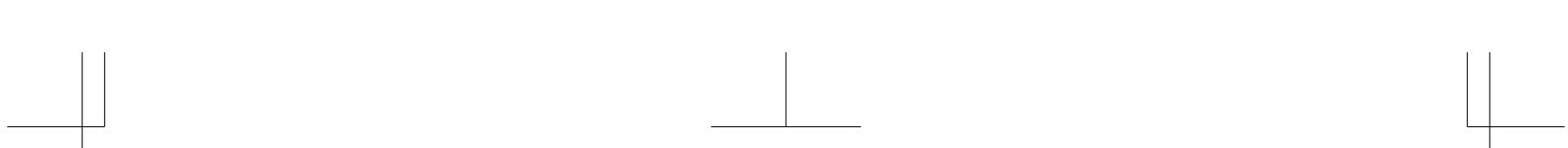


Unity と Wii リモコンで簡単アーケードゲーム制作

トミー (@Tomy_0331) 著

2019-01-26 版 発行



第1章

はじめに

1.1 この本について

「Unity を使えば簡単にゲームを作れる！」と聞いてダウンロードし、入門サイトを見ながら簡単なゲームはどうにかできたけど、その後どうすればいいの・・・と悩んでしまった方も多いのではないでしょうか。この本では「入門レベルよりちょこっとクオリティの高いゲームを作ってみたい」「弾転がしとかブロック崩しはもういい」「もっと Unity のいろんな機能を使いたい」というような Unity 初心者～中級者手前の人をターゲットにしています。

また、タイトル通り文化祭などで“映える”アーケードゲームを作ることをメインとしています。アーケードと言っても、Wii リモコンと自作ペダルのみの単純なものですが、シンプルかつ直感的な操作性は子どもや非ゲーマーでも楽しめると思います。

なお、この本では Unity 画面の見方や操作など基本的な部分や、機能の説明は省略することが多いです。プログラミングの C#における基本文法も説明を省かせていただくので、疑問が浮かんだときや詰まった時は適宜調べるか、一旦入門サイトや書籍でミニゲームを作ってみることをおすすめします。

1.2 開発環境

Windows10

一般的な OS. Unity は Mac 版でも操作は基本的に同じですが、今回使う Wii リモコンやペダルの Mac での入力方法は対象外とさせてもらうのでご了承ください。

Unity2018.3.2f1

今回のメインで使うゲームエンジン。3D ゲームを作るのに最適で個人からゲーム会社まで広い場所で使われています。以前は Unity4, Unity5 のような言い方でしたが、201x というバージョンナンバリングに変わりました。

blender

フリーの 3D モデリングソフト。造形からテクスチャのマッピング、アニメーション作成までできます。3D ビューの操作が独特で混乱しがち。

Visual Studio 2017

Unity でスクリプトを書くときに使います。Unity 環境の補完もしてくれます。

第 2 章

Unity でゲーム部分を作ろう

ゾンビを射撃する FPS ゲームを今回作ります。Wii リモコンで撃てるようにならたいのですが、Unity で作るときは Windows のマウスのクリックで入力できれば OK です。

この章では制作の順を追って解説していきます。

2.1 プロジェクトを作成する

Unity を起動し「New」から新規に作るプロジェクトの設定をします。Project name は「shoot」など任意の名前、Template は「3D」、他パスの設定などして Create project でプロジェクトを作成します。

2.2 地面を作る

ゲームの舞台となる地形を作ります。Unity には山や植物をペイントツールのように作る「Terrain」という機能があります。

Standard Assets のインポート

「Terrain」を使う前に、地面に適用するテクスチャや草木のデータを使うために公式の「Standard Assets」をインポートします。

Unity の Asset Store から「Standard Assets」で検索するとパッケージが出てくるのでダウンロードし、インポートします。インポート時に色々選択できますが、必要なテクスチャや草木データが入っている「Environment」のみでいいでしょう。

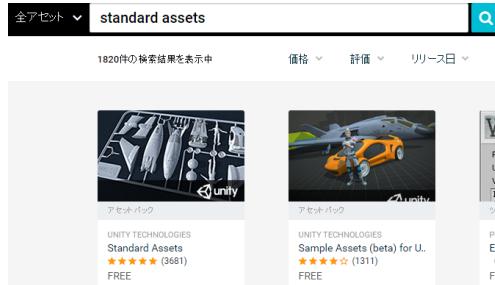


図 2.1: Standard Assets

Terrain の設置

「Hierarchy」の「Create」から「3D Object>Terrain」で作ります。「Inspector」の「Terrain」で設定ができ、「Paint Texture」を選択し「Edit Terrain Layers」から草のテクスチャ画像を指定しましょう。

