Unity講座
～FPS的な(2)～

3年　　海苔　威

目次

1. 手榴弾に制限を付ける
2. 銃に制限を付ける
3. UIを作る
4. 敵を用意する
5. 敵の移動と生成

1.手榴弾に制限を付ける

- いまのままだと手榴弾を無限に投げれる
- 投げた後数秒間は投げることが出来ないようにする
- 「C01_Player」を変更（ここから3ページ続きます）

- 制限用の変数追加

```
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
private bool used_bom = false;     //手榴弾の攻撃制限
```

- さらにAttack_Weaponに追加、if文で囲む

```
//■■■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.trans
            obj.name = "bullet";
            break;
        case 1:
            if (!used_bom)
            {
                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;

                used_bom = true;
                StartCoroutine("bomCharge");
            }
            break;

    }
}
```

- Attack_Weaponの下にコルーチン追加

```
//■■■bomChargeコルーチン■■■
IEnumerator bomCharge()
{
    yield return new WaitForSeconds(3.0f);
    used_bom = false;
}
```

- ゲーム実行
- 手榴弾を投げてから3秒間は次が投げれないはず

- 制限するbool変数used_bomを用意
  - ・true→使用中
  - ・false→未使用

- 攻撃時（Gを押したとき）
  - ・true→何もしない
  - ・false→手榴弾で攻撃 + used_bomをtrueに + コルーチンbomChargeを呼ぶ

- コルーチンbomCharge

  3秒待機して、経過したらused_bomをfalseに

- 以上で手榴弾を投げてから3秒間は投げれないようにしている

2. 銃に制限を付ける

- さっきと同様に「C01_Player」をいじる（ここから3ページ続きます）
- 銃弾の変数と初期化を追加

```
private bool used_bom = false;            //手榴弾の攻撃制限
private int bullet_num;                   //銃弾の残弾数
private const int BULLET_MAX_NUM = 30;    //銃弾の最大装填数

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

- Attack_Weaponに追加

```
//■■■Gキーで攻撃■■■
private void Attack_Weapon()
{
    switch (weapon.getType())
    {
        case 0:
            if (bullet_num == 0)
            {
                return;
            }

            GameObject obj = Instantiate(bullet, tr
            obj.name = "bullet";

            bullet_num--;

            if (bullet_num == 0)
            {
                StartCoroutine("bulletCharge");
            }

