed;
```

```
// ▼▼▼プレイヤーの向き変更▼▼▼
Vector3 playerDir = new Vector3(Input.GetAxis("Horizontal"), 0.0f, 0.0f);
playerDir = transform.TransformDirection(playerDir);
if (playerDir.magnitude > 0.1f)
{
    Quaternion q = Quaternion.LookRotation(playerDir);
    transform.rotation = Quaternion.RotateTowards(transform.rotation, q, rotationSpeed * Time.deltaTime);
}
```

```
//▼▼▼移動処理▼▼▼
charaCon.Move(move * Time.deltaTime);
```

- ゲーム実行
- 動き方が変わった！（わかりづらい）

- のでカメラをプレイヤーと一緒に動くようにする
- ヒエラルキの「Main Camera」を「Player」にドロップ&ドロップ



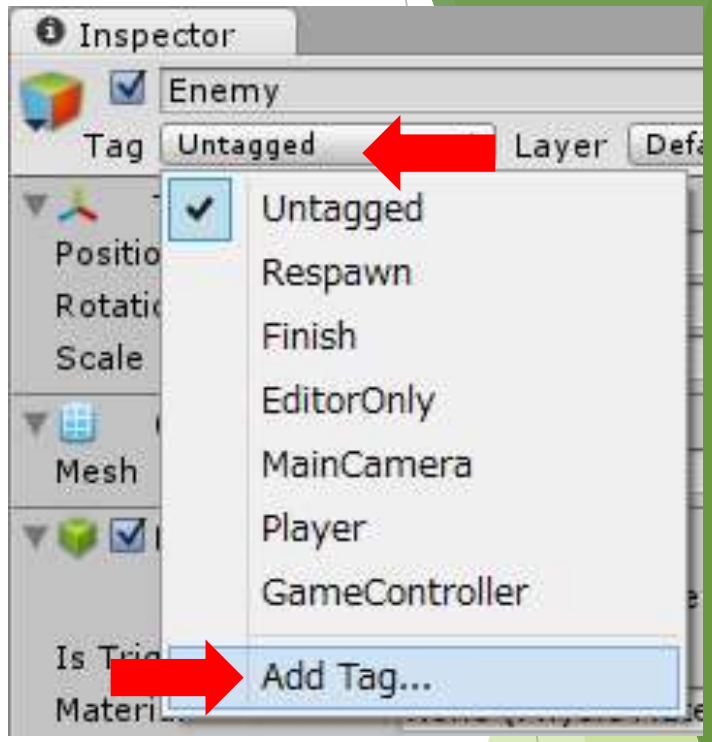
- 子オブジェクトになったMain CameraのPositionを（0, 0.8, 0）にする
- でゲーム実行

- 一人称視点で動ける

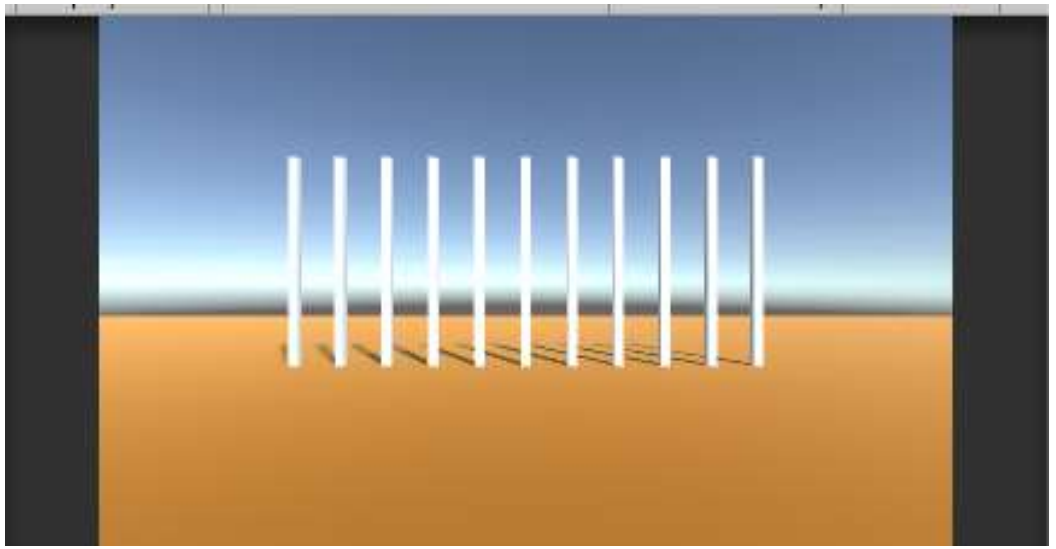
- とりあえずプレイヤーの移動は終了

3. 射撃

- まずは的を作る
- 「GameObject」→「3D Object」→「Cube」
- 名前は「Enemy」
- Positionは（0, 2 ,10）、Scaleは（0.2, 5, 0.2）
- 「Tag」→「Add Tag」をクリック
- Tagsの+マークをクリックし「Enemy」と入力
- ヒエラルキのEnemyのタグを追加した「Enemy」に変更



- ヒエラルキの「Enemy」を選択した状態で「Ctrl + D」を連打
- 的を10個くらい作る
- それぞれPositionのxの値を-5~5くらいで1ずつずらしていって並べる



- ↑のかんじで。本数とか場所は適当でいい

- 次は銃弾

- 「GameObject」→「3D Object」→「Sphere」
- 名前は「P_Bullet」
- Scaleは（0.2, 0.2, 0.2）
- 「Add Component」→「Physics」からRigidbodyを追加
- Rigidbodyの「Use Gravity」のチェックは外す

- フォルダ「F01_Script」で右クリック、「Create」→「C# Script」
- 名前は「C02_Bullet」
- 「C02_Bullet」をヒエラルキのP_Bulletに追加

- ヒエラルキのP_Bulletをフォルダ「F03_Prefab」に移動
- ヒエラルキのP_Bulletは削除

- 「C02_Bullet」の中身を書く

- 弾が生成されると同時に読み込まれる

- 消滅までの時間が設定されていて
  - ・時間がくる
  - ・時間がくる前に、オブジェクトに衝突で消滅

- 衝突したオブジェクトのタグが「Enemy」であるとき、相手オブジェクトを破壊

```
using UnityEngine;
using System.Collections;

public class C02_Bullet : MonoBehaviour
{
    private float bulletSpeed = 7.0f; //弾の速度
    private const float LIFETIME = 3.0f; //弾の消滅時間
    private float time = 0.0f; //生成からの時間

    // Update is called once per frame
    void Update()
    {
        time += Time.deltaTime;
        if (time > LIFETIME)
        {
            Destroy(gameObject);
        }
        transform.Translate(0.0f, 0.0f, bulletSpeed * Time.deltaTime);
    }