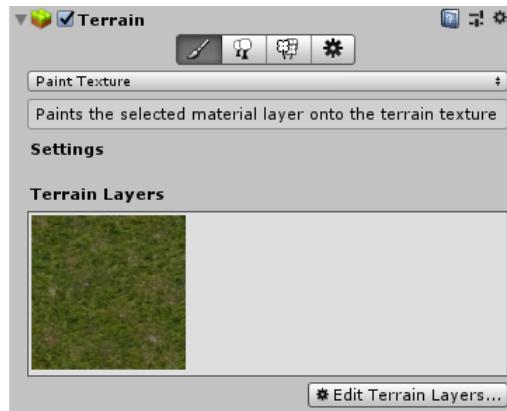


図 2.2: 草テクスチャの指定

2.3 敵 (ゾンビ) の用意

今回メインの作業である敵キャラの動作を作ります。

ゾンビが今回唯一登場するキャラクターです。人型の3Dモデルを扱う際、アニメーションやラグドール、移動AIなど初心者には難しいところがあるかもしれません、ここを作ることができると”ゲーム感”がとても出て感動も大きいはずです。

ゾンビのモデルをインポート

Asset Store で「zombie」と検索すると出てくる一番人気のアセットをインポートします。



図 2.3: ゾンビの素材

筆者はこの時インポートしたファイル名の一つが、大文字小文字の区別に関するエラーがなぜか出てきたので手動で修正しました。

ゾンビの死体を作る

恐ろしい見出し名ですが、ゾンビを撃った後に物理法則に従って倒れるようにラグドールを適用します。ラグドールとは、洋ゲーとかで敵が死ぬとぐったり人形のように崩れていくアレです。インポートしたアセットの「Prefabs」というフォルダにゾンビのプレハブがあるのでシーンに追加し、メニューの「GameObject>3D Object>Ragdoll...」から Create Ragdoll ウィンドウを開きます。

頭、腕、足など指定するためシーンのゾンビの階層を展開して、ドラッグアンドドロップで一つずつ体のパーツを当てはめていきます。

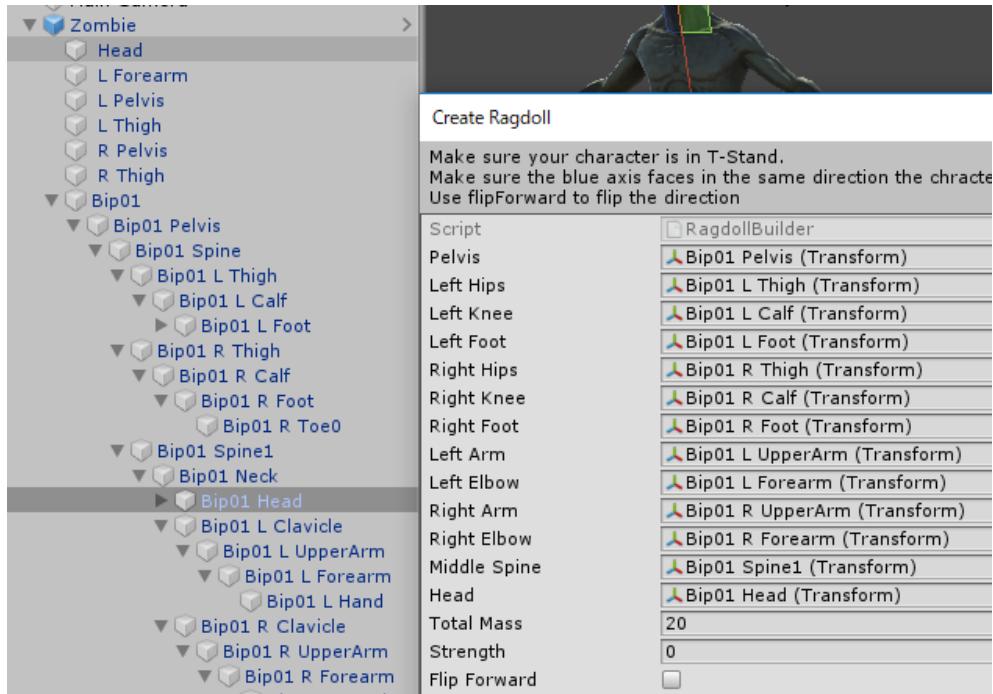


図 2.4: ラグドールの設定

タグを付ける

この後ゾンビの撃つスクリプトを書きますが、プログラムで識別するためのタグを付けています。インスペクタの「Tag > add tag」から「enemy」と名付けたタグを追加し、ゾンビのコライダがついているオブジェクト全てに適用しましょう。階層を shift キーで推しながら複数選択してから Tag を設定すると、一度に付けることができます。