            break;
```

- bomChagreコルーチンの下あたりにでも

```
//■■■bulletChargeコルーチン■■■
IEnumerator bulletCharge()
{
    yield return new WaitForSeconds(3.0f);
    bullet_num = BULLET_MAX_NUM;
}
```

- ゲーム実行
- 弾を30発撃つとしばらく撃てなくて、3秒経過でまた撃ち始める

- はじめに今の銃弾数と最大数を設定

- 攻撃時に今の銃弾数が0か確認
  - ・0のとき→何もせず処理終了
  - ・0以外→弾を1発発射 + 今の銃弾数から-1 + 銃弾数が0になったか確認

- 弾を発射したうえで、銃弾数が0になったとき

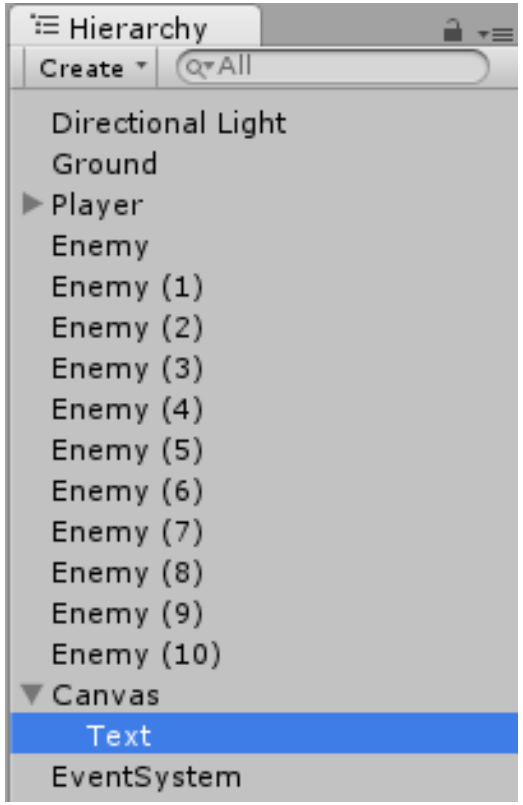
  →3秒間待機してからBULLET_MAX_NUMで更新（bullet_numに30発入る）

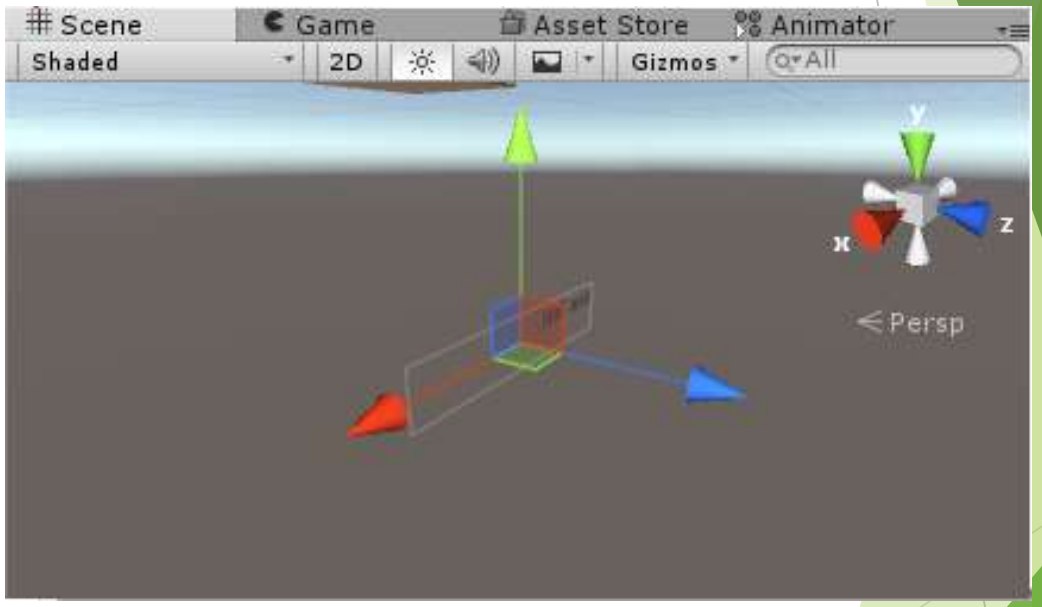
3. UIを作る

- ブロック崩しで使ったGUIではなくuGUIのキャンバスを使います
- （新しくなったUnityではこれが主流らしいです）
- （これからはこれでお願いします）

- とりあえず「GameObject」→「UI」→「Text」
- 名前はそのままで

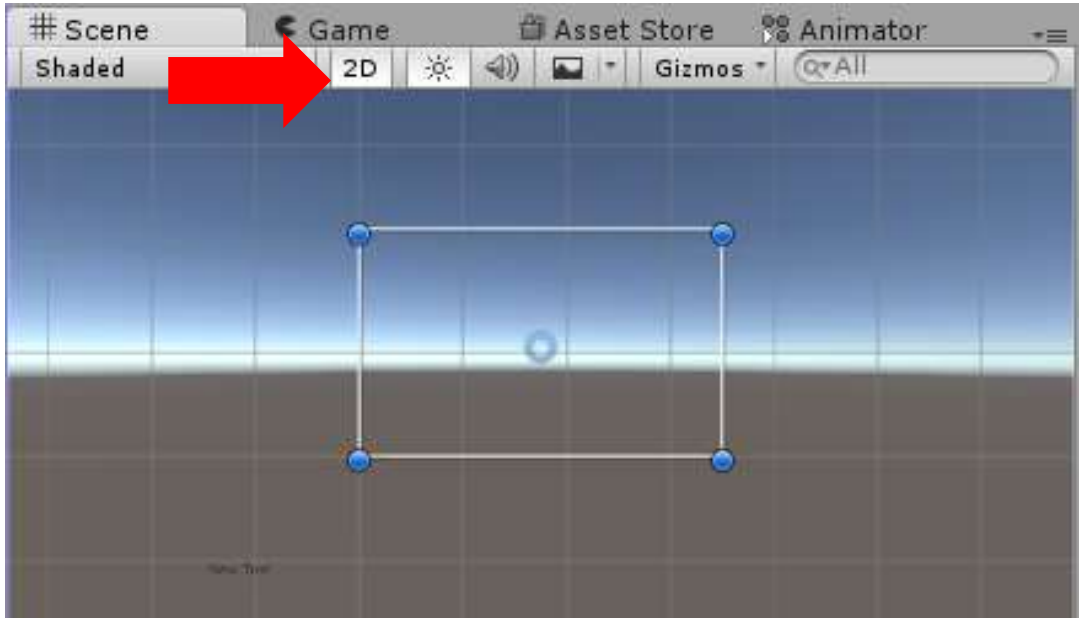
- 「Canvas」とその子オブジェクトとして「Text」
- それに「EventSystem」という3オブジェクトがヒエラルキにできる
- 「EventSystem」はそのままでも削除しても大丈夫
- 「Text」をダブルクリック





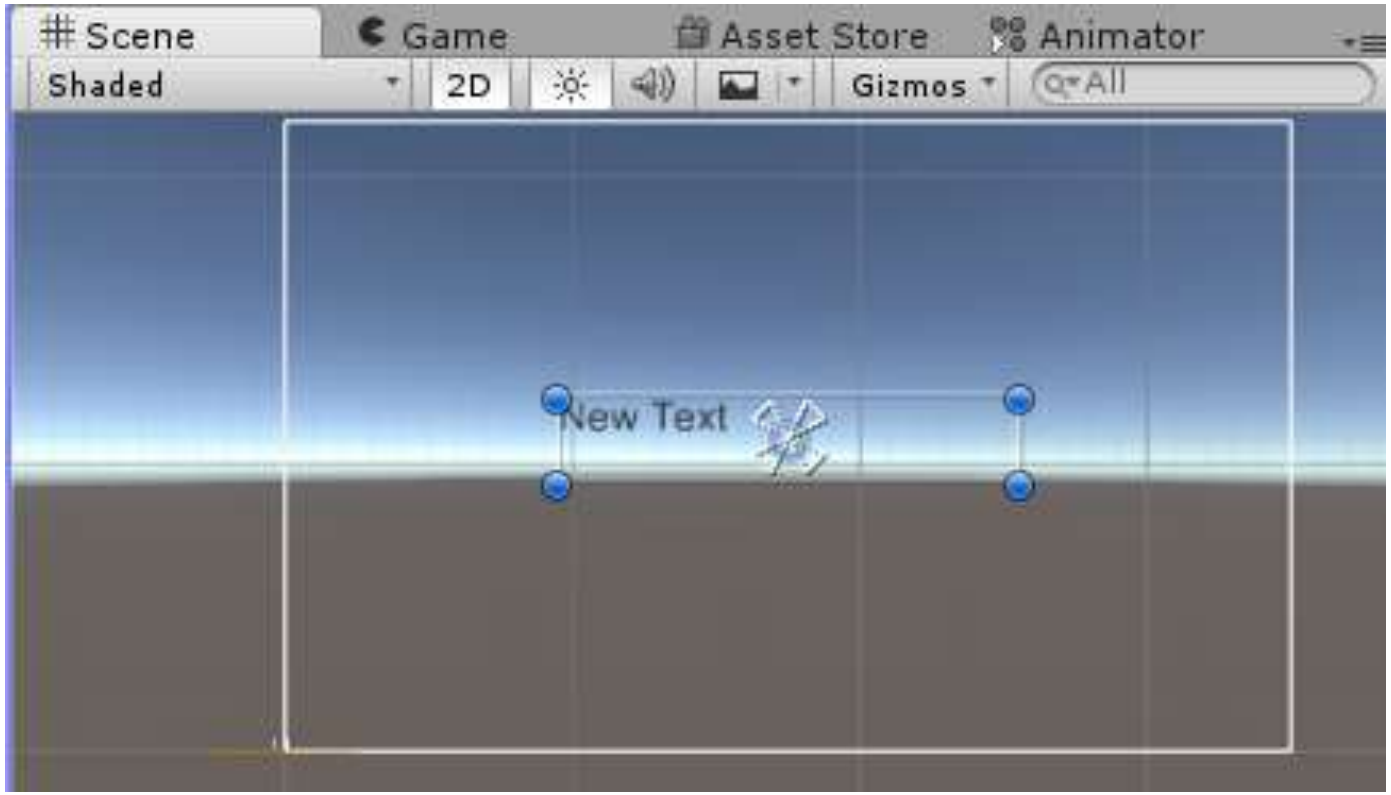
- 右の感じでシーン上のテキストが見えるが非常に見づらい

- シーンビューの2Dをクリック
- さらにヒエラルキの「Canvas」をダブルクリック



- 上みたいな表示になる
- この四角形がUIが表示される範囲
- 左下にちょこっと見えるのが「Text」で今は範囲外にあることが分かる

- 「Text」の移動と表示拡大でCanvasの中心に持ってくる

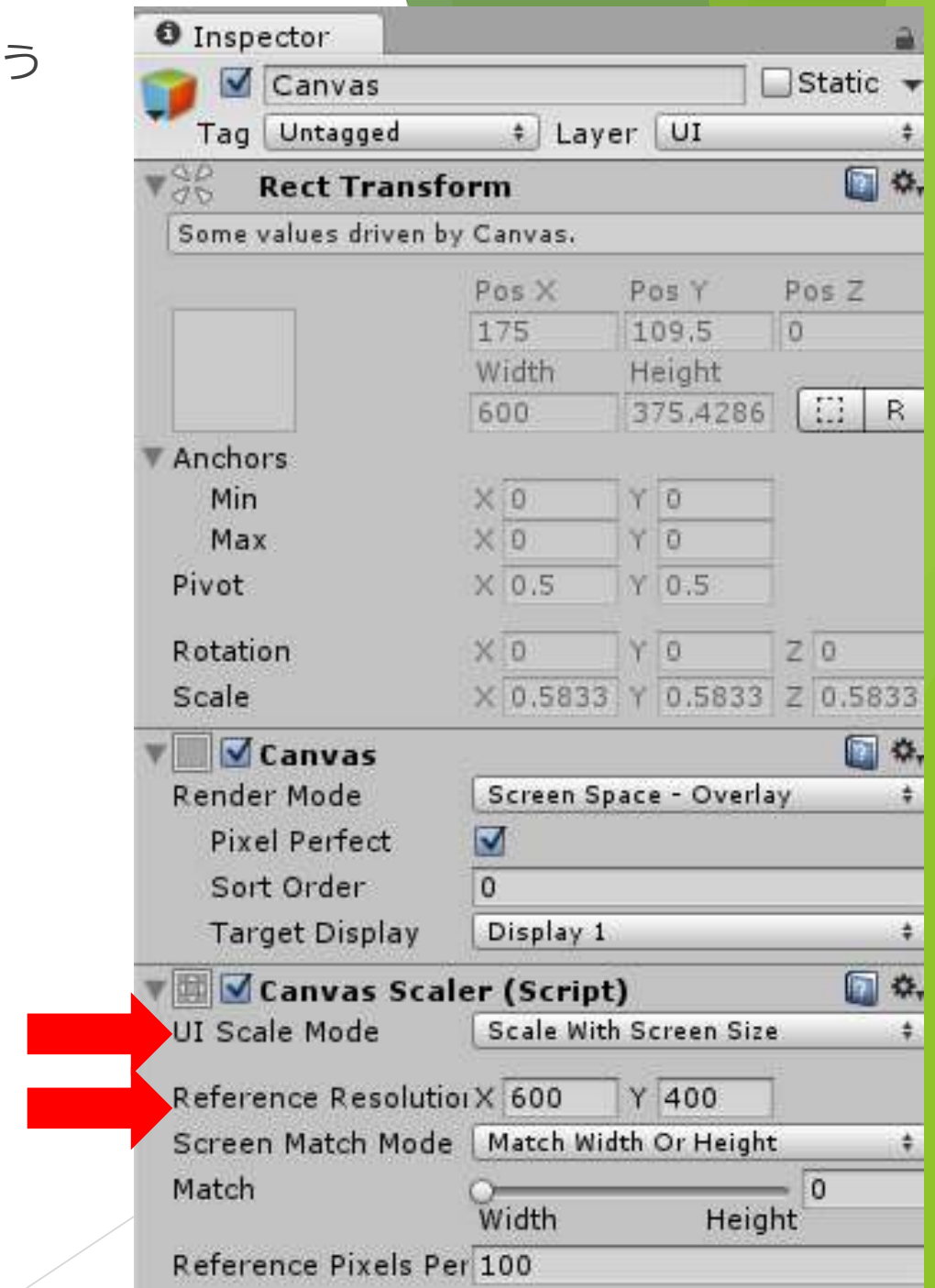


- こんな感じになるとゲーム上で中心に「New Text」という文字列が見える
- 実際にゲームビューを見ると分かる

- この状態だとスクリーンの拡大縮小で見え方が変わってしまう
- ヒエラルキの「Canvas」のインスペクタ、Canvas Scaler のUI Scale Modeを「Scale With Screen Size」に変更

- その下の「Reference Resolution」を600×400に変更
- ここでキャンバスのサイズを変えられます
- 小さくすると文字が相対的に大きくなります

- 準備ができたのでここから実際にUIを使っていきます

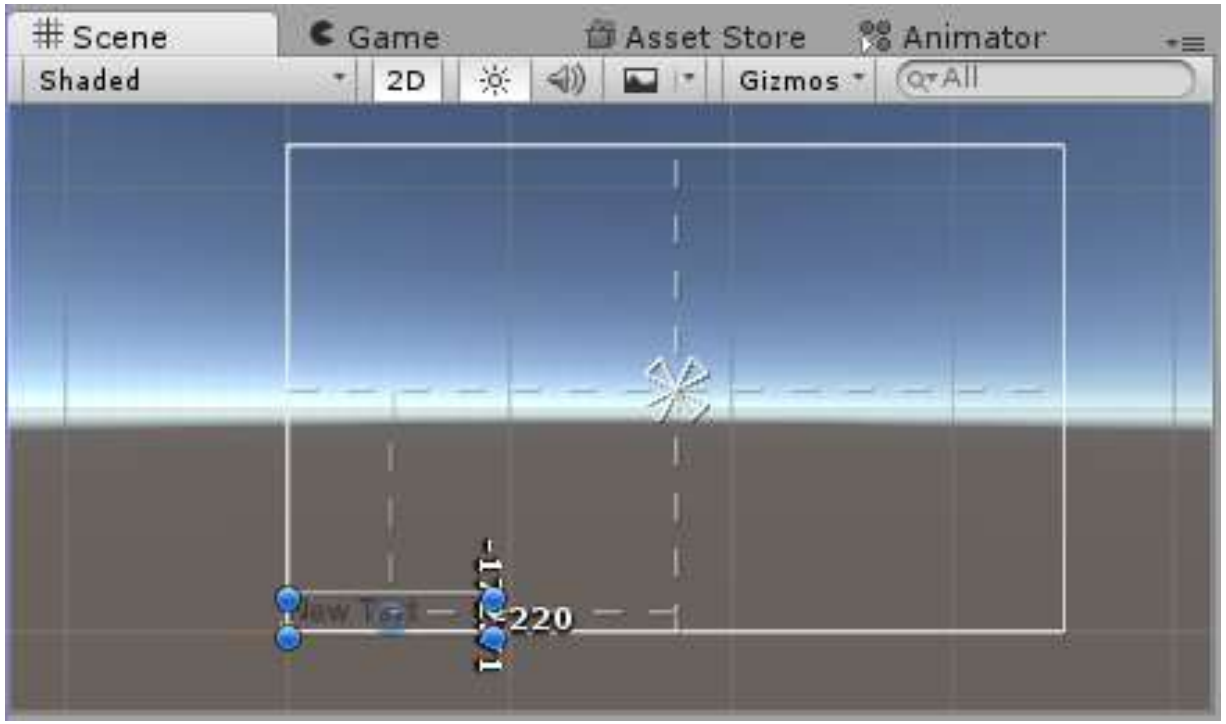


- さっきの「Text」の名前を「Text_Bullet」に変更
- 位置を中心から左下にずらす

（境界線と当たると線がオレンジに変わるのでそこで止める）

↓の画面だとオレンジじゃないですが、プリントスクリーンの仕様なので

ホントはオレンジになってます

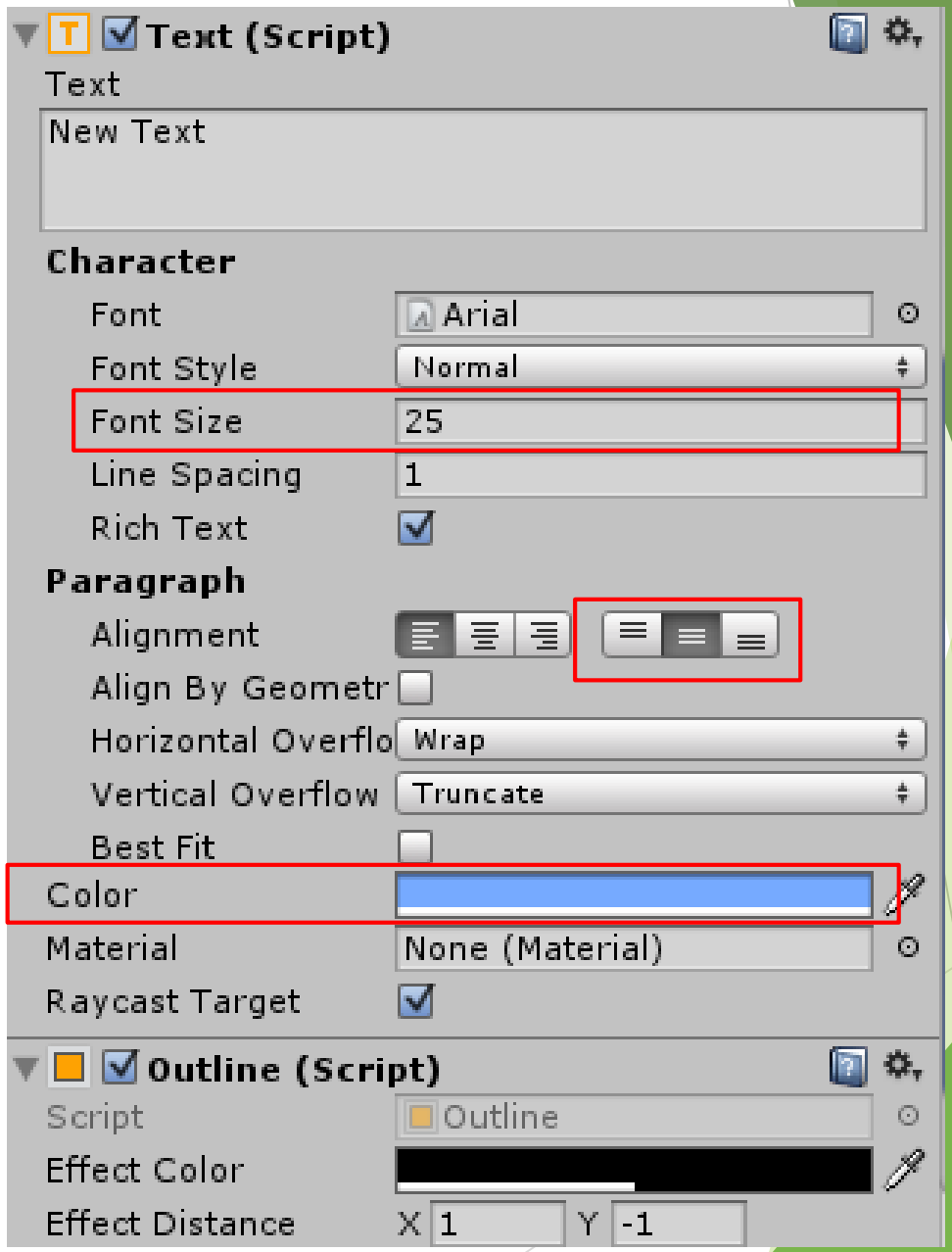


- 「Text_Bullet」のインスペクタ

- 一番下の「Add Component」から
  「UI」→「Effects」→「Outline」を追加

- いい感じに設定を変更
- ここでは3か所いじってます

- テキストの大きさ・色・位置を変え
  輪郭も追加



- スクリプトを書いて「Text_Bullet」の中身に残弾数を表示する

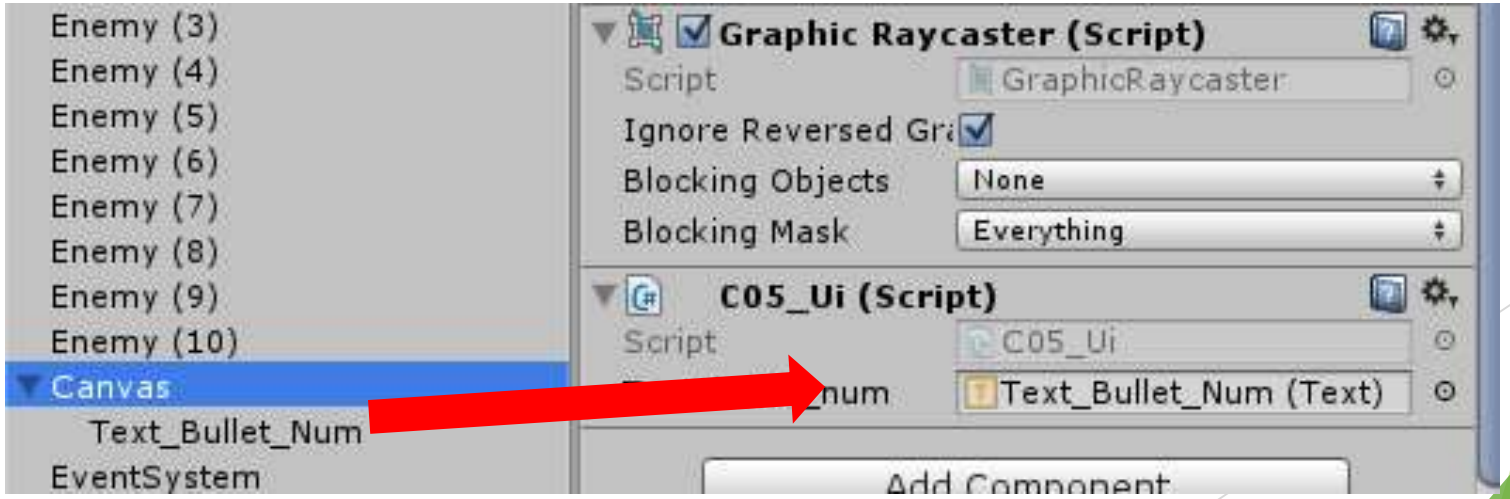
- 「F01_Script」で右クリック「Create」→「C# Script」
- 名前は「C05_Ui」

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class C05_Ui : MonoBehaviour {
    public Text text_bullet_num;

    // ■■■残弾数テキストを変更■■■
    public void changeText_BulletNum(int num)
    {
            if (text_bullet_num != null)
            {
                text_bullet_num.text = "残弾数：" + num;
            }
    }
}
```