    //■■オブジェクトに衝突したときに呼び出される■■
    void OnCollisionEnter(Collision other)
    {
        Destroy(gameObject); //弾オブジェクトを破壊
        if (other.gameObject.tag == "Enemy") //衝突がEnemyなら
        {
            Destroy(other.gameObject);
        }
    }
}
```

- 銃を撃つプレイヤー、撃たれる的、弾の動作の準備が完了
- あとはプレイヤーが弾を発射するだけ
- 「C01_Player」を変更

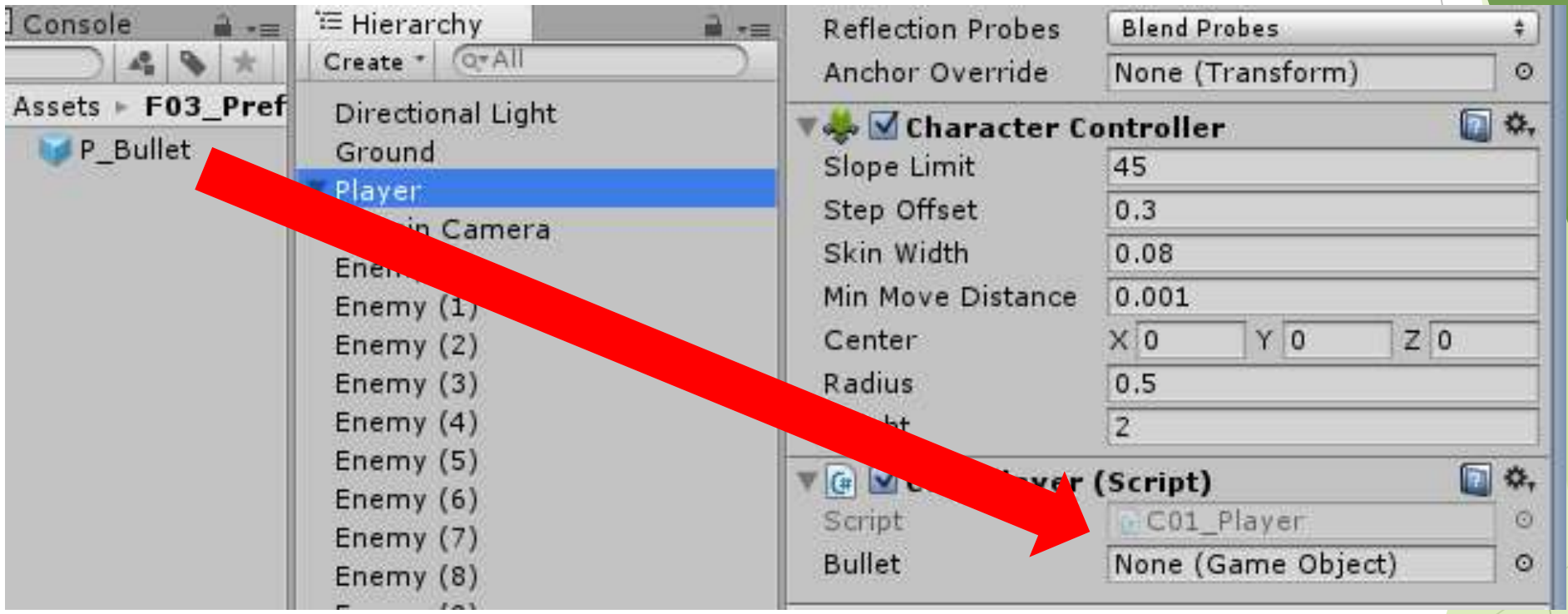
```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;    // キャラクターコンポーネント用の変数
    private Vector3 move = Vector3.zero;     // キャラ移動量.
    private float speed = 5.0f;              // 移動速度
    private float jumpPower = 10.0f;         // 跳躍力.
    private const float GRAVITY = 9.8f * 2; // 重力
    private float rotationSpeed = 180.0f;    // プレイヤーの回転速度
    public GameObject bullet;                //弾のオブジェクト
    GameObject cam;                          //カメラオブジェクト格納用

    void Start()
    {
        charaCon = GetComponent<CharacterController>();
        cam = GameObject.FindWithTag("MainCamera");

    }
```

- 残り
- 移動処理の下にでも書いて

```
//▼▼▼移動処理▼▼▼
charaCon.Move(move * Time.deltaTime);