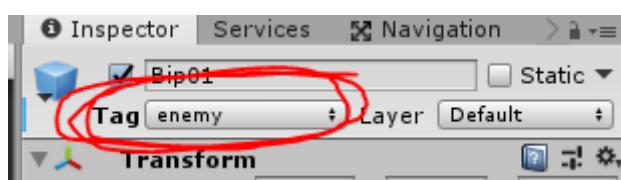


図 2.5: tag の設定

2.4 ゾンビを撃つスクリプトを書く

ここでゲームの動作部分になるプログラムを作ります。「ShotCam」という名前でスクリプトを新規作成し、^{*1}以下のコードを書きます。

スクリプトは Main Camera にアタッチします。

ShotCam.cs

```
public class ShotCam : MonoBehaviour {
    void Start() {
    }
    void Update() {
        GameObject clickObject = getClickObject();
        if (clickObject == null || clickObject.gameObject.tag != "enemy") {
            return;
        }

        //クリックしたのが敵なら
        Vector3 vec = clickObject.transform.position - this.transform.position;
        //射撃した部位に力を加える
        clickObject.GetComponent<Rigidbody>().velocity = vec.normalized * 30;
        //ゾンビ側のスクリプトの death() 呼び出し
        clickObject.transform.root.GetComponent<Zombie>().death();
    }
    // 左クリックしたオブジェクトを取得する関数
    public GameObject getClickObject() {
        GameObject clickObject = null;
        if (Input.GetMouseButtonDown(0)) { //左クリック
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit = new RaycastHit();
            if (Physics.SphereCast(ray, 0.1f, out hit)) {
                clickObject = hit.collider.gameObject;
            }
            Debug.Log(clickObject);
        }
        return clickObject;
    }
}
```

続いてゾンビのオブジェクト（一番上の階層）に Zombie というスクリプトを作成し、以下のクラスを追加しアタッチします。

Zombie.cs

^{*1} Unity では C# の他 JavaScript, Boo といったプログラミング言語を使えましたが、廃止となっています。

```
public void death() {
    GetComponent<Animator>().enabled = false; //アニメーション無効
    Invoke("destroyObject", 5f); //5秒後に消滅させる
}
void destroyObject() {
    Destroy(gameObject); //オブジェクトを消す
}
```

プログラム解説

まずは ShotCam の説明をします。

getClickObject 関数は、左クリックされると画面上に Ray と呼ばれるレーザーのような線を出し、衝突したオブジェクトを取得し return で返しています。毎フレーム呼び出される Update 関数では、getClickObject 関数でクリックしたオブジェクトを検知し続け、if 文にて enemy タグが付いているかを判別します。

変数 vec は撃った部位に力を加えて吹き飛ぶ演出をするためのベクトルを保持しています。撃った部位の座標からゲーム画面を映すカメラの座標を減算することで、ベクトルが取得できます。そして normalized で単位ベクトル化し、Rigidbody コンポーネントで力を加えています。「*15」は力の大きさです。試しに大きくして遊んだり自分好みの値にしてみるのもいいでしょう。

enemy タグはゾンビの胴体、手足、頭につけているので、clickObject.transform.root でいざれからも一番上の階層のオブジェクトから、Zombie.cs の関数 death を呼び出せます。

関数 death が呼び出されるとアニメーションを無効にして 5 秒後に消す処理を行います。アニメーションが無効になると魂が抜けたようにふっと体が崩れ (=ラグドールにより倒れ) ます。

シーンのカメラに見える位置にゾンビを設置し、Unity の三角ボタンで実行してみましょう。クリックするとゾンビが吹き飛ぶと OK です。

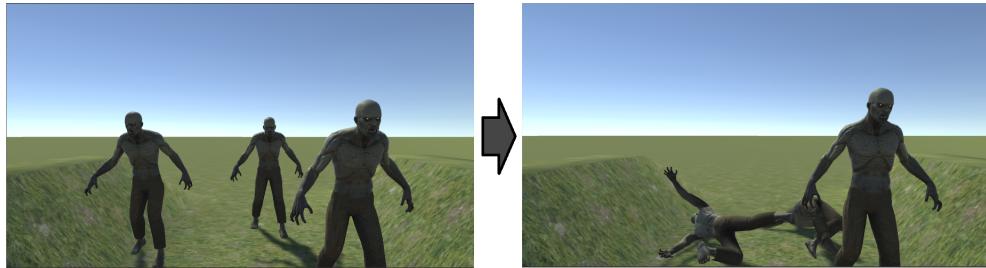


図 2.6: 実行結果(クリックすると倒れる)

2.5 ゾンビを歩かせる

ゾンビをプレイヤーまで歩かせるために、経路に沿って移動していく Agent とアニメーションとして体が動く Animator の二つの機能を使います。==== 経路探索 AI の実装 Unity には嬉しいことに経路探索 AI が自動で入っています。まず、ゾンビに「Nav Mesh Agent」のコンポーネントを追加し図のように設定します。

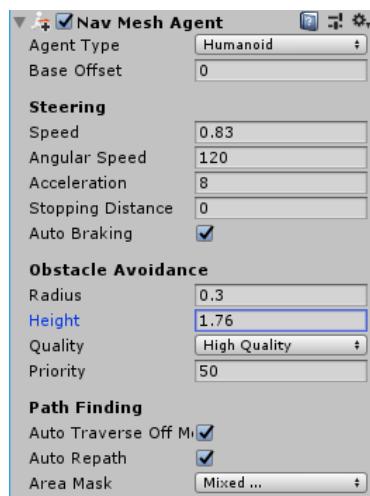


図 2.7: Nav Mesh Agent の設定

次に「Terrain」に経路探索のために必要な作業をします。「Terrain」を選択するとインスペクタの隣に「Navigation」というタブが出てきます。選択し「Bake」からパラメータを調整して Bake することで、ゾンビが歩ける場所が青く表示されます。

