

Wii リモコンで遊べる Unity ゲーム制作

トミー (@Tomy_0331) 著

2019-03-04 版 発行

目次

第 1 章	はじめに	1
1.1	この本について	1
1.2	開発環境	1
第 2 章	Unity でゲーム部分を作ろう	2
2.1	事前準備	2
2.2	敵(ゾンビ)の用意	3
2.3	ゾンビを撃つスクリプトを書く	5
2.4	ゾンビを歩かせる	7
2.5	ゾンビを攻撃させる	10
2.6	エフェクトを付ける	13
第 3 章	マップのモデリング	15
3.1	blender によるモデリング	15
3.2	テクスチャのマッピング	18
3.3	Unity でモデルを配置する	19
第 4 章	ゲームを仕上げる	21
4.1	ゾンビのスポナーをつくる	21
4.2	UI とゲームオーバーを実装する	23
4.3	ビルド	25
第 5 章	Wii リモコンで遊べるようにする	26
第 6 章	余談	29
6.1	ゲームをさらに面白くしよう	29
6.2	フットペダルと高専祭	29
第 7 章	あとがき	31

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。本書では「入門レベルよりちょっとクオリティの高いゲームを作ってみたい」「弾轟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、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普段ゲームをしない人でも楽しめるはずです。

なお、本書では 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。C#におけるプログラミングの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 版も操作は基本的に同じですが、今回使う Wii リモコンの Mac での入力方法は対象外とさせていただきます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。

blender 2.79

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

今回はゾンビを射撃する FPS ゲームを作ります。Wii リモコンで撃てるようにするのですが、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（詳細は 5 章にて）。この章では制作の順を追って解説していきます。

2.1 事前準備

プロジェクトを作成する

Unity を起動し「New」から新規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」、その他設定して Create project をクリックします。

Standard Assets のインポート

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。

「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。インポート時にエフェクトや車データ、コントローラーなど選択できますが、今回は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。インスペクタの「Terrain」で設定ができます、「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。

2.2 敵 (ゾンビ) の用意

今回メインの作業とな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」と検索すると最も上位に出るアセットをインポートします。



図 2.2: ゾンビの素材

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などによく見る、敵が死ぬとぐったり人形のように崩れていくアレです(バグって荒ぶるのもよく見る)。

インポートしたゾンビのアセットの「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オブジェクト階層を展開して、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。例えば、「Pelvis」にはゾンビの「Bip01 Pelvis」を指定します。Elbow(肘)と Forearm(前腕)など言い方が異なる部分もあるので注意してください。