//▼▼▼攻撃処理▼▼▼
if (Input.GetKeyDown(KeyCode.G))
{
    GameObject obj = Instantiate(bullet, transform.TransformPoint(0.3f, 0.5f, 1), cam.transform.rotation) as GameObject;
    obj.name = "bullet";
}
```

- Gキーを押したときプレイヤーの前あたりにカメラの向きで弾を生成する

- 出来たら「F03_Prefab」フォルダのP_Bulletを
  ヒエラルキの「Player」のインスペクタ内C01_PlayerのBulletに追加



- ここまでやったらゲーム実行
- 自分の向いてる向きに弾が出て、的に当たったとき的が消える？

4. 射撃にエフェクトをつける

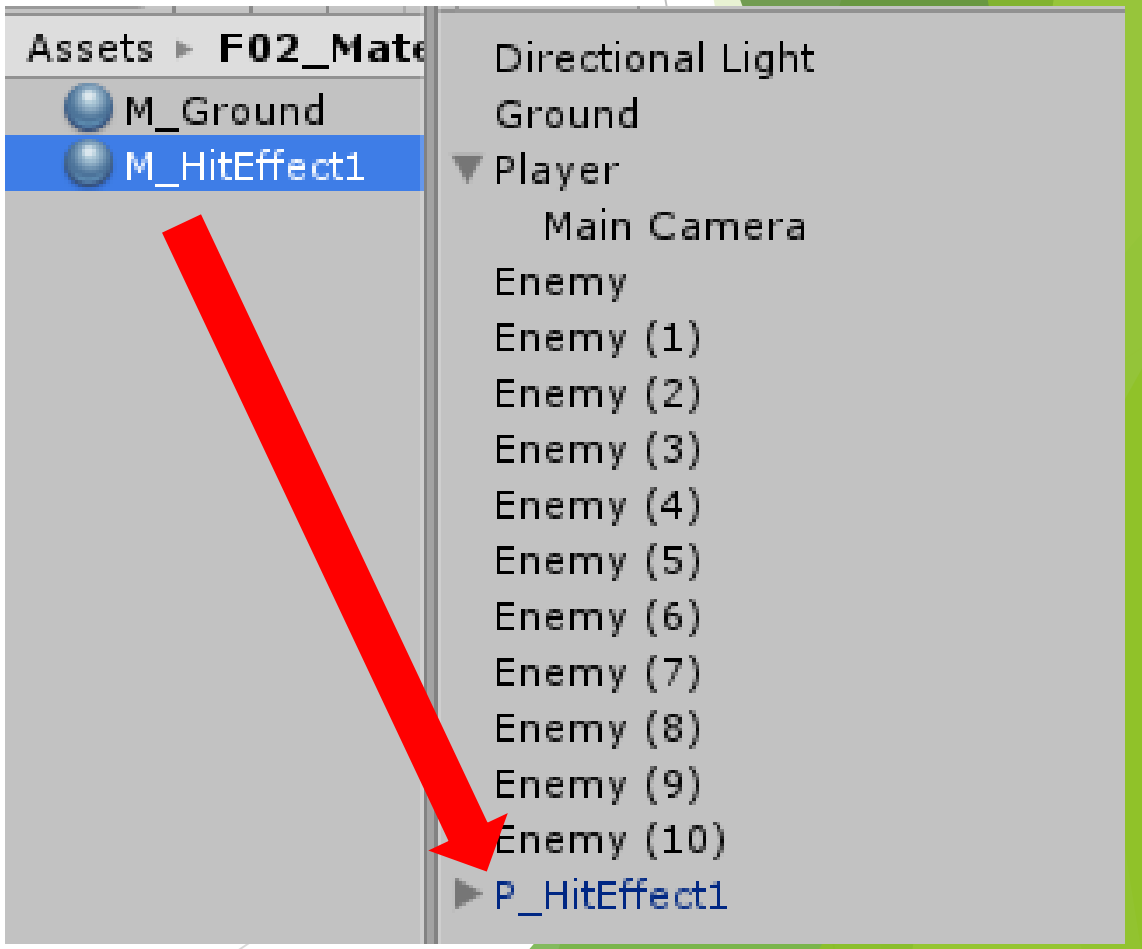
- 弾のヒット時、爆発エフェクトが出た方がかっこいい
- つくるの大変…
- そんな時はアセットストア

- 「KY」「magic」で検索。「KY Magic Effects Free」をインポート
- 「Assets」に「KY _effects」というフォルダができる
- 「KY_Effects」→「MagicEffectPackFree」→「Prefab」→「skillAttack」を選択
  選択した状態で「Ctrl + D」
- できたコピーを「F03_Prefab」に移動
- 「F03_Prefab」に移動したものの名前を「P_HitEffect1」に変更

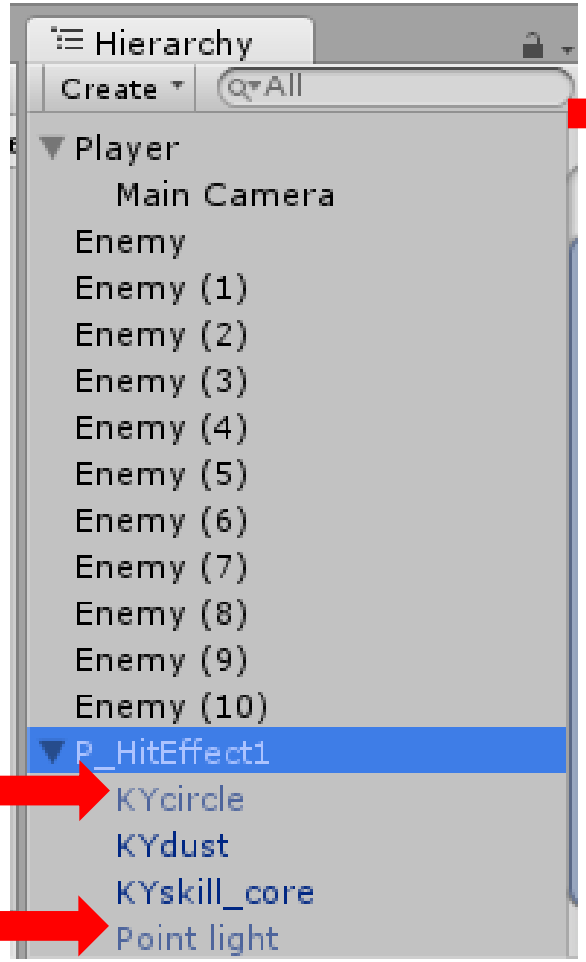
- 同様に「KY_Effects」→「MagicEffectPackFree」→「Material」→「hit4x2」を選択した状態で「Ctrl + D」
- コピーしたものを「F02_Material」に移動、名前は「M_HitEffect1」

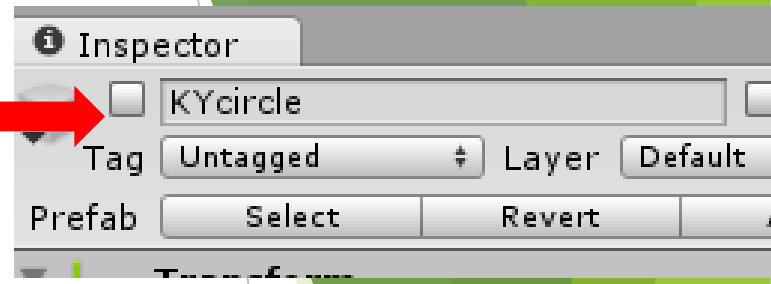
- 「P_HitEffect1」をヒエラルキにドラッグ&ドロップ

- 「M_HitEffect1」をヒエラルキの「P_HitEffect1」にドラッグ&ドロップ

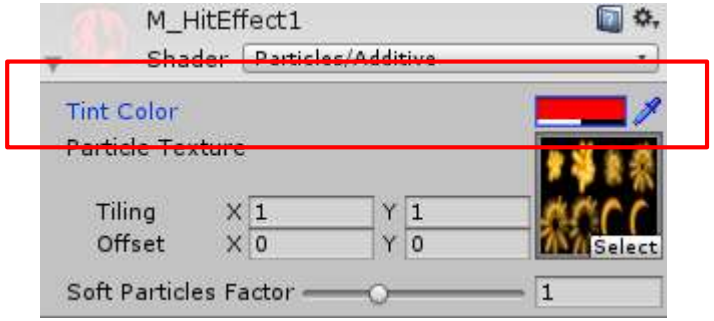


- ヒエラルキの「P_HitEffect1」を開く
- 子オブジェクトである
  「KYcircle」と「Point Light」のインスペクタのレ点をそれぞれ外す





- 「P_HitEffect1」のインスペクタ下
  さっき追加した「M_HitEffect1」の欄を見る
- 「Tint Color」を赤っぽい色に変える
  （爆発は緑がいいって人は緑とか青とかで）

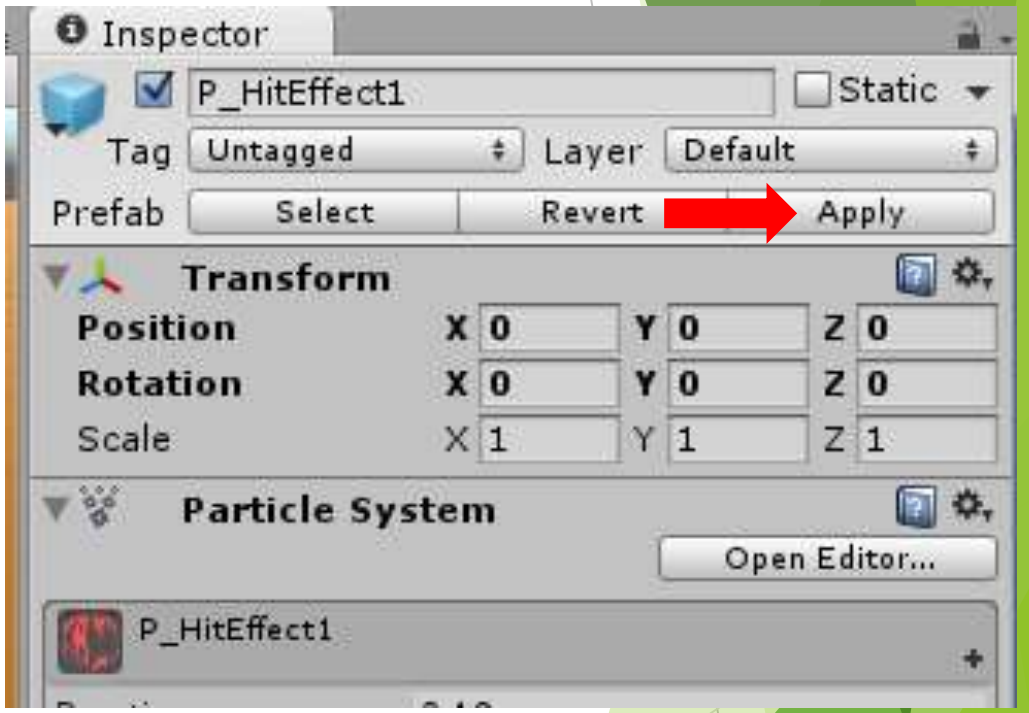


- ここまでいじった「P_HitEffect1」はヒエラルキのものであり
  元のプレハブ化した方にはデータが反映されてないので

- ヒエラルキの「P_HitEffect1」のインスペクタ右上
- 「Apply」というボタンを押す

- これでプレハブ化した元の方にデータが反映される

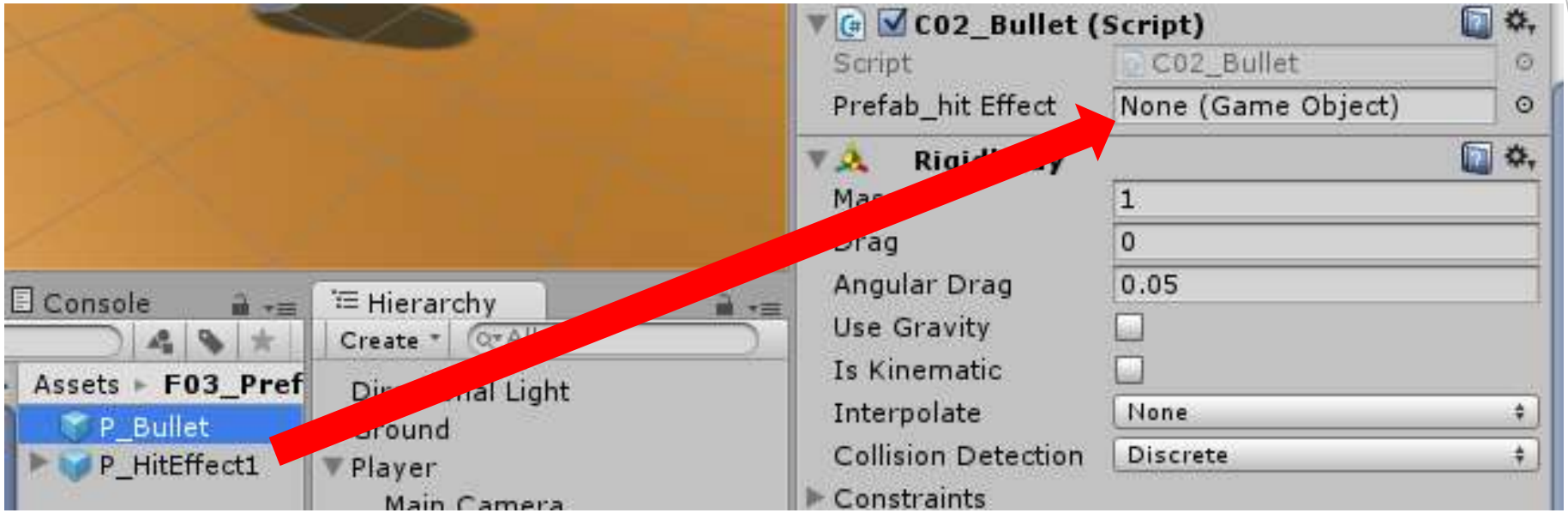
- 反映できたらヒエラルキの「P_HitEffect1」は削除



- 弾の衝突時にエフェクトを呼び出す