なお、試しに障害物として画像では Cube を置いていますが、インスペクタから「static」

にチェックを入れるのを忘れないようにしてください。

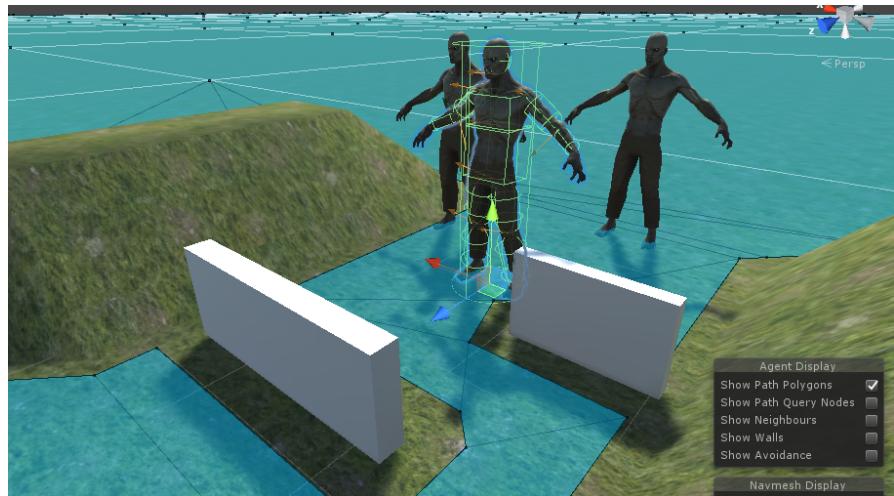


図 2.8: 経路となる領域を Bake した様子

アニメーション遷移の実装

移動中には歩くアニメーション、止まるときには待機のアニメーションを適用するため Animator の設定をします。アセットに walk, idle, attack のアニメーションが含まれているのでノードと遷移を設定します。矢印はノードを右クリックし、「Make Transition」を選択します。矢印を選択したときにインスペクタに表示される Conditions で「state」と名付けた int 型の遷移条件変数を追加し、walk への矢印には Equals 0, idle へは Equals 1 を割り当てます。遷移条件変数は Animator ウィンドウ左の Parameters で「+」マークを押すと作成できます。

また、矢印を選択したときにインスペクタに表示される「Has Exit Time」のチェックは外してください。これは state の変更があった時にアニメーションを中断してすぐに切り替えるためです。チェックが付いたままですると、例えばゾンビが立ち止まったのに、アニメーションでは一瞬歩く動作のままで滑るような動きをしてしまいます。

攻撃アニメーションのノード「attack」に関しては、後で実装するので今は矢印を付けなくていいです。

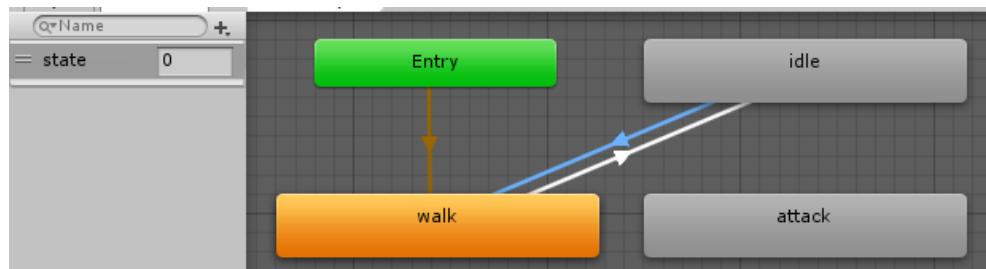


図 2.9: Animator の設定

スクリプトを書き換える

経路探索のために AI 機能をインポートします

Zombie.cs

```
using UnityEngine.AI;
```

そして以下のように書き換えていきます

Zombie.cs

```
public class Zombie : MonoBehaviour {
    private new GameObject camera;
    private NavMeshAgent agent;
    private bool stop;
    private enum state { walk, idle } //アニメーションの状態
    private Animator animator;
    void Start() {
        camera = GameObject.Find("Main Camera").gameObject;
        agent = GetComponent<NavMeshAgent>();
        agent.SetDestination(camera.transform.position); //目標座標を設定
        stop = false;
        animator = GetComponent<Animator>();
        SetKinematic(true); //物理演算を無効にする
    }
    void Update() {
        //ゾンビが目標点まで 2m 近づいたら立ち止まる
        if (stop ||
            Vector3.Distance(camera.transform.position, this.transform.position) >= 2f)
        {
            return;
        }
    }
}
```

```
animator.SetInteger("state", (int)state.idle);
Vector3 p = camera.transform.position;
p.y = this.transform.position.y;
transform.LookAt(p);
agent.isStopped = stop = true;
}
//死ぬ処理
public void death() {
    GetComponent<Animator>().enabled = false; //アニメーション無効
    Invoke("destroyObject", 5f); //5秒後に消滅させる
    SetKinematic(false); //物理演算を付ける
    agent.enabled = false;
}
void destroyObject() {
    Destroy(gameObject); //オブジェクトを消す
}

public void SetKinematic(bool newValue) {
    Component[] components = GetComponentsInChildren(typeof(Rigidbody));
    foreach (Component c in components) {
        (c as Rigidbody).isKinematic = newValue;
    }
}
```