- 重要なのは「using UnityEngine.UI」という1文
- この文を書かないとUIを使えません

- 「C05_Ui」をヒエラルキの「Canvas」に追加
- （GameRootというオブジェクトを作ってそこに持たせてもおーけーです）
  その場合は自分でスクリプトを一部変更してください

- 「Canvas」のインスペクタ、「C05_Ui」にテキストを入れるところがあるので
  そこに「Text_Buleet_Num」をドラッグ&ドロップ



- ここまでで弾数を表示する部分は完成
- あとは呼び出すだけ
- 「C01_Player」をいじる

- UIをいじる用の変数、Start関数の用意

```
private int bullet_num;                         //銃弾の残弾数
private const int BULLET_MAX_NUM = 30;          //銃弾の最大装填数
private C05_Ui ui;                              //UIのオブジェクト
```

```
void Start()
{
    charaCon = GetComponent<CharacterController>();
    cam = GameObject.FindWithTag("MainCamera");
    weapon = GetComponent<C03_Weapon>();
    ui = GameObject.Find("Canvas").GetComponent<C05_Ui>();
    bullet_num = BULLET_MAX_NUM;
    ui.changeText_BulletNum(bullet_num);
}
```

- Attack_Weapon内とbulletChargeコルーチンに1か所ずつ追加

```
case 0:
    if (bullet_num == 0)
    {
        return;
    }

    GameObject obj = Instantiate(bullet, transf
    obj.name = "bullet";

    bullet_num--;
    ui.changeText_BulletNum(bullet_num);

    if (bullet_num == 0)
    {
        StartCoroutine("bulletCharge");
    }
```

```
//■■■bulletChargeコルーチン■■■
IEnumerator bulletCharge()
{
    yield return new WaitForSeconds(3.0f);
    bullet_num = BULLET_MAX_NUM;

    ui.changeText_BulletNum(bullet_num);
}
```

- 簡単にいうと
- 「C01_Player」で弾数に変更がある度（-1や再装填で30になるとき）にテキストをいじるスクリプトに今の銃弾数を送る

- 「C05_Ui」は送られてきた数を表示する

- ゲーム実行
- 銃で攻撃すると左下の数が減って、0になって3秒経過で30に戻る

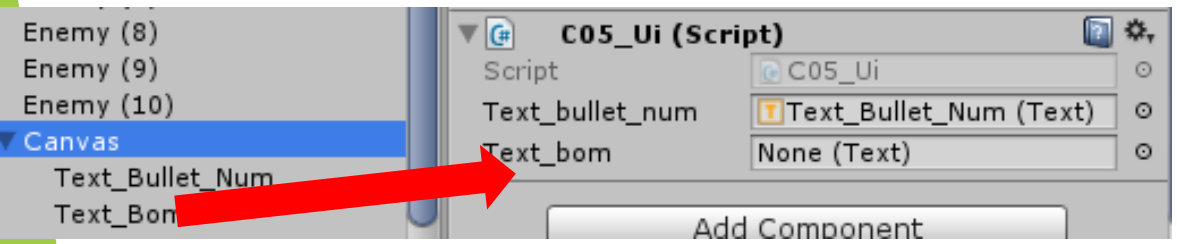
- 続けて手榴弾用のUIを作る
- ヒエラルキの「Text_Bullet_Num」を選択して「Ctrl + D」
- コピーの名前を「Text_Bom」に変更
- 「C05_Ui」に追加

```
public class C05_Ui : MonoBehaviour {
    public Text text_bullet_num;
    public Text text_bom;

    // ■■■残弾数テキストを変更■■■
    public void changeText_BulletNum(int num)
    {
            if (text_bullet_num != null)
            {
                text_bullet_num.text = "残弾数：" + num;
            }
    }

    //■■■手榴弾テキストを変更■■■
    public void changeText_Bom(bool used)
    {
        if (text_bom != null)
        {
            if (used == true)
            {
                text_bom.text = "装填中";
            }
            else
            {
                text_bom.text = "使用可";
            }
        }
    }
```