```
public class C02_Bullet : MonoBehaviour
{
    private float bulletSpeed = 7.0f; //弾の速度
    private const float LIFETIME = 3.0f; //弾の消滅時間
    private float time = 0.0f; //生成からの時間
    public GameObject prefab_hitEffect; //着弾のエフェクト

    // Update is called once per frame
    void Update()
    {
        time += Time.deltaTime;
        if (time > LIFETIME)
        {
            Destroy(gameObject);
        }
        transform.Translate(0.0f, 0.0f, bulletSpeed * Time.deltaTime);
    }

    //■■オブジェクトに衝突したときに呼び出される■■
    void OnCollisionEnter(Collision other)
    {
        Destroy(gameObject); //弾オブジェクトを破壊
        GameObject effect = Instantiate(prefab_hitEffect, transform.position, Quaternion.identity) as GameObject; //ヒットエフェクト生成
        if (other.gameObject.tag == "Enemy") //衝突がEnemyなら
        {
            Destroy(other.gameObject);
        }
        Destroy(effect, 0.2f); //エフェクトを破壊
    }
}
```

- P_Bulletのインスペクタ内「C02_Bullet」欄のprefab_hitEffectに
  P_HitEffect1をドラッグ&ドロップ



- ゲーム実行
- 弾が的に当たった時、エフェクトが発生する

5. 武器の追加

- 今回は手榴弾を追加
- Assetsの「F01_Script」で右クリック「Create」→「C# Script」
- 名前は「C03_Weapon」
- 内容は→
- 「Player」にドラッグ&ドロップで追加

```
using UnityEngine;
using System.Collections;

public class C03_Weapon : MonoBehaviour {
    private int type = 0;        // 武器タイプ
    private int num = 2;         // 武器の種類数

    // ■■武器を変更■■
    public void changeWeapon()
    {
        type = (type + 1) % num;
        Debug.Log("現在の武器：" + type);
    }
}
```

- 「C01_Player」に追加

- 先ほどのスクリプトを使うための変数と準備
- 武器変更部分はUpdateの最初に書いとく

- Cキーを押したときに武器を変える関数を呼び出す

- ゲーム実行してCを押す
- コンソールにどんどん表示される

```
GameObject cam;                                    //カメラオブジェクト格
private C03_Weapon weapon;                         //武器スクリプト格納

void Start()
{
    charaCon = GetComponent<CharacterController>();
    cam = GameObject.FindWithTag("MainCamera");
    weapon = GetComponent<C03_Weapon>();
}

void Update()
{
    //▼▼▼武器変更▼▼▼
    if (Input.GetKeyDown(KeyCode.C))
    {
        weapon.changeWeapon();
    }

    //▼▼▼移動量取得▼▼▼
    float y_move;
```

- 「C03_Weapon」に追加

- 武器変更メソッドの下に書く
- これで今の武器が何かを返してもらう

```
// ■■■武器を変更■■■
public void changeWeapon()
{
    type = (type + 1) % num;
    Debug.Log("現在の武器：" + type);
}

// ■■■武器タイプを返す■■■
public int getType()
{
    return type;
}
```

- 手榴弾は一度おいといて…
- 「C01_Player」がごちゃっとしてきたのでまとめる

- 武器変更は「Change_Weapon」
- 攻撃は「Attack_Weapon」
- 移動は「Player_Moving」
- という関数にそれぞれまとめる

```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;   // キャラクターコンポーネント用の変数
    private Vector3 move = Vector3.zero;    // キャラ移動量.
    private float speed = 5.0f;             // 移動速度
    private float jumpPower = 10.0f;        // 跳躍力.
    private const float GRAVITY = 9.8f * 2; // 重力
    private float rotationSpeed = 180.0f;   // プレイヤーの回転速度
    public GameObject bullet;               //弾のオブジェクト
    GameObject cam;                         //カメラオブジェクト格納用
    private C03_Weapon weapon;              //武器スクリプト格納

    void Start()
    {
        charaCon = GetComponent<CharacterController>();
        cam = GameObject.FindWithTag("MainCamera");
        weapon = GetComponent<C03_Weapon>();
    }

    void Update()
    {
        //▼▼▼武器変更▼▼▼
        if (Input.GetKeyDown(KeyCode.C))
        {
            Change_Weapon();
        }