プログラム解説

Start 関数では、ゾンビが目的地とするカメラの取得と座標設定、Animator の取得をしています。

Update 関数では、ゾンビがカメラまで近づいたら動作する処理があり、state を待機(idle) にアニメーションを切り替えカメラの方向を向き、移動を停止しています。

そして death 関数の変更と SetKinematic という関数の追加も行っています。これは、NavMesh Agent と Animator、ラグドールを併用したために必要となったプログラムです。公式ドキュメントでは、NavMesh Agent は Physics と合わせて使用するのを非推奨としています。キャラの動きを NavMesh Agent で行っているのにそれに物理トリガを加えたりアニメーション動作を加えると競合してしまうからです。そこで、NavMesh Agent で動かしている、すなわちゾンビが生きている(?) 状態では Rigidbody の Is Kinematic をオンにして物理演算を無効にし、撃たれて死ぬ状態では NavMesh Agent をオフにしつつ Is Kinematic をオンにすることでラグドールを動作させるやり方になっています。SetKinematic 関数はそのために子オブジェクトが持つ Rigidbody コンポーネント全ての物理演算を切り替えるものです。

これでプレビューを実行してみると、ゾンビがカメラに向かって歩きだし、近づくと立ち止まります。

2.6 ゾンビを攻撃させる

さて、現状ではゾンビがただプレイヤーにわらわら集まつてくるだけです。ゾンビのモッシュですね。ですがゾンビはライブのパリピではなく人間を喰らう存在ですので、攻撃するよう実装していきましょう。

animator で攻撃の遷移を設定する

attack に画像のように矢印を付けますが、attack への遷移条件は「attack」Trigger が呼び出されたときという設定になります。そして attack から idle, walk への遷移は「Has Exit Time」のチェックを付けるのみにします。

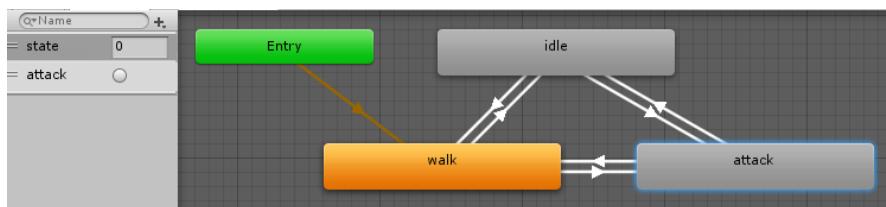


図 2.10: animator の設定



図 2.11: attack への遷移条件

カメラの揺れ演出の準備

ゾンビが攻撃するようになったのでプレイヤーもダメージを負う演出を加えます。ゾンビが攻撃するとカメラを揺らすのですが、一からコードを書くと面倒なので iTween という Asset を使おうと思います。これまでと同じように「iTween」と Asset Store で検索、インポートしたら準備は完了です。

プログラムを書き換える

攻撃動作を付けるために Zombie.cs を書き換えます。攻撃は指定秒ごとに行うようになります。以下のように時間を管理するメンバ変数を定義します。

Zombie.cs

```
private float timeOut;
private float timeElapsed;
```

そして Start 関数内に timeOut の初期化を追加。

Zombie.cs

```
timeOut = 3f;
```

Update 関数に以下のコードを追加します

Zombie.cs

```
timeElapsed += Time.deltaTime;
if (timeElapsed < timeOut || !stop) {
    return;
}

animator.SetTrigger("attack");
timeElapsed = 0.0f;
Invoke("damage", 0.9f);
```

最後に、以下の関数を追加します。

Zombie.cs

```
void damage() {
    //ゾンビが死んでいたら無効
    if (!agent.enabled) {
        return;
    }

    iTween.ShakePosition(camera, iTween.Hash("x", 0.1f, "y", 0.1f, "time", 1f));
}
```

プログラム解説

Time.deltaTime には、最後のフレームからの経過時間 [ms] が格納されています。Update() が実行されるたびに経過時間を積み上げていき、指定した時間を超えたら望みの処理を実行するようにします。

ゾンビが攻撃をしたらアニメーションが開始されますが、開始から 0.9 秒後と指定してカメラを iTween で揺らしています。0.9 秒は攻撃アニメーションが開始されてから引っ搔かれてダメージを負うのに丁度いい時間になっています。

しかし、この仕様のままだと”攻撃アニメーションが開始”された後から 0.9 秒以内にゾンビを撃った場合、ゾンビが倒れたのにダメージ演出が呼び出されてしまいます。そこで、NavMesh Agent が有効かどうかで if 判定を行っています。フラグ変数を作ってもいいのですが、簡略化のためこのようにしました。