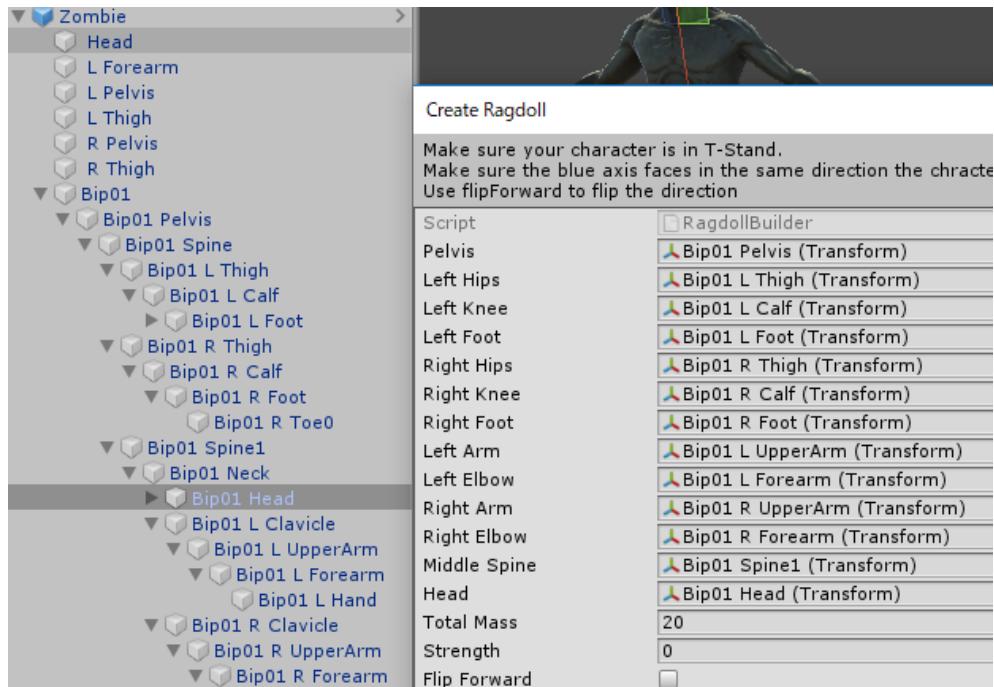


図 2.3: ラグドールの設定

タグを付ける

この後ゾンビを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おきます。インスペクタの「Tag > add tag」から「enemy」と名付けたタグを追加し、ラグドールを適用することで作られたコライダが付いているゾンビのオブジェクト全てに適用しましょう。コライダが付いていないオブジェクトのタグが変わっても問題ないので、階層を shift キーで押しながら複数選択してから Tag を設定すると、楽にまとめて変更できます。

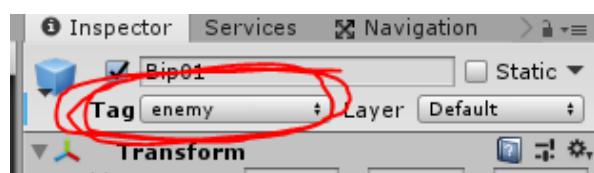


図 2.4: tag の設定

2.3 ゾンビを撃つスクリプトを書く

ここでゲームのプログラムを作ります。「ShotCam.」という名前で C#スクリプトを新規作成し、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 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「Zombie」というスクリプトを作成し、以下のクラスを追加しゾンビの親オブジェクト（一番上の階層）にアタッチします。

Zombie.cs

```
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、どこを撃たれても clickObject.transform.root によって 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。

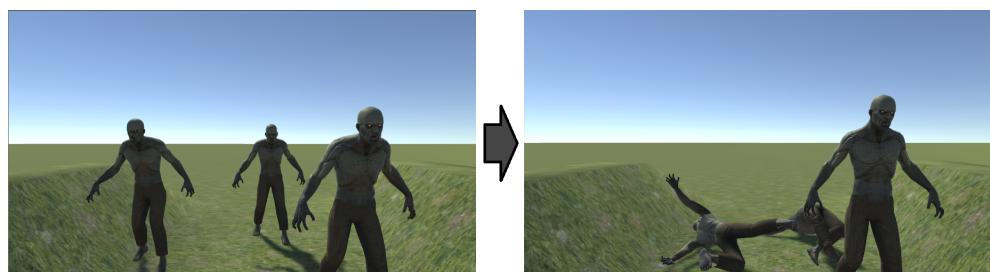


図 2.5: 実行結果(クリックすると倒れる)

2.4 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機能が標準で入っています。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。

表 2.1: NavMeshAgent の設定

プロパティ	機能	値
Speed	動きの最高速度 (単位: メートル/秒)	0.8
Radius	エージェントの半径。障害物同士の間を通り抜ける計算に使用	0.25
Height	エージェントの高さ。頭上にある障害物を通り抜ける計算に使用	1.8

次に Terrain に対し経路探索のための必要な設定をします。「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。選択して「Bake」からパラメータを調整し Bake することで、ゾンビが歩ける場所が青く表示されます。

画像では試しに障害物として 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。

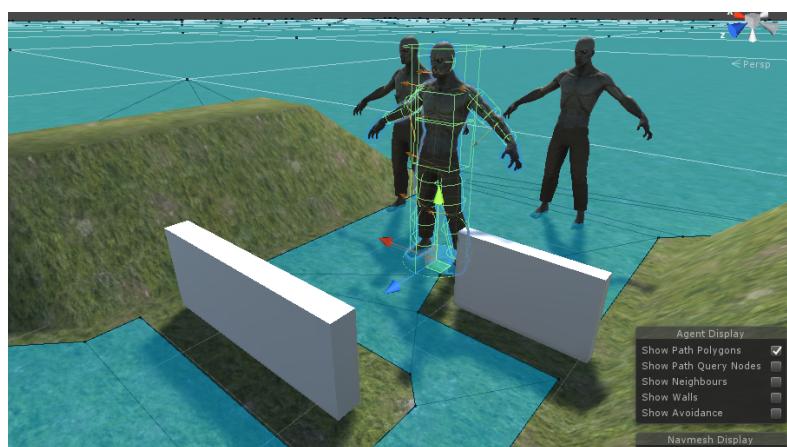


図 2.6: 経路となる領域を Bake した様子

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。インポートしたゾンビのアセットに 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。遷移を表す矢印はノードを右クリックし、「Make Transition」から作成します。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の設定です。チェックが付いたままだと、例えばゾンビが立ち止ったのに、アニメーションではしばらく歩く動作のまま滑るようになってしまいます。

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。

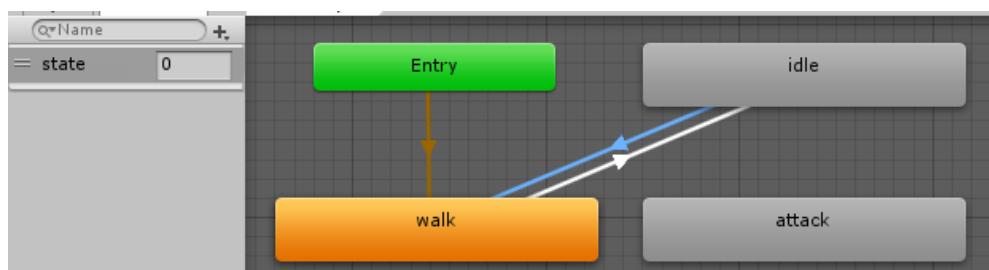


図 2.7: Animator の設定

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;
```