- ここまでで
  - ・テキストの初期化関数
  - ・銃のテキスト変更関数
  - ・手榴弾のテキスト変更関数
  - ・銃or手榴弾テキストのどちらを表示するか
    をtypeで判断する関数

  を用意

- スクリプトが書けたらいつも通り
  オブジェクトを追加する




```
// ■■■武器タイプによるテキストの表示オン／オフ■■■
public void changeText_enable(int type)
{
    switch (type)
    {
        case 0:     // 武器タイプが銃の場合
            text_bullet_num.enabled = true;
            text_bom.enabled = false;
            break;
        case 1:     // 武器タイプが手榴弾の場合
            text_bullet_num.enabled = false;
            text_bom.enabled = true;
            break;
    }
}

// ■■■テキスト初期化用■■■
public void initialize(int type, int num, bool used)
{
    changeText_enable(type);
    changeText_BulletNum(num);
    changeText_Bom(used);
}
```

- さっきまでに用意した関数を必要なときに呼び出す
- 「C01_Player」を変更

```
void Start()
{
    charaCon = GetComponent<CharacterController>();
    cam = GameObject.FindWithTag("MainCamera");
    weapon = GetComponent<C03_Weapon>();
    ui = GameObject.Find("Canvas").GetComponent<C05_Ui>();
    bullet_num = BULLET_MAX_NUM;
    //ui.changeText_BulletNum(bullet_num);
    ui.initialize(weapon.getType(), bullet_num, used_bom);
}
```

- コメント部分は削除で1行追加

- Attack_WeaponのSwitchのcase 1内に1行追加

```
case 1:
    if (!used_bom)
    {
        Vector3 pos = transform.position +
        GameObject bom = Instantiate(p_bom,

        Vector3 bom_speed = transform.Tran
        bom_speed += Vector3.up * 5;
        bom.GetComponent<Rigidbody>().velo
        bom.GetComponent<Rigidbody>().angu

        used_bom = true;
        ui.changeText_Bom(used_bom);
        StartCoroutine("bomCharge");
    }
    break;
```

- bomChargeコルーチンに1行

```
//■■■bomChargeコルーチン■■■
IEnumerator bomCharge()
{
    yield return new WaitForSeconds(3.0f);
    used_bom = false;
    ui.changeText_Bom(used_bom);
}
```

- 武器変更関数に1行追加

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

- 以上で終了

- 武器変更時に表示するテキストも変更
- 手榴弾を使うとused_bomがtrueになり、これをUIをいじるスクリプトに送ることでテキスト表示を“装填中”に変更する

- ゲーム実行
- テキスト表示が正しいか確かめる

4. 敵を用意する

- FPS的なゲームなんで敵を用意します
- 1から作る力は無いッ！
- アセットストアへ
- 「Zombie」で検索するとでてくる↓をインポート

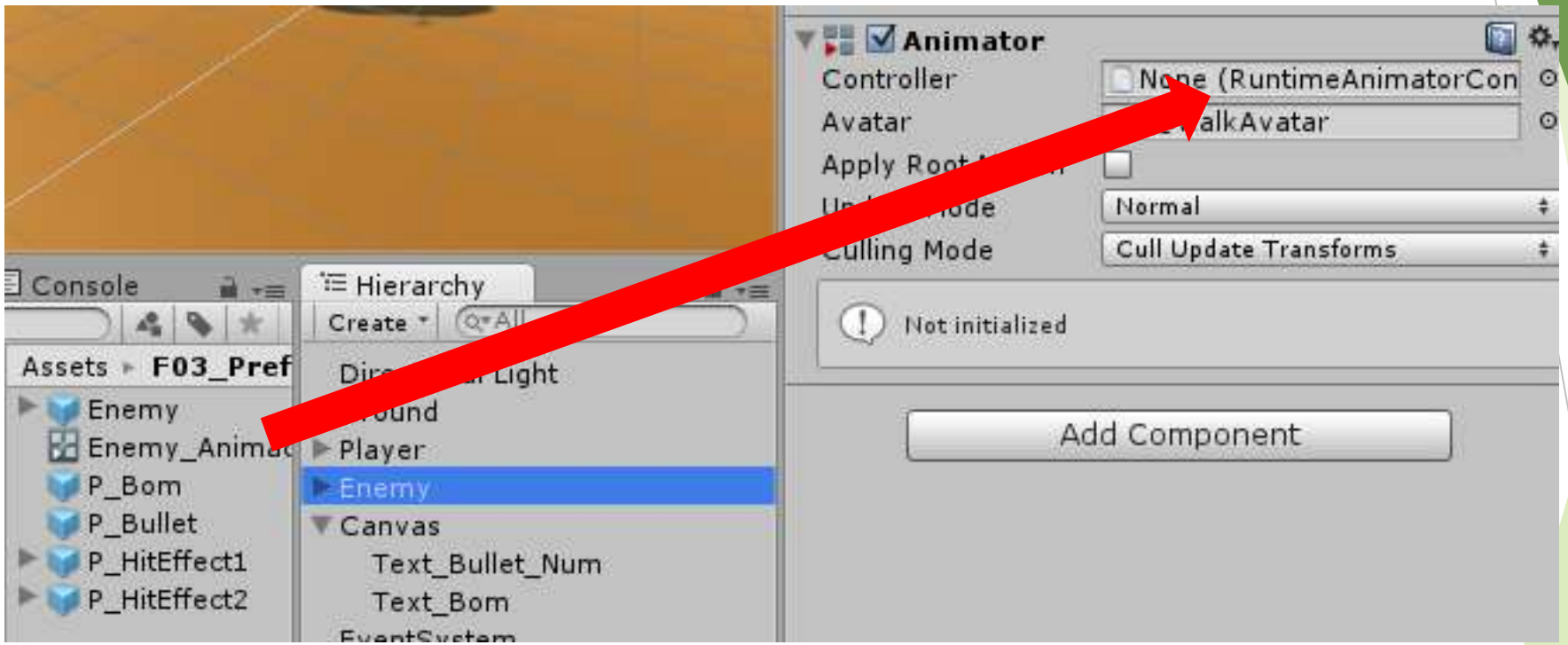
- 的として用意した棒たち（Enemy～Enemy(10)とかまで？）は削除

- Assetsに追加された「Zombie」→「Model」→「z@walk」を
  ヒエラルキにドラッグ&ドロップ
  名前を「Enemy」に変更してヒエラルキから「F03_Prefab」にドラッグ&ドロップ

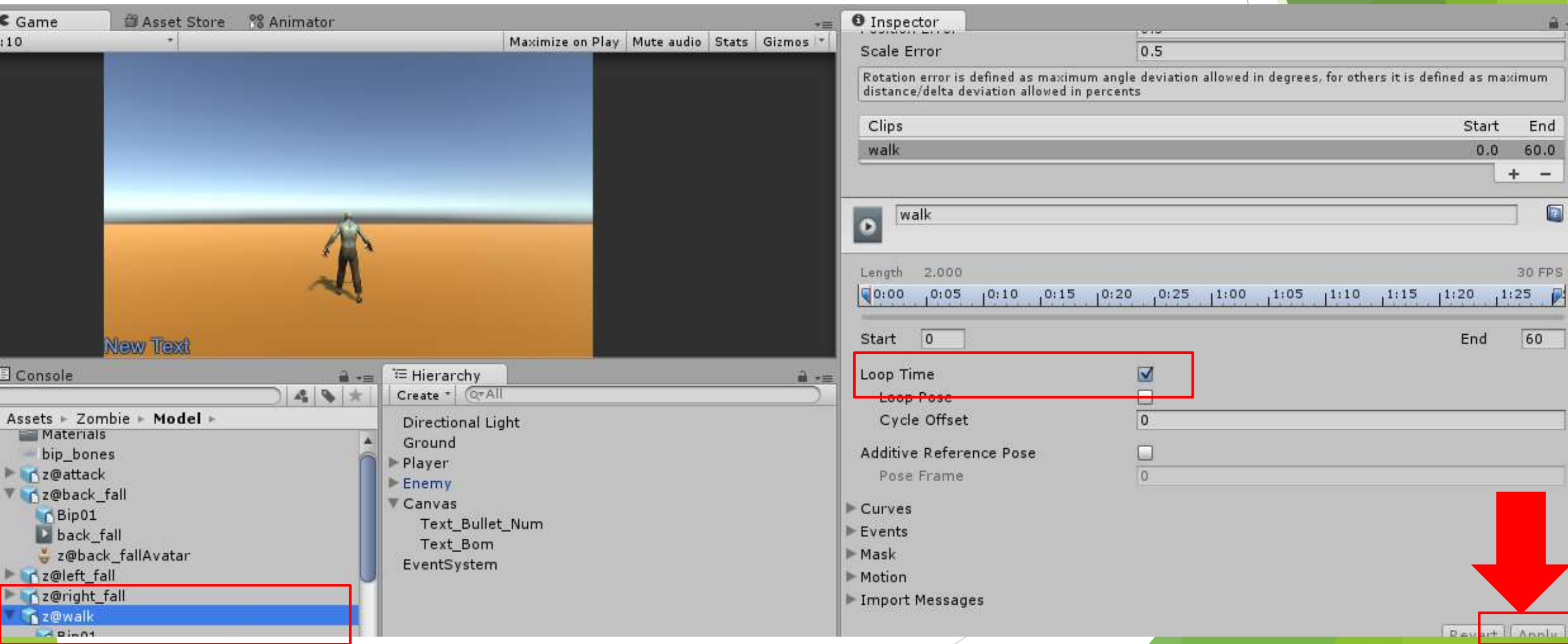
- ゾンビに歩くモーションを付ける
- この「Zombie」、歩くアニメーションと骨格を持っているッ！
- ちょろっと設定するだけ