■ コラム: 便利な iTween
めっちゃ便利～～～～～！ position,rotate 色々

2.7 エフェクトを付ける

ゲームの見栄えを良くするためにエフェクトを使うことは効果的です。プレイヤーに”撃った感”を感じてもらうために着弾点に爆発のようなエフェクトを付けます。エフェクトは Unity のパーティクルシステムという機能で作成することも可能ですが、今回は Asset Store で無料の素材をお借りしましょう。

Asset Store で「War Fx」と検索すると爆発や炎、銃撃などが入ったアセットが見つかります。



図 2.12: War Fx

インポートしたら次に、ShotCam.cs を改造します。クラスのメンバ変数として

ShotCam.cs

```
public GameObject hitEffect;
```

と宣言し、getClickObject 関数内の if 文に以下のように一行追加します

ShotCam.cs

```
if (Physics.SphereCast(ray, 0.1f, out hit)) {
    clickObject = hit.collider.gameObject;
    // [追加] 着弾点エフェクト
    Instantiate(hitEffect, hit.point, Quaternion.identity);
}
```

そして hitEffect は public で宣言しているため、Main Camera のインスペクタから手動でアタッチする必要があります。先ほどインポートしたアセットの「JMO Assets>WarFX>_Effects>Explosions>WFX_Explosion Small」をドラッグ&ドロップでアタッチします。実行してみるとわかるのですが、エフェクトが大きすぎて不自然です。そこでエフェクトの Scale を x, y, z 全部 0.3 にしたら丁度よくなりました。エフェクトを変えてもプログラム自体は変える必要ないので他のアセットやサイズ、オリジナルエフェクトなど試してみるのもいいでしょう。

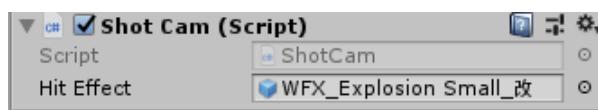


図 2.13: War Fx

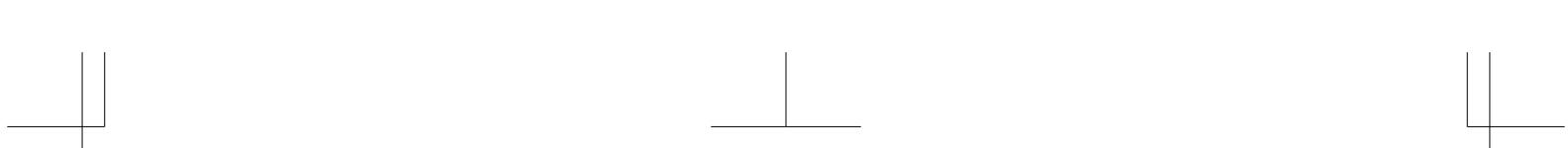
動作確認

さて、ラグドール、アニメーション、Navmesh Agent、エフェクトなど様々な実装をしましたが、いかがだったでしょうか。Null参照などのエラーが出る場合、コンソールをダブルクリックすることで原因となる処理のソースが分かることもあります。

正しく実行できている場合、図のようにゾンビが経路に従い歩き、カメラに近づくと攻撃、カメラが揺れます。ゾンビを撃つと火花のエフェクトとともにゾンビが吹き飛びます。



図 2.14: 実行結果



第3章

マップのモデリング

さて、一章ではゲームの基本システムを作成しました。しかし地平線やただの地形だけの背景だとどうしても寂しいですよね。

そこでゲームには不可欠なマップの作成ですが、今回はモデリングによるオリジナルマップ作成を簡単にですが紹介しようと思います。この本で扱うゲームはどんなマップでも基本遊べますので、この章はスキップしてアセットストアの無料マップを使うのも大丈夫です。

3.1 blenderによるモデリング

今回無料3Dモデリングソフト「blender」を扱います。少々（少々か？）操作に癖のあるソフトですがモデリング、テクスチャマッピング、ボーンやアニメーションの作成など一通りの機能を揃えています。日本語にも対応しており、設定で切り替えられます。

実際にある建物のモデリング

開発者の趣味でゲームを作っているとき、マップはどうすればいいでしょうか。ゾンビを倒すゲームということで王道に荒廃した架空の都市でもいいでしょう。今回は少し捻つて実際に存在する場所を舞台にしてみます。今回は最近流行りの某アニメをパロって佐賀駅をモデリングしちゃいます。ゾンビに溢れ荒れる佐賀駅、いいですね。インターネットミームで海底都市SAGAとかありますが、筆者のゲームでは屍都市SAGAです。何にでもなれるのが佐賀の魅力だと思います（？）。