そして次のように書き换えていきます

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() {
        if (!agent.enabled) return;
        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 で行っている上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が発生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（矛盾）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という手法にしています。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。

2.5 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ってるだけです。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。

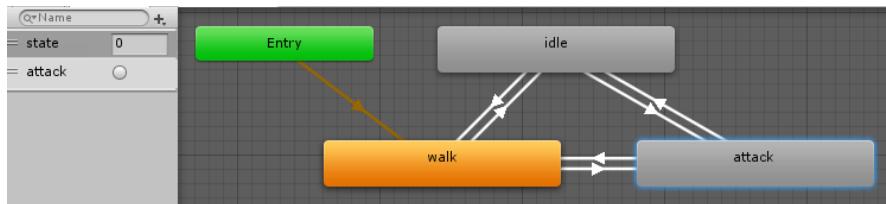


図 2.8: animator の設定



図 2.9: 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ようにするのですが、1からコードを書くと面倒ですので 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ため、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 関数の先頭に以下のコードを追加します

Zombie.cs

```
timeElapsed += Time.deltaTime;
if (timeElapsed >= timeOut && stop) {
    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() {
    if (!agent.enabled) {
        return; //ゾンビが死んでいたら無効
    }
    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 が有効かどうか確認する if 判定を加えています。フラグ変数作ってもいいのですが、簡略化するためこのような手法をとりました。

2.6 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"撃った手ごたえ"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 2.10: 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します。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、爆発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 全て 0.3 にしますと適切なサイズになります。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エフェクトアセットを使ったりエフェクトのプロパティを変えるなど、好みに合うよう試してみるのもいいでしょう。

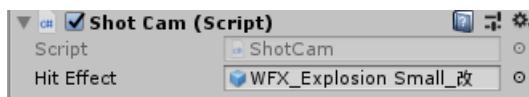


図 2.11: スクリプトにアタッチしたエフェクト

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、コンソールのエラーメッセージをダブルクリックすることで原因となる処理の行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 2.12: 実行結果

■ コラム: 正常に動かない?

スクリプトを書いて「これでゲームが動くぞっ」と実行してもエラーが出たり思つた通りに動かないということはよくあります。「プログラミング難しい」「Unity 何も分からん」など思ってしまいますが、一発で動くことのほうが多いです。エラー文での検索や、「Debug.Log」関数で原因を探してみましょう。

第3章

マップのモデリング

2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を、モデリングにて紹介しようと思います。本書で扱うゲームはどんなマップでも基本的に遊べますので、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ってもOKです。

3.1 blenderによるモデリング

無料3Dモデリングソフト「blender」を使います。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在の場所を舞台にしてみます。最近オタクで流行っている某アニメをパロって佐賀駅をモデリングしてしまいましょう。ゾンビに溢れて荒れる佐賀駅、いいですね。