- 「F03_Prefab」で右クリック「Create」→「Animator Controller」
- 名前は「Enemy_Animator」

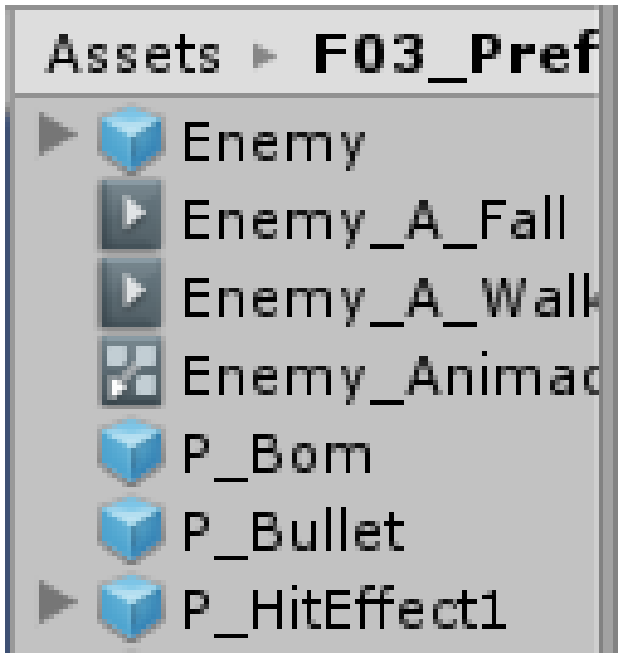
- ヒエラルキの「Enemy」のインスペクタ、Animator内の「Controller」に「Enemy_Animator」をドラッグ&ドロップ



- 「z@walk」のインスペクタ、Loop Timeにチェックを入れる
- 右下の「Apply」を押す



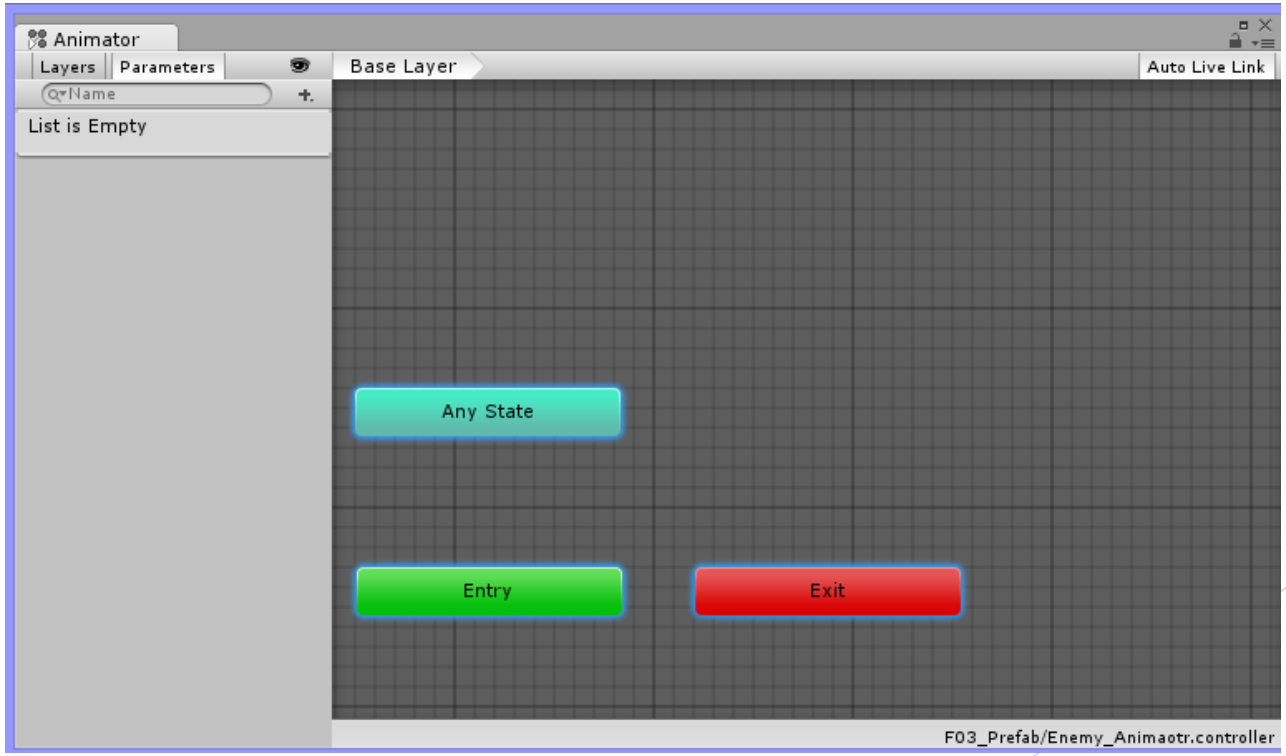
- 「Zombie」→「Model」→「z@back_fall」の「back_fall」をCtrl + Dでコピー
- 同様に「z@walk」の「walk」をコピー
- それぞれ名前を「Enemy_A_Fall」「Enemy_A_Walk」とし、「F03_Prefab」にドラッグ&ドロップ



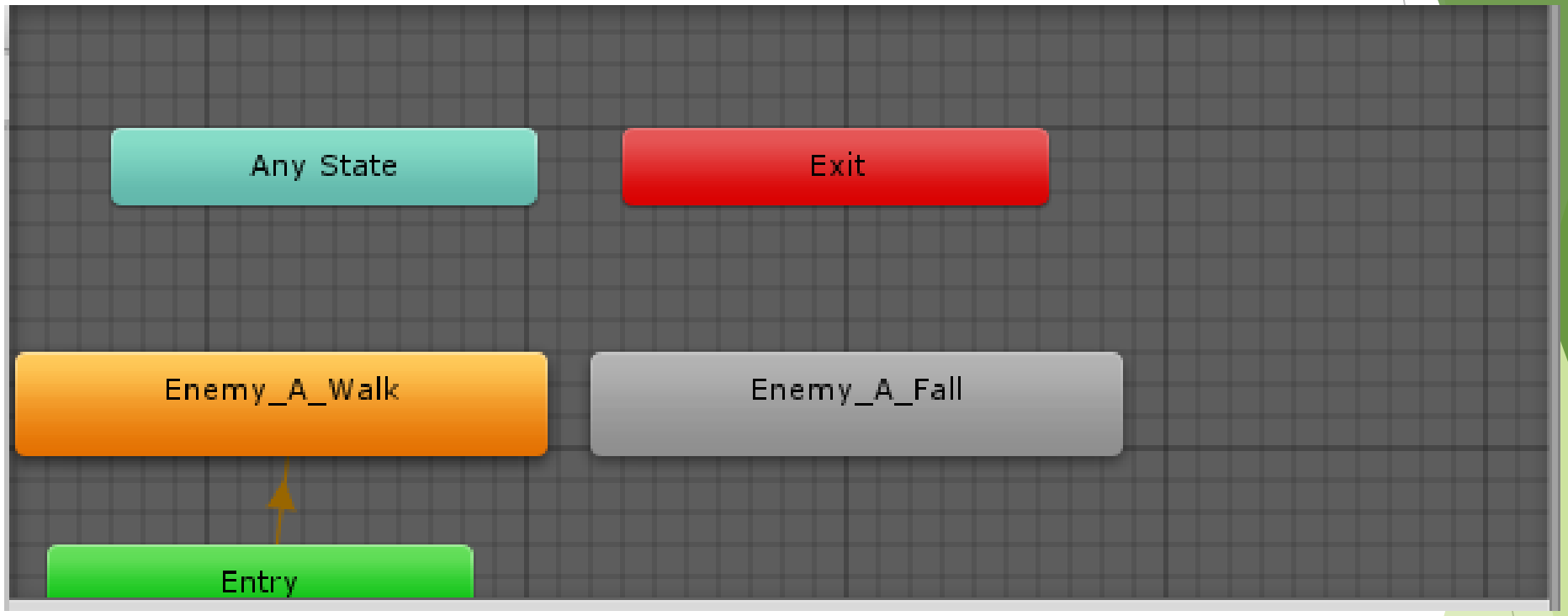
- 「Enemy_Animator」をダブルクリックしてAnimatorウィンドウを開く
- 上部メニューの「Window」→「Animator」でも開ける

- Animatorウィンドウをドラッグしてunityの画面外で離すと別ウィンドウになる

- 別ウィンドウになったAnimatorウィンドウを大きくしてAny Statesというやつを探す



- Animatorウィンドウに「Enemy_A_Walk」をドラッグ&ドロップ
- その後「Enemy_A_Fall」をドラッグ&ドロップ

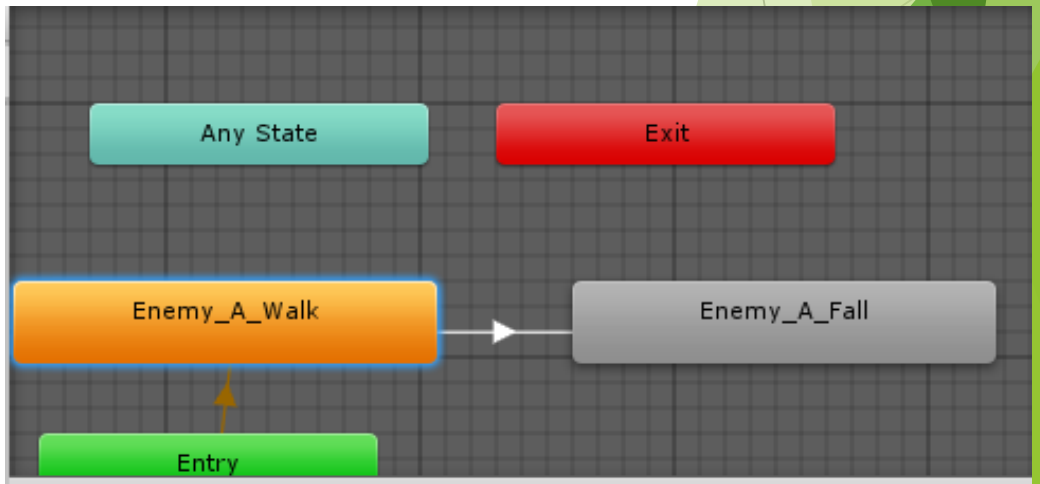


- 「Enemy_A_Walk」がオレンジの状態でゲーム実行
- ゾンビがその場で動く事を確認
- 見えない場合はヒエラルキの「Enemy」のpositionのzを5とかにする