なお、この章では佐賀駅を例としていますが、最寄り駅、学校、勤務先など任意の荒らしたい舞台にしちゃっていいです。むしろそうしましょう。

素材となる資料

実在する建物の場合おすすめはGoogleMAPの航空写真です。細かい部分はストリートビューで見ることもでき、3Dビュー対応の地域だと色々な角度から見えます。



図 3.1: 佐賀駅の 3D ビュー

航空写真は blender の「下絵」機能を使って寸法を再現できます。高さは感覚での調整になりますが、これだけで再現度が上がりリアルになります。

シンプルな建築物の 3D モデリングの基本は、立方体の「分割」と「押し出し」の繰り返しです。

blender は沢山の操作ができショートカットキーも同じように覚えきれないほどあります。しかし基本的に立方体の押し出しの組み合わせで作る程度ですと、最低限の操作さえ把握していれば作ることができます。

以下、筆者が良く使うショートカットキー一覧です。逆にこれ以外は使わなくてもほぼ作ることができます。

表 3.1: ショートカットキー

ショートカット	操作	説明
Tab	モード切替	オブジェクトモードと編集モードに切り替える
G	移動	選択している部分の座標を変更する
R	回転	選択しているオブジェクトを回転させる
S	拡大縮小	選択している部分を拡大縮小する
E	押し出し	選択した頂点・辺・面を押し出す
W	細分化	ポリゴンを分割する
Ctrl+R	ループカット	選択した辺に隣接している全ての面を繋げてカットする
Ctrl+B	面取り	角を丸める
X/Y/Z	軸選択	上記の編集中に押すことで各軸ごとに移動や押し出しができる

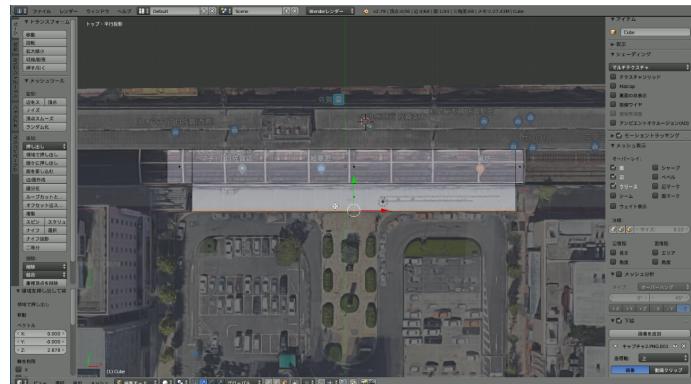


図 3.2: 下絵機能を使い上から見た形を作る様子

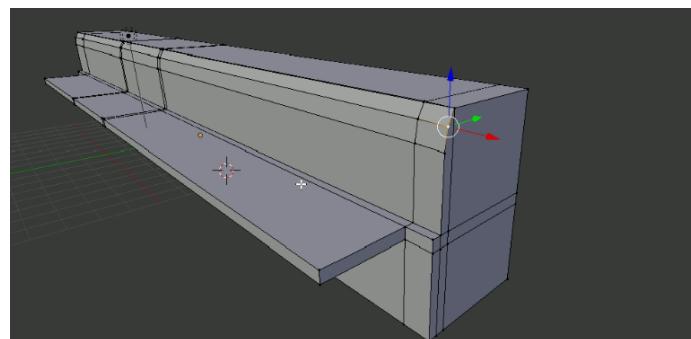


図 3.3: 横側に「押し出し」でいる様子



図 3.4: モデリングした佐賀駅南口

3.2 テクスチャのマッピング

モデリングをしたら建物の形はできますが単色でのつぱりしたままです。モデルの平面に画像を張り付けて質感を与えることをテクスチャマッピングといいます。

「UV 展開」をして画像を適用する方法がおすすめです。UV 展開はアジの開きの要領で辺に切り込みを入れていって展開し、一枚の画像（ループ可）を反映させるイメージです。