インターネットミームで海底都市SAGAというものがありますが、筆者のゲームでは屍都市SAGAです。もしゲームがヒットしたら佐賀駅が聖地になります。よろしくお願ひします（？）。

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。屍都市弊社！

素材となる資料

実在する建物の場合、おすすめの資料は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 (Google MAP より)

航空写真は blender の「下絵」機能を使って表示し、それに合わせてモデリングすることで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 は沢山の機能があるため、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。逆にこれら以外は使わなくてもほぼ作ることは可能です。

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

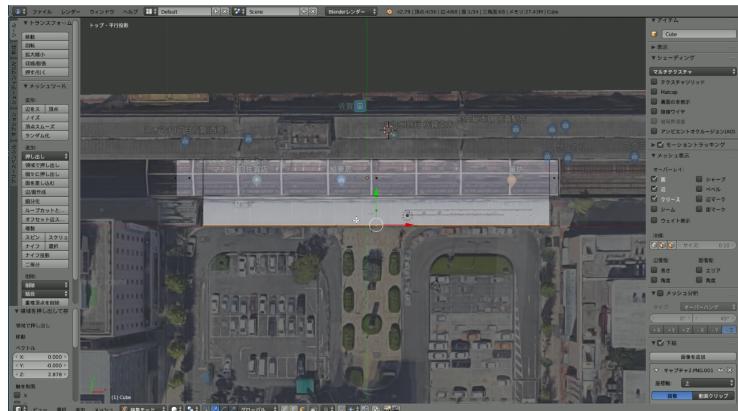


図 3.2: 下絵機能を使い上から見た形を作る様子

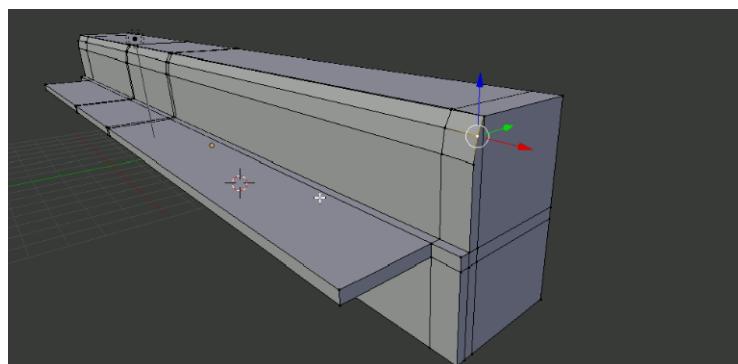


図 3.3: 横側に「押し出し」でいる様子



図 3.4: モデリングした佐賀駅南口

この他、周辺の建物（西友）もモデリングしました。

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。モデルの面に対し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。

テクスチャマッピングは「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展開していき、ループも可能な一枚の画像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。

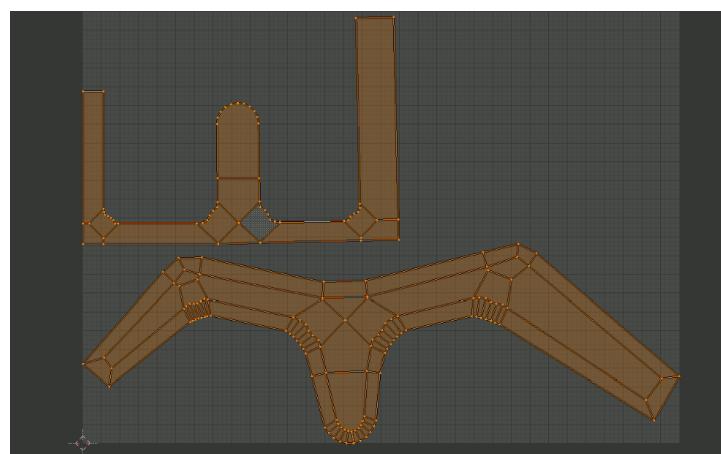


図 3.5: UV 展開

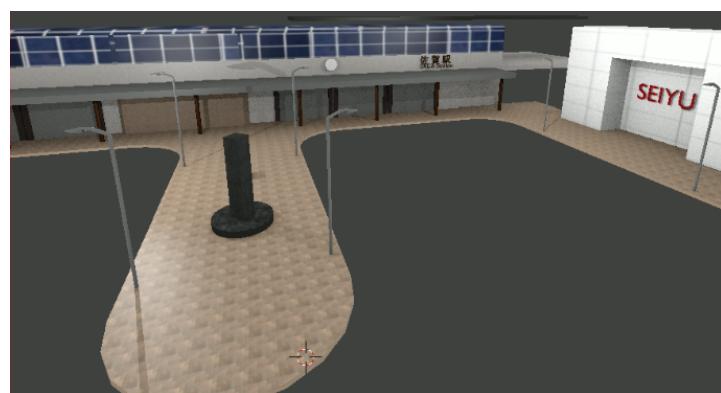


図 3.6: テクスチャを貼った様子

■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、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。

テクスチャには材質の英単語で検索すると見つけやすいです。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)を絞り込めます。

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Blender側でテクスチャを新規作成し画像の参照をさせることと、マテリアルや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などです。



図 3.7: Unity に設置した駅やフリー素材のモデル

佐賀駅や、Asset Storeでインポートした車やビルを配置しました。木はTerrainで生やしています。また、火事を演出する炎のエフェクト、煙、一帯を覆う霧などを設置しました。これらもAsset Storeからの素材で、設置したい場所にプレハブを置くだけです。

Unity上でのマップ作成でAsset Storeから「Simple Cars Pack」、「Simple City pack plain」、「Unity Particle Pack 5.x」をお借りしました。



図 3.8: 炎や煙のエフェクトを設置し完成した MAP

■コラム: マップは完璧にしなくていい?

今回モデリングしたマップですが、実は駅の裏側は何もなく、ハリボテになっています。それどころか電車は貫通したりビルは浮いてたりしています。普通だったらありえないことですが、ゲームでは必ずしも現実や物理法則に従う必要はないのです。駅の裏側はゲームをプレイしていて見えないところで作りこまなくていいですし、逆に奥のビルは丁度良く見えるように浮かせています。

ゲームをしていて見えないところにも、こだわってオブジェクトを置いたり高画質なテクスチャを貼ると、パフォーマンスが落ちてしまい本末転倒です。ゲームのマップは"舞台装置"ですから必要最低限でいいのです。



図 3.9: 駅の裏側

第4章

ゲームを仕上げる

4.1 ゾンビのスポナーをつくる

ラストスパートです。マップが完成し、ゲームとして遊べるよう仕上げていきます。

ゾンビがプレイヤーに向かってずっと集まってくるようするため、スポナーを作り設置します。スポナーとは、プレハブ（クローン）の生産工場のようなもので、シーン上に設置すると次々にオブジェクトを生み出します。

オブジェクト指向と言われる考え方でゾンビのクラスがあるため、スポーンしたゾンビは互いに物理的な接触以外に干渉することなく、独立して歩いたり撃たれて倒れたりします。

スポナーのスクリプトを書く

新たに「Spawner.cs」とスクリプトを作成し、以下のコードを書きます。

Spawner.cs