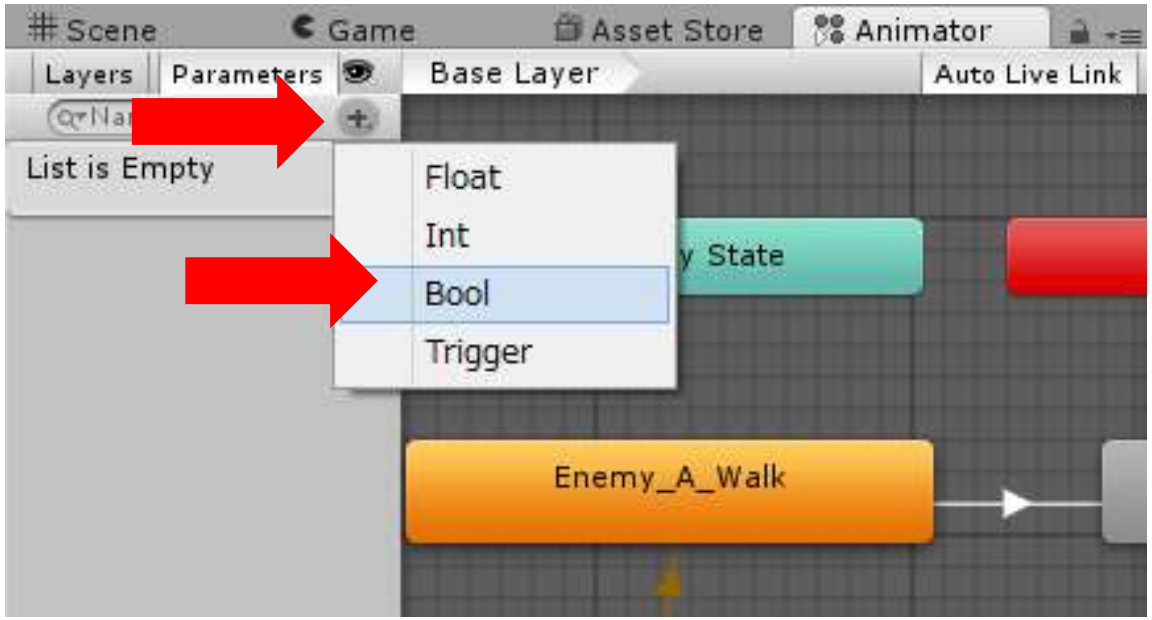
- 次にAnimatorウィンドウで「Enemy_A_Fall」を右クリック
  「Set As Layer Default State」を選択
- ゲーム実行
- ゾンビが倒れることを確認

- 確認できたら「Enemy_A_Walk」を右クリック
  「Set As Layer Default State」を選択

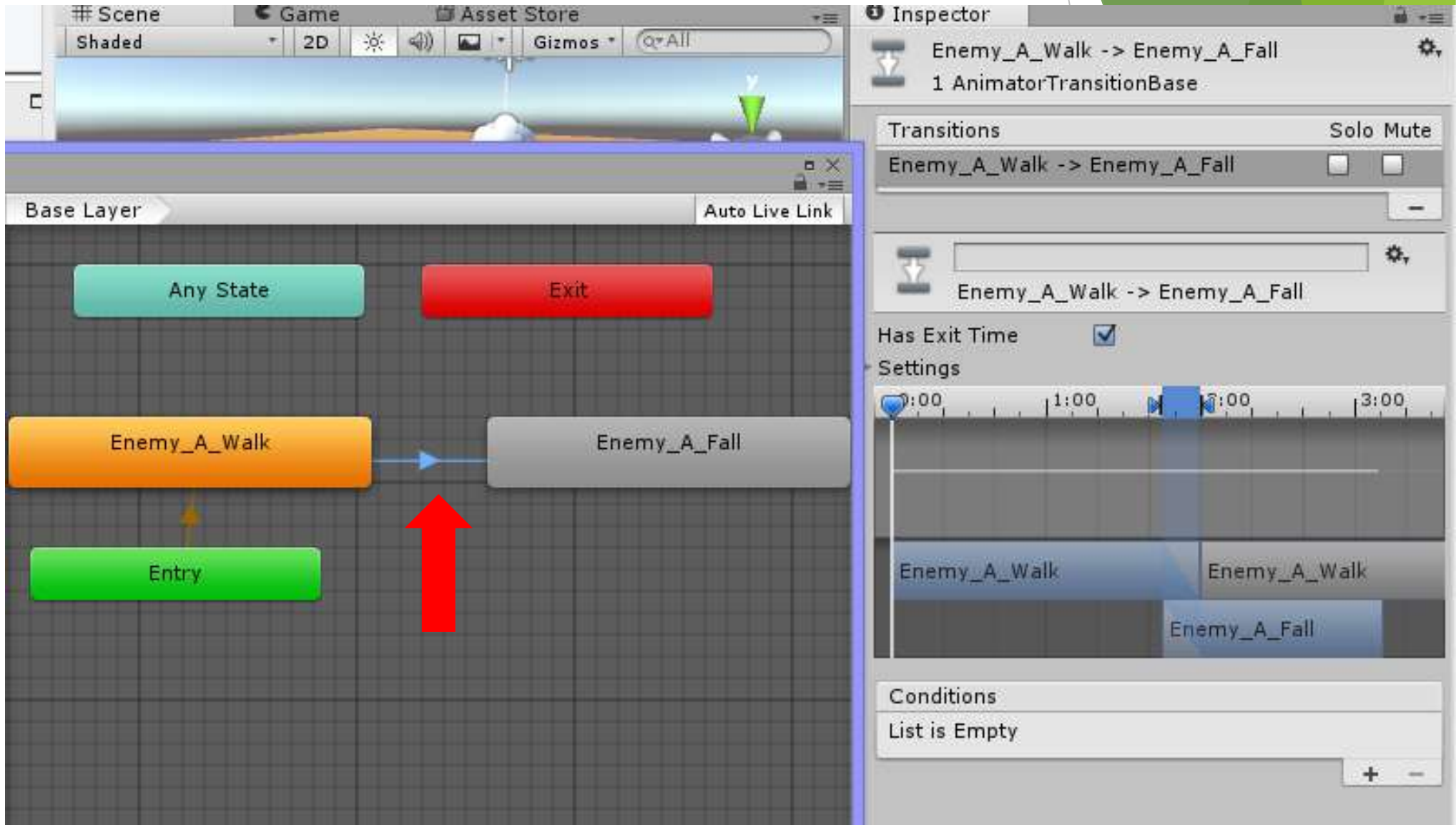
- 「Enemy_A_Walk」を右クリック「Make Transition」を選択
- そのまま「Enemy_A_Fall」を選択すると矢印が出る



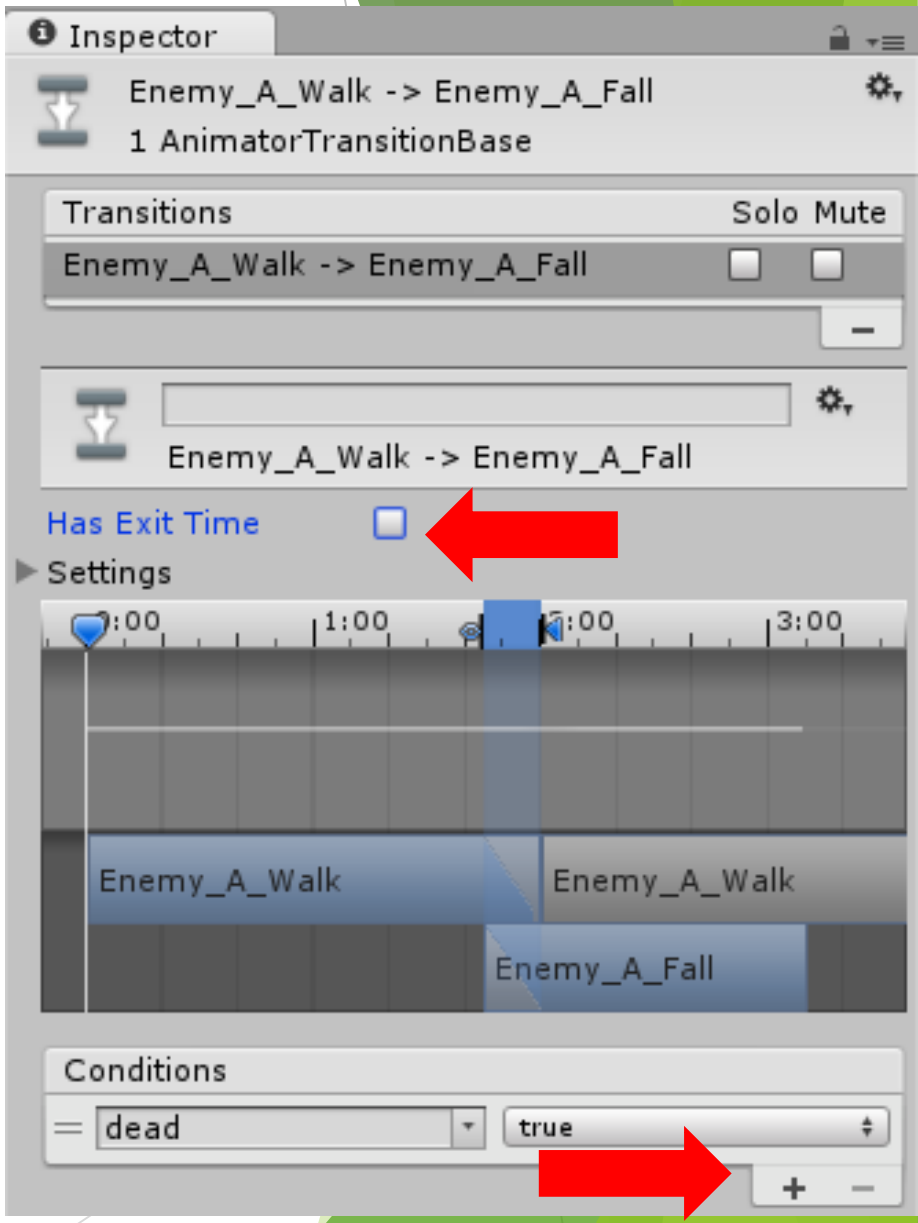
- Animatorウィンドウの左上「Parameter」の右下「+」からBoolを選択
- 名前は「dead」