        //▼▼▼攻撃処理▼▼▼
        if (Input.GetKeyDown(KeyCode.G))
        {
            Attack_Weapon();
        }

        //▼▼▼移動処理▼▼▼
        Player_Moving();
    }
```

```
//■■■プレイヤーの移動■■■
private void Player_Moving()
{
    float y = move.y;
    move = new Vector3(0.0f, 0.0f, Input.GetAxis("Vertical"));
    move = transform.TransformDirection(move);
    move *= speed;

    //▼▼▼ジャンプ処理▼▼▼
    move.y += y;
    if (charaCon.isGrounded)
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            move.y = jumpPower;
        }
    }
    move.y -= GRAVITY * Time.deltaTime;

    // ▼▼▼プレイヤーの向き変更▼▼▼
    Vector3 playerDir = new Vector3(Input.GetAxis("Horizontal"), 0.0f, 0.0f);
    playerDir = transform.TransformDirection(playerDir);
    if (playerDir.magnitude > 0.1f)
    {
        Quaternion q = Quaternion.LookRotation(playerDir);
        transform.rotation = Quaternion.RotateTowards(transform.rotation, q, rotationSpeed * Time.deltaTime);
    }
    charaCon.Move(move * Time.deltaTime);
}
```

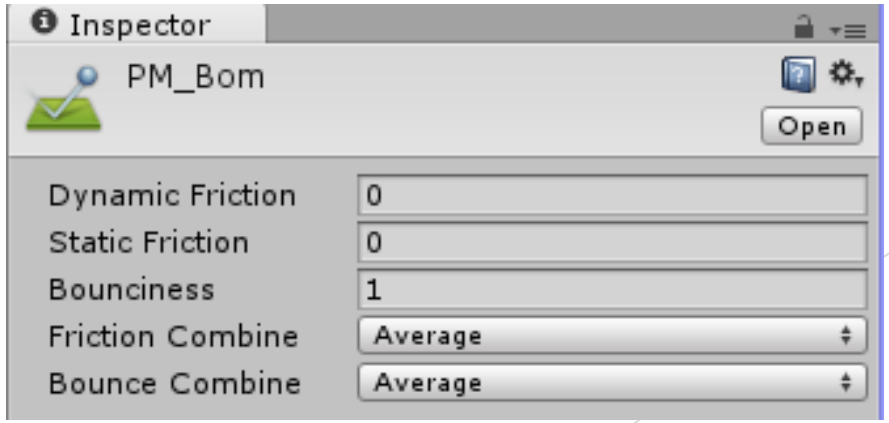
```
//■■■Gキーで攻撃■■■
private void Attack_Weapon()
{
    switch (weapon.getType())
    {
        case 0:
            GameObject obj = Instantiate(bullet, transform.TransformPoint(0.3f, 0.5f, 1), cam.transform.rotation) as GameObject;
            obj.name = "bullet";
            break;
    }
}

//■■■Cキーで武器変更■■■
private void Change_Weapon()
{
    weapon.changeWeapon();
}
```

- ここまで出来たら手榴弾に戻ります。

- 「GameObject」→「3D Object」→「Capsule」
- 名前は「P_Bom」、Scaleは（0.2, 0.2, 0.2）
- インスペクタから「Add Compoent」→「Physics」→「Rigidbody」を追加

- 「F02_Material」で右クリック「Create」→「Material」
- 名前は「M_Bom」
- 適当に色を決める（個人的には深緑）

- 「F02_Material」で右クリック「Create」→「Physic Material」
- 名前は「PM_Bom」
- 右の画像と同じ設定に



- 「M_Bom」と「PM_Bom」を「P_Bom」にドラッグ&ドロップ

- 「F01_Script」で右クリック「Create」→「C# Script」
- 名前は「C04_Bom」

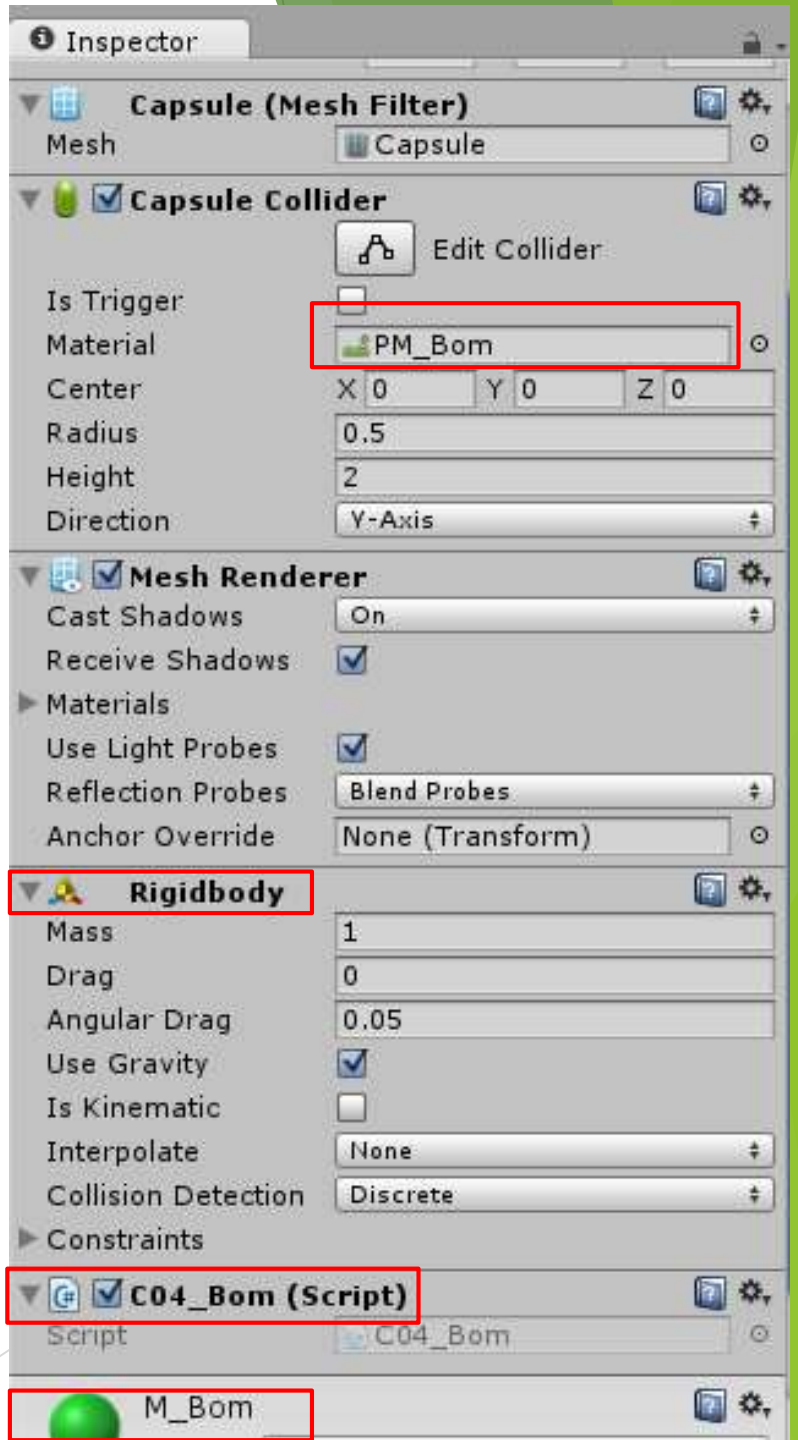
- 中身→

- 書けたら「P_Bom」
  にドラッグ&ドロップ

- コルーチン？？
- 説明は後で
  まずは手榴弾を完成させます