辺を選択し「シームを付ける」で赤色になった部分が切れ込みです。一通り終えたら「展開」して「UV/画像エディター」で画像を貼ります。

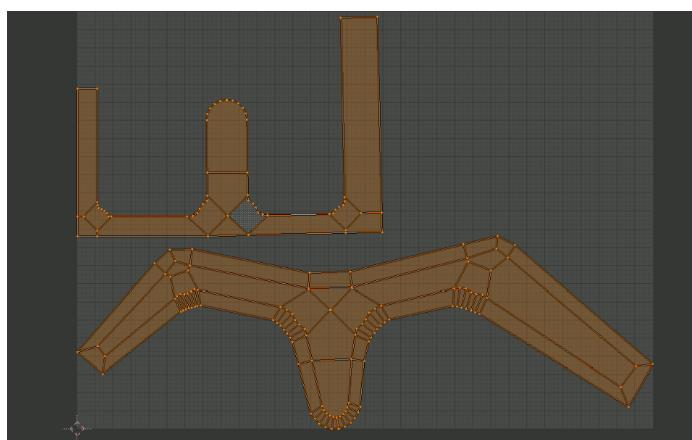


図 3.5: UV 展開

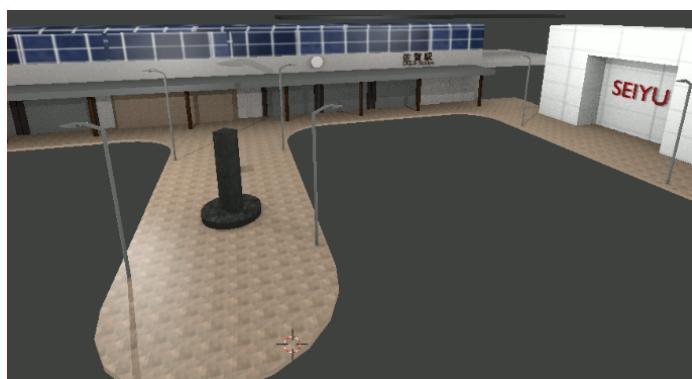


図 3.6: テクスチャを貼った様子

■コラム: テクスチャや3Dモデル素材

テクスチャや3Dモデルの無料素材が欲しい!と思った時、どうしますか? UnityのAssetStoreで探すのもいいですが、インターネットには数多くの素材サイトがあります。

3Dモデルなら「Free3D」や「TURBOSQUID」などがおすすめです。

テクスチャに関しては材質の英単語で検索すると多く出ます。その時「seamless」と入れるとパターン画像(上下左右ループできる)を絞り込みます。

例えば石畳のテクスチャが欲しいときは「stone paving texture seamless」といった感じです。素材は基本的に英語で探したほうがヒットは多くなります。

3.3 Unity でモデルを配置する

blenderでfbx形式にエクスポートしてからUnityに取り込みます。その時注意することはUV/画像エディターの他にもテクスチャを新規作成して画像の参照をさせることと、blender側でマテリアルとテクスチャ名を重複させないこと、Unityにインポートするとテクスチャが一度分解されてしまうので再度シーンビューで画像をD&Dして反映されることです。



図 3.7: Unity に設置した駅やフリー素材のモデル

モデリングした駅の他、Asset Storeでインポートした車やビルを配置しました。木はTerrainの機能で生やしています。

また駅をさらに荒らすため火事を演出する炎のエフェクト、煙、一帯を覆う霧を設置

しました。これらも Asset Store でインポートし、設置したい場所にプレハブを置くだけです。



図 3.8: 炎や煙のエフェクトを設置し完成した MAP

以下が今回マップ作成で Asset Store からお借りした素材になります。

- Simple Cars Pack
- Simple City pack plain
- Unity Particle Pack 5.x

第4章

ゲームを仕上げる

ラストスパートです。

4.1 ゾンビのスポナーをつくる

すぽぼーん！ w

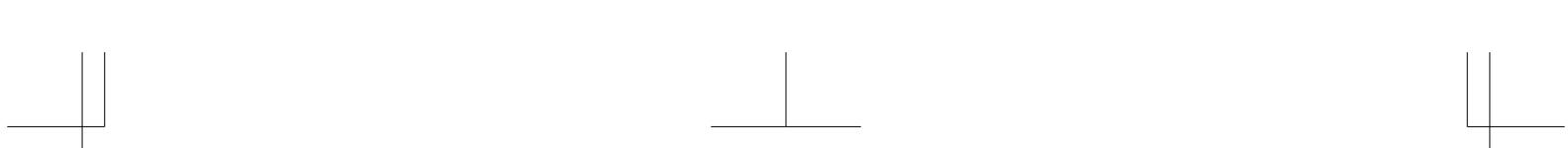
4.2 ライフとゲームオーバーを作る

UIとか

シーン遷移

4.3 ビルド

ここで詰まると萎えるよね。



第5章

Wii リモコンで遊べるようにする

ゲームが完成したらいよいよ”アーケード”ゲームにするべく Wii リモコンの入力を対応させます。

とはいっても、実は特殊なことをするわけではなく Wii リモコンを Windows マウスカーソルに変換するフリーソフト「TouchMote」を使うだけです。アーケードゲームとかっこよく言っていますが、実は簡単ですね。

Wii リモコンは Bluetooth でゲーム機で通信しており、Bluetooth のペアリングは WindowsPC でも可能です。

ちなみに、センサーバーと呼ばれるテレビの上下に設置するアレは赤外線の LED が入っているだけでリモコンがその光で向ける方向を認識しています。”センサー”と名前が付いてはいますが受信機能はありません。仕組みはシンプルなためセンサーバーを作成する人もいるそうです。

5.1 TouchMote のインストール

TouchMote は Windows 上のマウスカーソルやキーボード入力に Wii リモコンのポインター やボタンを割り当てることのできるフリーソフトです。

公式サイト^{*1}からダウンロードしインストールします。

5.2 Wii リモコンのペアリング

Bluetooth で Wii リモコンをペアリングします。Wii リモコンは本来 PC に繋げる想定はしていないため少々癖があります。

手順を追って説明します。

^{*1} <http://touchmote.net/>

終わりがみえてきたぞ・・・・・

Unity と Wii リモコンで簡単アーケードゲーム制作

2019年1月26日 初版第1刷 発行
著者 トミー (@Tomy_0331)