- 「Enemy_A_Walk」から「Enemy_A_Fall」への矢印をクリック
- インスペクタが出る



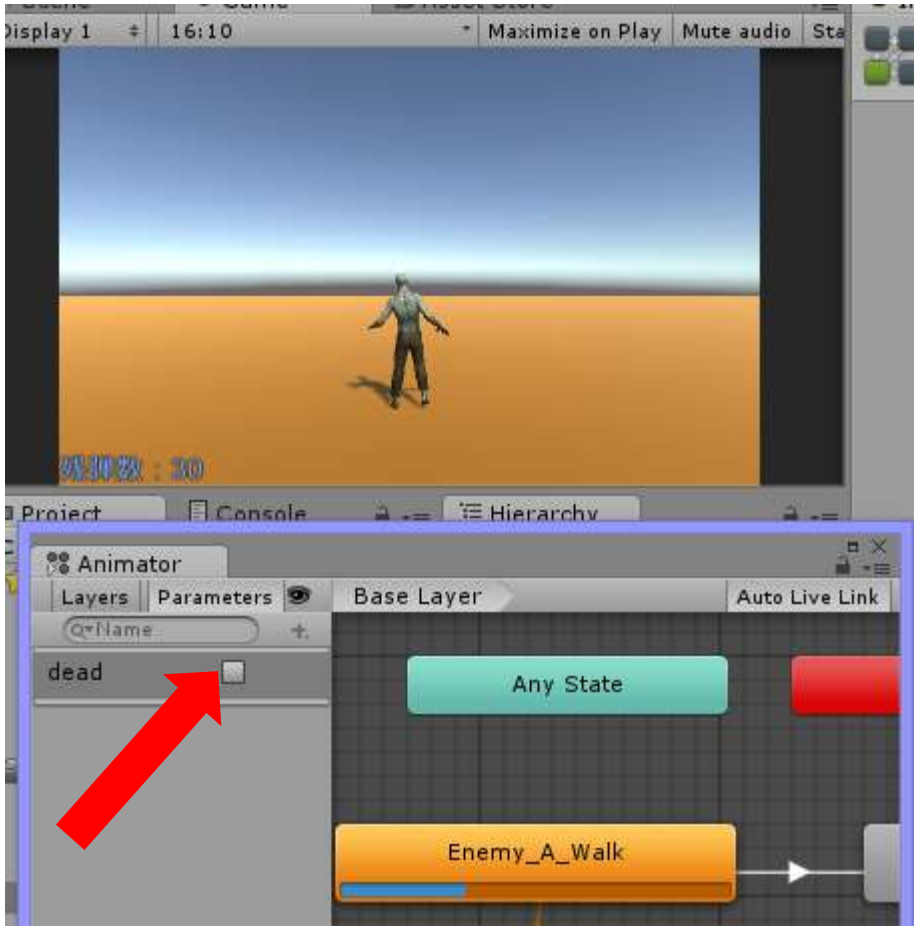
- 「Has Exit Time」のチェックを外す
- 下の「Conditions」の「+」をクリック
  「dead」が追加される

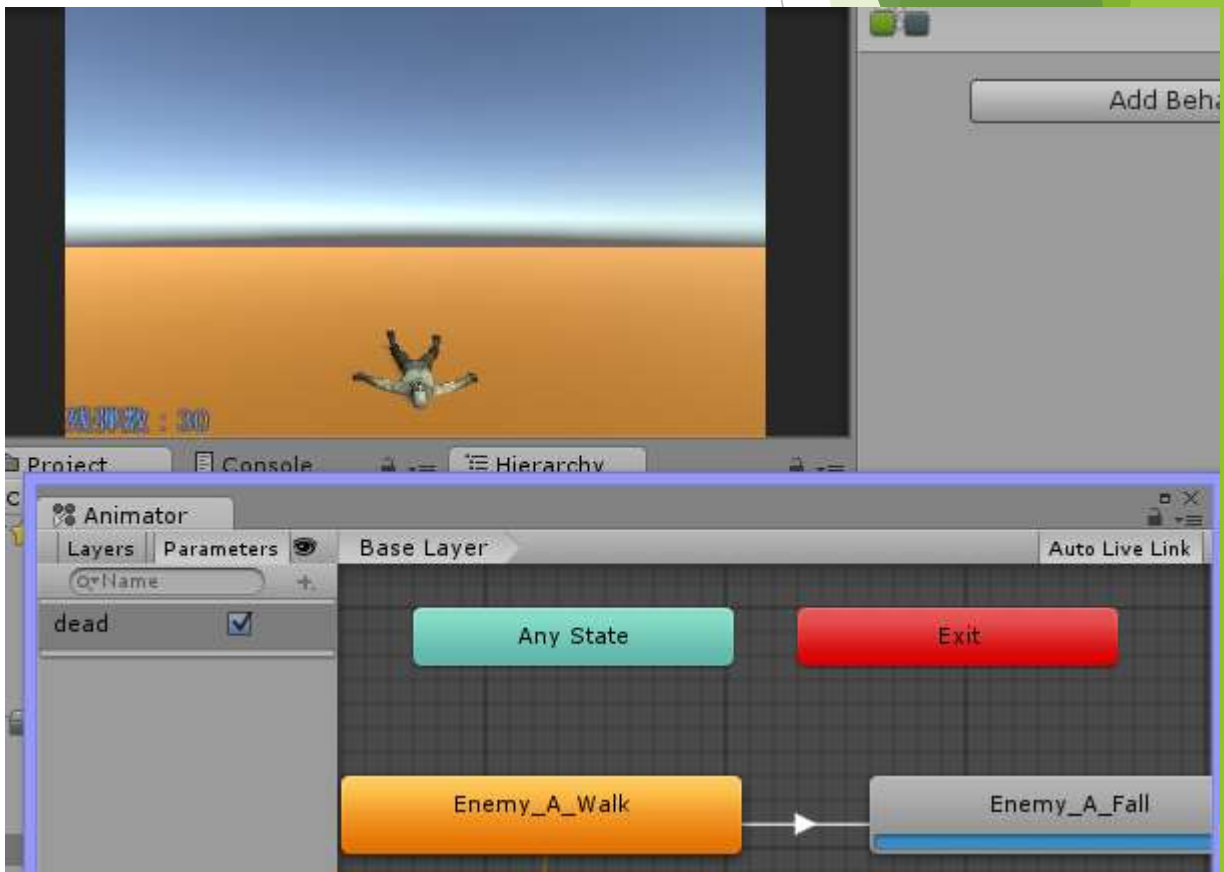


- Bool型のdeadという変数で、ゾンビの生死を判定
- 矢印に対して「dead」が「true」のときという条件を付けた
  →「dead」がtrueのときに矢印の遷移を行う

- 通常は歩くモーションだが
  「dead」がtrue、つまり死んだとき倒れるモーションに遷移する

- ゲーム実行
- まずは歩くモーションかどうかを確認
- Animatorウィンドウを開いてdeadにチェックを入れる
- 入れるのと同時にゾンビが倒れればおっけー

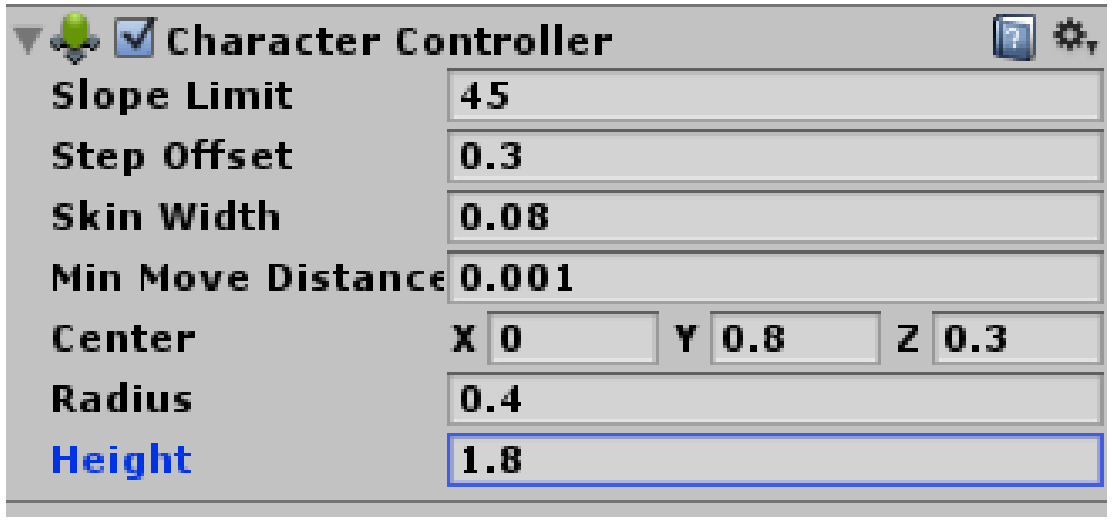




- deadの値をスクリプトでいじる
- 「F01_Script」で「Create」→「C# Script」
- 名前は「C06_Enemy」
- ヒエラルキの「Enemy」に追加
- ヒエラルキの「Enemy」のタグを「Enemy」にする
- ヒエラルキの「Enemy」の「Apply」を押してPrefabに情報を更新する

```
using UnityEngine;
using System.Collections;

public class C06_Enemy : MonoBehaviour {
    private Animator animator;

    void Start () {
        animator = GetComponent<Animator>();
    }

    public void damage()
    {
        animator.SetBool("dead", true);
    }
}
```