```
using UnityEngine;
using System.Collections;

public class C04_Bom : MonoBehaviour {
    void Start()
    {
        StartCoroutine("bom");  // コルーチン開始
    }

    //■■■bom コルーチン■■■
    IEnumerator bom()
    {
        yield return new WaitForSeconds(2.5f);  // 2.5秒、処理を待機.
        Destroy(gameObject);
    }
}
```

- 「P_Bom」が→になったら「F03_Prefab」にドラッグ&ドロップ
- プレハブ化したらヒエラルキの「P_Bom」は削除
- 「C01_Player」をいじって手榴弾を使えるようにする

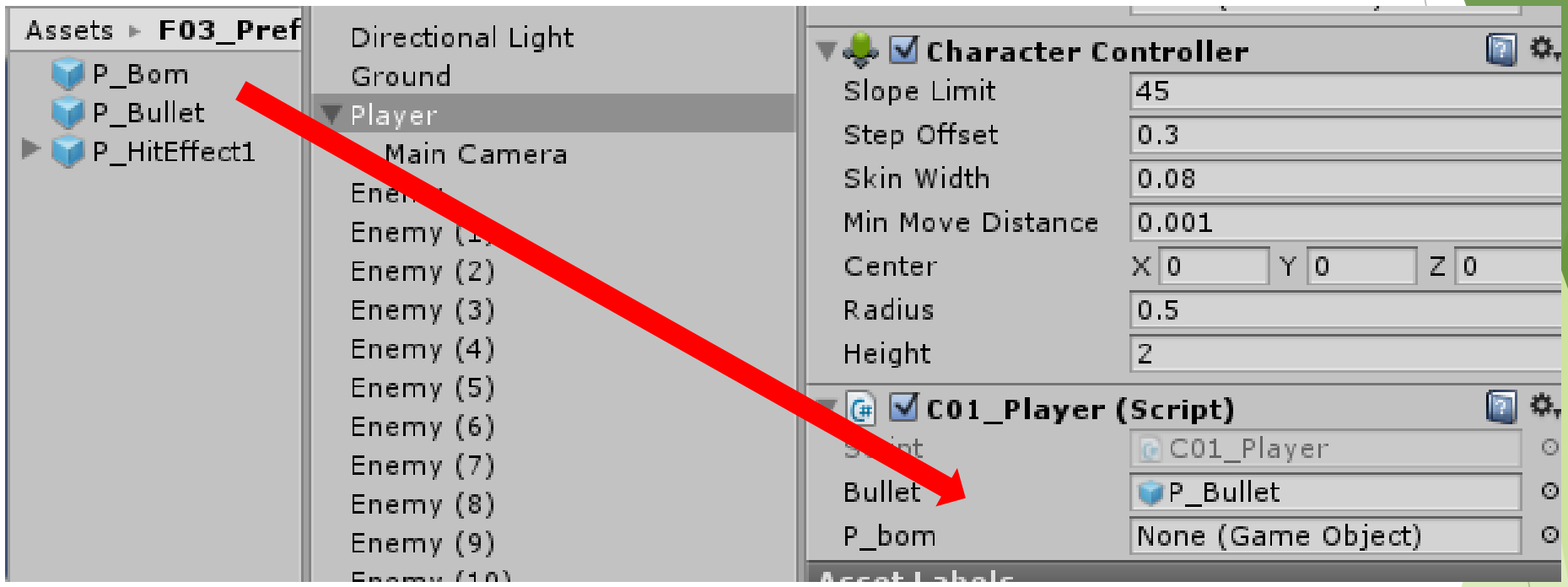


- 手榴弾のオブジェクト格納用の変数とswitch文のケースを追加

```
public class C01_Player : MonoBehaviour {
    private CharacterController charaCon;    // キャラクターコンポーネン
    private Vector3 move = Vector3.zero;     // キャラ移動量.
    private float speed = 5.0f;              // 移動速度
    private float jumpPower = 10.0f;         // 跳躍力.
    private const float GRAVITY = 9.8f * 2; // 重力
    private float rotationSpeed = 180.0f;    // プレイヤーの回転速度
    public GameObject bullet;                //弾のオブジェクト
    public GameObject p_bom;                   //手榴弾のオブジェクト
    GameObject cam;                          //カメラオブジェクト格納用
                                             納
```

```
//■■■Gキーで攻撃■■■
private void Attack_Weapon()
{
    switch (weapon.getType())
    {
        case 0:
            GameObject obj = Instantiate(bullet, transform.TransformPoint(0.3f, 0.5f, 1), cam.transform.rotatio
            obj.name = "bullet";
            break;
        case 1:
            Vector3 pos = transform.position + transform.TransformDirection(Vector3.forward);
            GameObject bom = Instantiate(p_bom, pos, Quaternion.identity) as GameObject;

            Vector3 bom_speed = transform.TransformDirection(Vector3.forward) * 5;
            bom_speed += Vector3.up * 5;
            bom.GetComponent<Rigidbody>().velocity = bom_speed;
            bom.GetComponent<Rigidbody>().angularVelocity = Vector3.forward * 7;
            break;

    }
}
```

- 「Player」のインスペクタ、「C01_Player」に「P_bom」が追加されているので「F03_Prefab」の「P_Bom」をドラッグド&ロップ

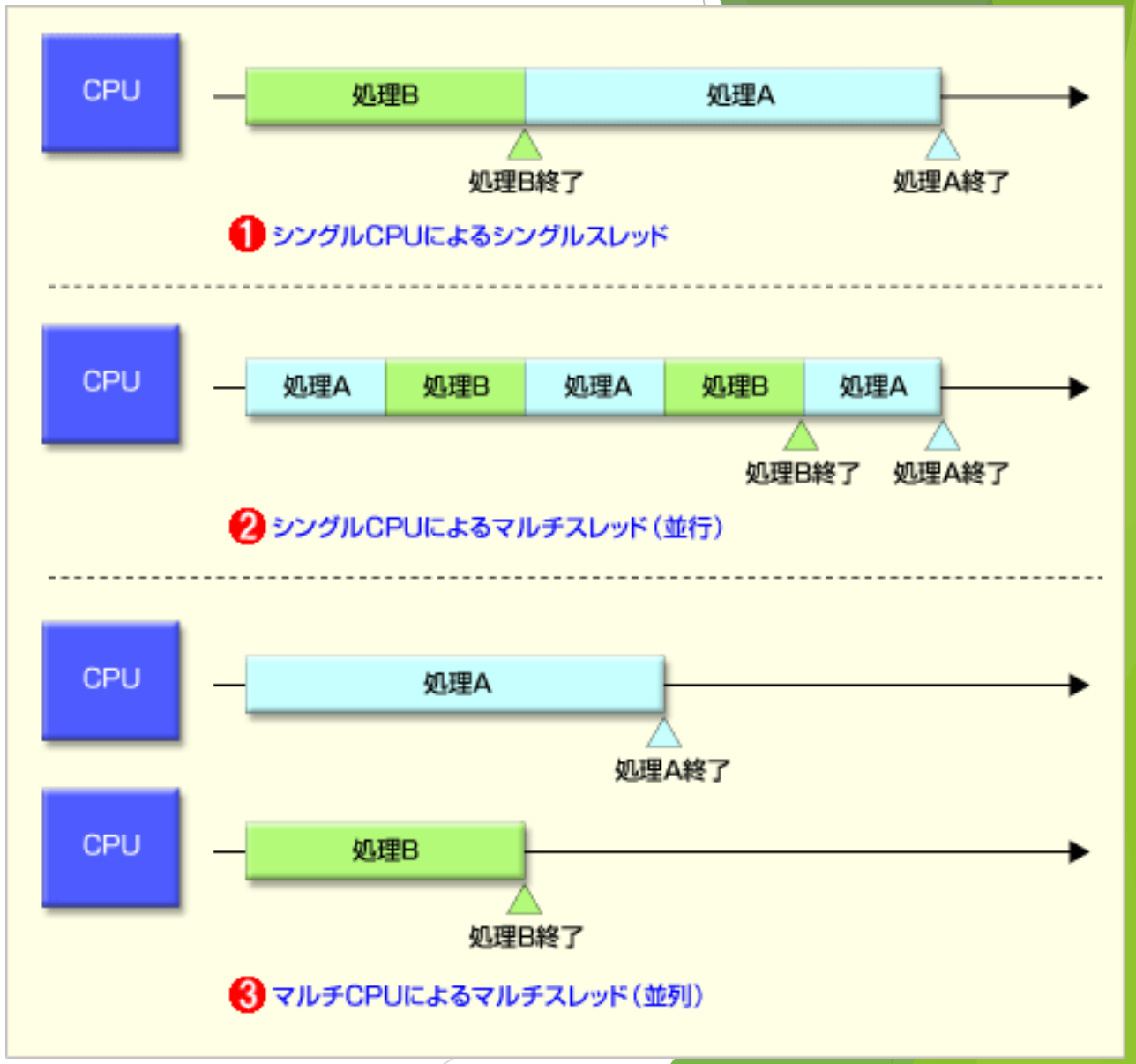


- ゲーム実行
- 最初は銃、Cを押した後は手榴弾、またCを押すと銃...
  と攻撃が変化することを確認

- コルーチンについて
- マルチスレッドのお話し

- シングルスレッド（たぶん今まではこれ）
  - ・ある処理が終了したら次の処理を始める
  - ・後に待ってる処理は全く進まない

- マルチスレッド
  - ・ある処理を少し進めたら一旦停止し、
    他の処理を少し進める
  - ・全体が少しずつ処理されていく



- Unityではコルーチンを使ってマルチスレッドを行う
- さっきの手榴弾のスクリプトではUpdate関数を書いていないが
  2.5秒経過でオブジェクトが消えるはず

- これは手榴弾生成時にコルーチンを呼び出すことで他のUpdate関数と並列で
  処理を行っているから

- 無理に使わずともプログラムは書けるけど使うと便利！って場面があるよ～

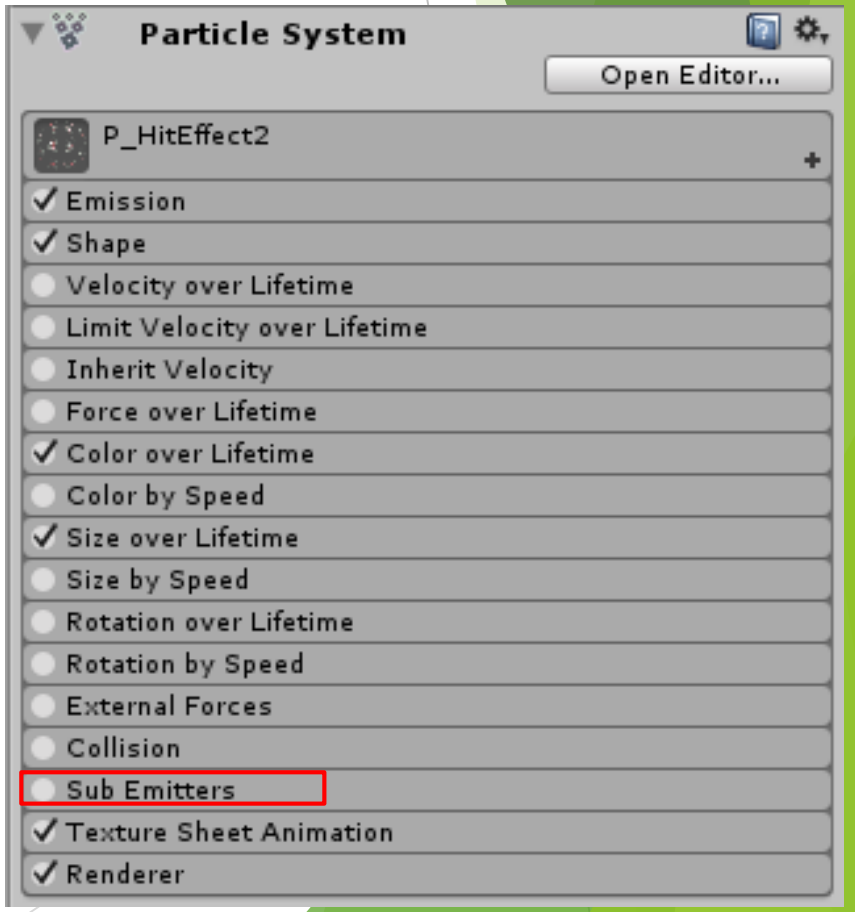
- マルチスレッドはいつか授業とかでちゃんとした説明を受けると思うので
  ここでの話はそんなもんあるんだー、くらいで

- 続けて手榴弾をいじります
- 今は投げて消えるだけなのでエフェクトを追加

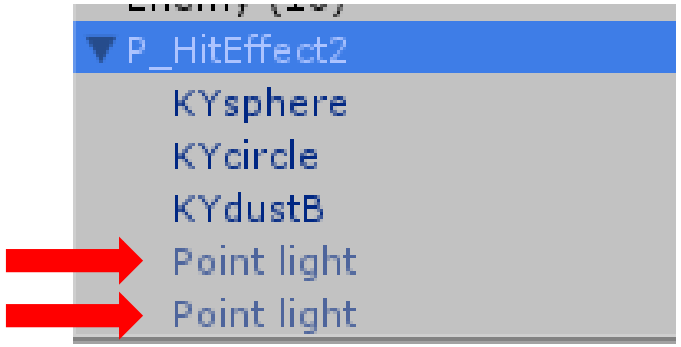
- Assetsの「KY_effects」の中にある「Prefab」内「WhiteBomb」を選択して「Ctrl + D」
- 出来たコピーを「F03_Prefab」にドラッグ&ドロップ
- 名前は「P_HitEffect2」

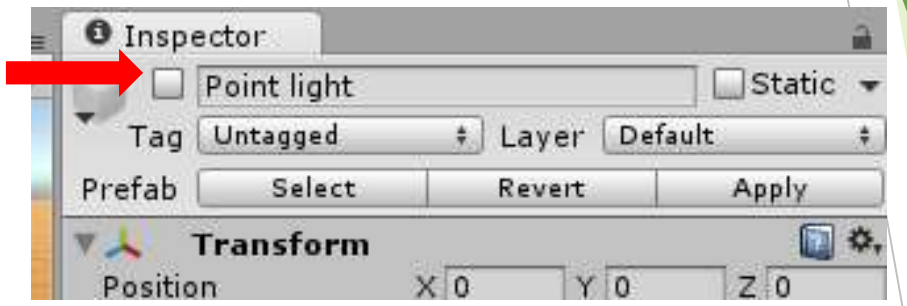
- 「P_HitEffect2」をヒエラルキにドラッグ&ドロップ

- まずインスペクタの「Sub Emitters」のレ点を外す

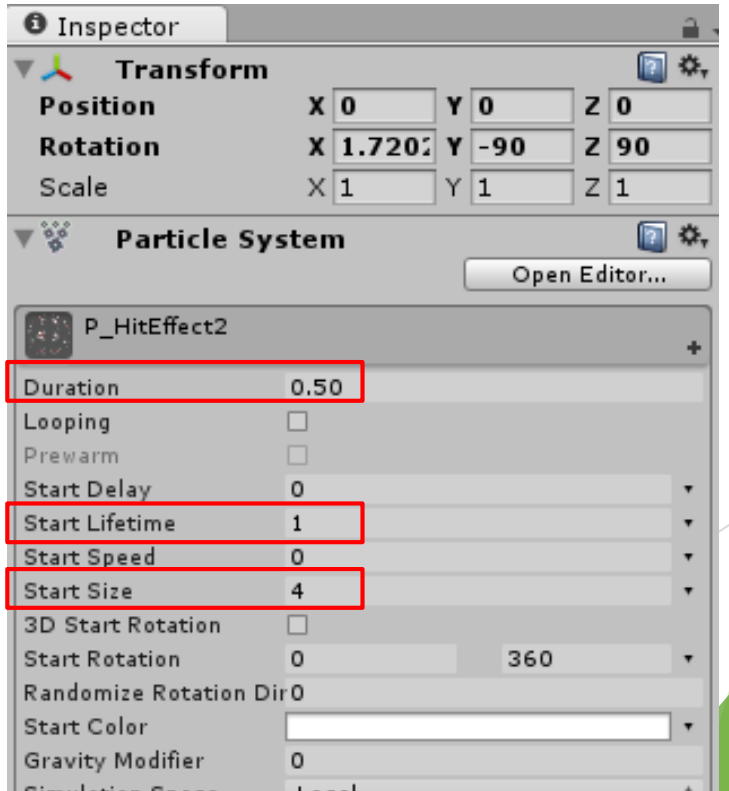


- ヒエラルキの「P_HitEffect2」の子オブジェクトから2つのライトを選択
- それぞれレ点を外す





- 「P_HitEffect2」のインスペクタの値を→と同じにする

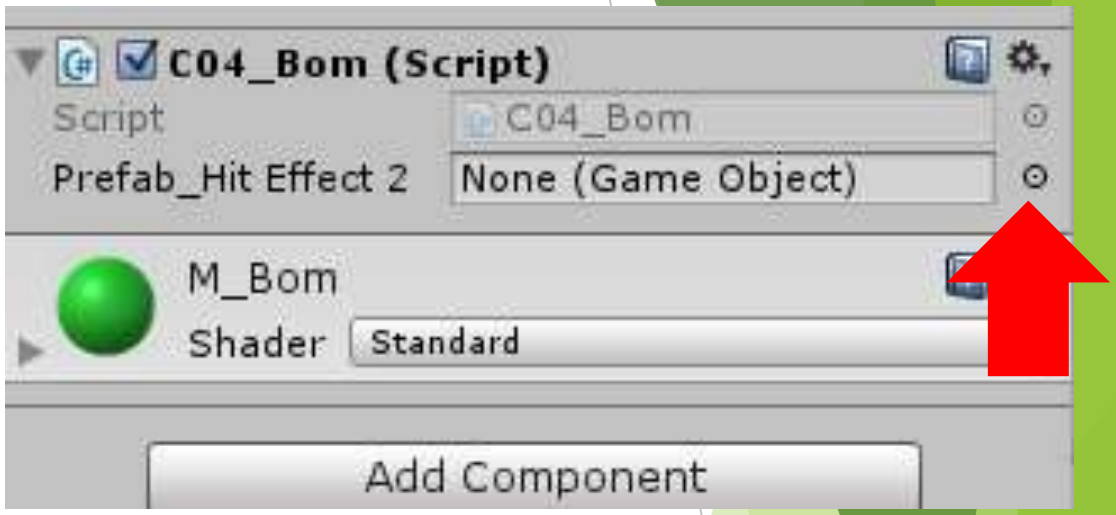


- できたら右上の「Apply」を押す
- 「F03_Prefab」に反映出来たらヒエラルキのは削除

- 「C04_Bom」に追加してエフェクトが出るようにする