- このゾンビはコリダーが無いので弾が当たらない
- ヒエラルキの「Enemy」に
  「Add Component」→「Physics」→「Character Controller」
- 値を下のように
- （あたり判定の大きさなんで自由で）



- あとは弾が当たったときに「C06_Enemy」のdamage関数を呼び出すだけ
- 「C02_Bullet」を開く
- OnCollisionEnterに1行追加+1行変更

```
//■■■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, transf
    if (other.gameObject.tag == "Enemy") //衝突がEnemyなら
    {
        other.gameObject.GetComponent<C06_Enemy>().damage();
        Destroy(other.gameObject, 2.0f);
    }
    Destroy(effect, 0.2f); //エフェクトを破壊
}
```

- damage関数を呼んでゾンビが倒れる→2秒後に消滅
- ゲーム実行

- ついでに「C04_Bom」を開いて手榴弾の攻撃でも倒せるようにする

```
// ■■■ボムによる攻撃処理■■■
private void bomAttack()
{
    Collider[] targets = Physics.OverlapSphere(transform.positi
    foreach (Collider obj in targets)
    {
        if (obj.tag == "Enemy")
        {
            obj.gameObject.GetComponent<C06_Enemy>().damage();
            Destroy(obj.gameObject, 2.0f);
        }
    }
}
```

- あとゾンビが小さい気がするのでヒエラルキの「Enemy」のScale-yを1.3に
- ヒエラルキの「Enemy」の「Apply」を押して情報を更新

5. 敵の移動と生成

- 「C06_Enemy」をいじってゾンビがプレイヤーにまっすぐ向かってくるようにする

```
public class C06_Enemy : MonoBehaviour {
    private Animator animator;              // Animator用の変数
    private CharacterController charaCon;   // CharacterController用の変数
    private GameObject player;              // プレイヤーオブジェクト格納
    private Vector3 move = Vector3.zero;    // 移動用変数
    private bool isDead = false;            // 死亡判定
    private const float GRAVITY = 9.8f;     // 重力
    private float rotationSpeed = 180.0f;   // 回転速度

    void Start () {
        animator = GetComponent<Animator>();
        charaCon = GetComponent<CharacterController>();
        player = GameObject.Find("Player") as GameObject;
    }

    void Update()
    {
        if (isDead)
        {
            return;
        }
        enemyMove();
    }
```

- 続き

```
// ■■■移動■■■
private void enemyMove()
{
    // ▼▼▼移動量の取得▼▼▼
    float y = move.y;
    move = (player.transform.position - transform.position).normalized * 1.5f;

    move.y = 0.0f;

    // ▼▼▼向き変更▼▼▼
    Quaternion q = Quaternion.LookRotation(move);
    transform.rotation = Quaternion.RotateTowards(transform.rotation, q, rotationSpeed * Time.deltaTime);
    move.y = y;
    move.y -= GRAVITY * Time.deltaTime;

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

- 移動処理に関してはプレイヤーのものとほぼ同じです

- ゲーム実行
- 前にいるゾンビがこっちに向かってくる

- 次はゾンビの生成
- 「F01_Script」で「Create」→「C# Script」
- 名前は「C07_Spawn」
- 中身は次ページ

```
public class C07_Spawn : MonoBehaviour {
    public GameObject enemy;        // 敵を格納する変数
    private GameObject en_folder;   // 敵を格納するフォルダー
    private int max_enemy = 10;     // 敵の最大数
    private GameObject player;

    void Start () {
        en_folder = new GameObject("en_folder");
        player = GameObject.Find("Player");
        StartCoroutine("spawn");
    }


    IEnumerator spawn()
    {
        while (true)
        {
            yield return new WaitForSeconds(5.0f);

            if (en_folder.transform.childCount <= max_enemy)
            {
                float rndX = Random.Range(-24, 25);
                float rndZ = Random.Range(-24, 25);
                Vector3 pos = new Vector3(rndX, 0, rndZ);
                if (Vector3.Distance(player.transform.position, pos) > 5)
                {
                    GameObject obj = GameObject.Instantiate(enemy, pos, Quaternion.identity) as GameObject;
                    obj.transform.parent = en_folder.transform;
                }

            }
        }
    }
}
```

- 敵を入れるフォルダを作成
- フォルダ内の敵が10体以下のとき生成を行う
- x,z座標を（-24~24）でそれぞれランダムに取得
  （x,z座標で見て　中心（0, 0）に50×50の「Ground」があるため）

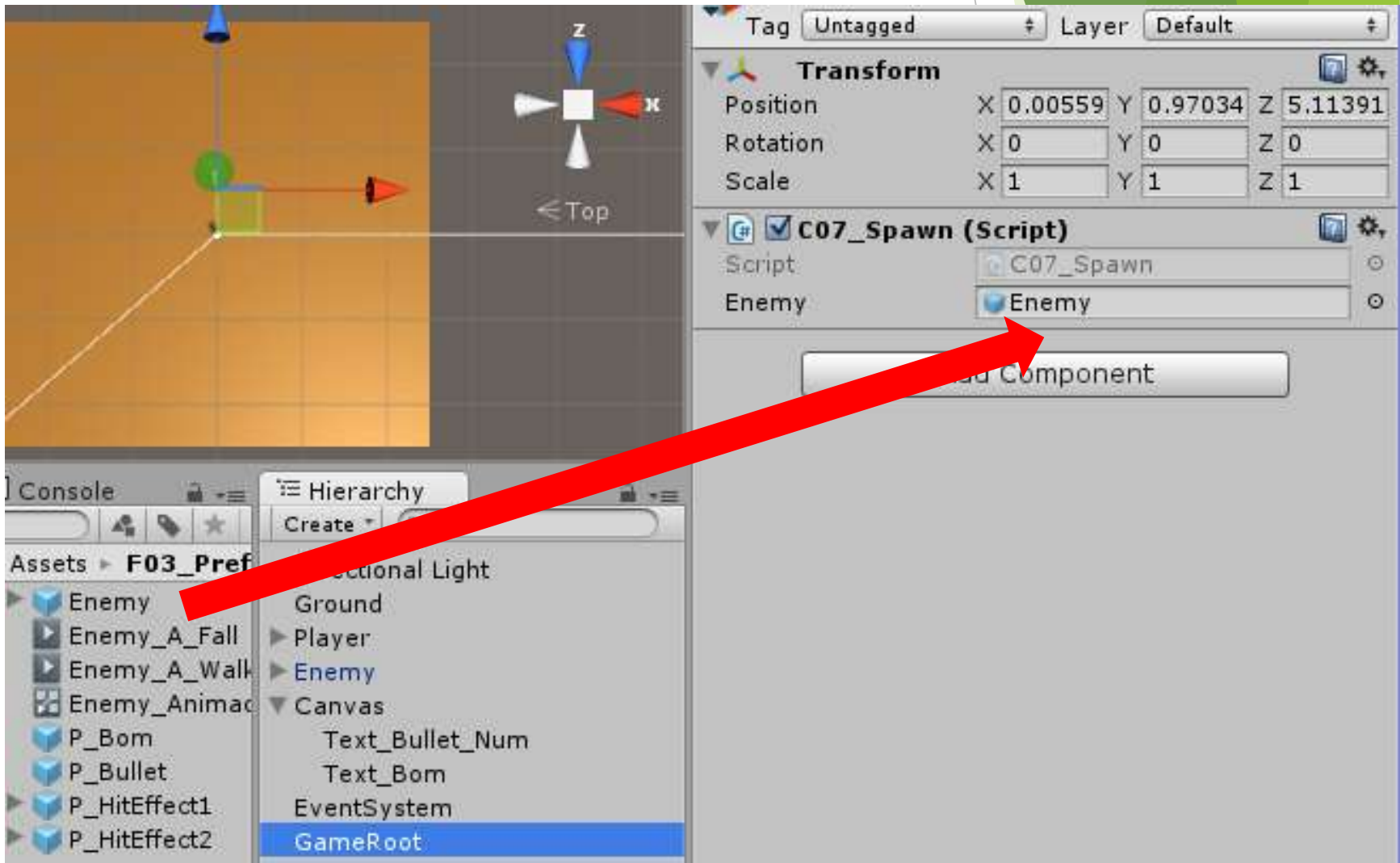
- 生成（するかもしれない）位置とプレイヤーの位置を比較
- 距離が5未満で生成を行う場合は生成をやめる（近すぎるため）

- 生成可能な場合は「en_folder」を親として敵を生成

- 「GameRoot」という空のオブジェクトを作るor既存のオブジェクトに「C07_Spawn」をドラッグ&ドロップ

- プレハブ化してある「Enemy」を「C07_Spawn」にドラッグ&ドロップ

- ここまで出来たらヒエラルキの「Enemy」は消して大丈夫



- ゲーム実行
- 5秒に1回敵がマップ上のどこかに生成されてこっちに向かってくる
- 攻撃を当てると倒れる

- 問題があって

  倒れた敵が消滅するまでの間に倒れたままこっちに向かってくる

  →倒れてもCharacter Controllerは生きているから

- 「C06_Enemy」のdamage関数に1行追加
- 死ぬ前にCharacter Controllerをなくす
- これで消滅までその場で倒れたまま

```
//■■■攻撃を受けると呼ばれる■■■
public void damage()
{
    charaCon.enabled = false;
    animator.SetBool("dead", true);
}
```

FPS作成はここまで

- FPSっぽい部分をいろいろ付け足しました
- 敵の攻撃とかHPとかやってないのでどんな風にやるかなーみたいなことを一度調べながら考えるといいかも
- コードもまとめられるところとか書き直したほうが良い部分とかあるのでそこらへんも考えてみて
- あとは自分で調べた方が身につきます
- 自分も細かいこととかなぜ動くかとかよくわかんないので

お疲れさまでした
ありがとうございました

この後も質問等は受け付けます