```
public class C04_Bom : MonoBehaviour {
    public GameObject prefab_HitEffect2;

    void Start()
    {
        StartCoroutine("bom");  // コルーチン開始
    }

    //■■■bom コルーチン■■■
    IEnumerator bom()
    {
        yield return new WaitForSeconds(2.5f);  // 2.5秒、処理を待機
        GameObject effect = Instantiate(prefab_HitEffect2 , transform.position , Quaternion.identity) as GameObject;    // エフェクト発生
        Destroy(effect, 1.0f);      // エフェクトを、１秒後に消滅させる
        Destroy(gameObject);        //手榴弾自体を消す
    }
}
```

- 「F03_Prefab」の「P_Bom」のインスペクタを確認
- 「Prefab_HitEffect2」というとこの丸いやつをクリック



- 検索ウィンドウが出てくる
- どこかに「P_HitEffect2」があるので探して選択

- ゲーム実行
- 爆発することを確認

- 最後に爆発で敵を倒せるようにする
- 「C04_Bom」をいじる
- 関数追加と呼び出し

```
void Start()
{
    StartCoroutine("bom");  // コルーチン開始
}

//■■■bom コルーチン■■■
IEnumerator bom()
{
    yield return new WaitForSeconds(2.5f);  // 2.5秒、処理を待機
    GameObject effect = Instantiate(prefab_HitEffect2 , transform.position , Quaternion.identity) as GameObject;    // エフェクト発生
    bomAttack();
    Destroy(effect, 1.0f);      // エフェクトを、１秒後に消滅させる
    Destroy(gameObject);        //手榴弾自体を消す
}

// ■■■ボムによる攻撃処理■■■
private void bomAttack()
{
    Collider[] targets = Physics.OverlapSphere(transform.position, 1.5f);
    foreach (Collider obj in targets)
    {
        if (obj.tag == "Enemy")
        {
            Destroy(obj.gameObject);
        }
    }
}
```

- 攻撃でやってること
  - ・手榴弾から1.5f以内のオブジェクトを配列に格納
  - ・配列内のオブジェクトのタグを確認
  - ・タグが「Enemy」であるオブジェクトを破壊する
- これだけ

- Collider[] targets～の1文、1.5fを変更すると当たり範囲を変えられます

- ゲーム実行
- 爆発時に手榴弾の近くにある的が消えることを確認

本日はここまで

- FPSの基礎的な部分を作りました
- 次回は細かなところを仕上げていきたいと思います

- もし色々自分でいじりたい人がいたら
  作ったプロジェクトをコピーして、コピーしたプロジェクトをいじってください
- 今日作ったものは次も使うので

お疲れさまでした

この後も質問等は受け付けます