```
public class Spawner : MonoBehaviour {
    public GameObject zombie;
    [Range(1,2000)]
    public int probability = 100; //スポーンの確率
    void Update() {
        if (Random.Range(1, probability) == 1) {
            spawn();
        }
    }
    void spawn() {
        GameObject.Instantiate(zombie, this.transform.position, Quaternion.identity);
    }
}
```

Random.Range は便利な関数で、指定した値の範囲でランダムに返します。このプログラムでは毎フレーム判定し「1」となればスポーンさせているため第二引数の probability の値が小さければ高確率、大きければ低確率となります。

スポナーを設置する

Unity の「Hierarchy」の「Create>Create Empty」をクリックすると座標データである Transform コンポーネントのみ付いたオブジェクトが作成できます。これに Spawner.cs を加え、ゾンビプレハブをアタッチすることでスポナーになります。

ゾンビをスpawnさせたい位置にこのスポナーを設置しましょう。コピーしたオブジェクトを様々な場所に置くことも可能です。スポナーは車や物陰の裏など、直接プレイヤーが見えない場所に設置し、ゾンビがどこからともなく現れるような工夫をします。

このスポナーは 3D モデルのようにメッシュがないため、見失ったり距離感が分からなくなることがあります。そこで、インスペクタの左上のアイコンをクリックするとシーン上で見える印を設定できます。この印はゲーム実行時には表示されないため、MAP 編集の時に便利です。



図 4.1: 設置したスポナー



図 4.2: 様々な場所からプレイヤーに群がるゾンビ

Spawner クラスの probability は public 変数で宣言され、Unity のインスペクタで弄れるようになっているためプレイして試しながら調整するといいでしょう。

4.2 UIとゲームオーバーを実装する

スクリプトを書く

ゾンビを撃った時に獲得できるスコアと、プレイヤーの残りライフを画面に表示します。

スコアとライフの変数、そしてUIを管理するためのマネージャークラス「Manager.cs」を新規作成します。「using UnityEngine.UI;」を付け加えるのを忘れないよう気を付けてください。

Spawner.cs

```
public class Manager : MonoBehaviour {
    public Text scoreText;
    public Text lifeText;
    private int life = 2;
    private int score = 0;

    public void scoreUP(int n) {
        score += n;
        scoreText.text = score.ToString();
    }
    public void lifeDown() {
        life--;
        lifeText.text = "残り"+life.ToString();
    }
}
```

そしてマネージャーとして「Create Empty」で作ったオブジェクトにアタッチします。またUIに必要なCanvasとスコア、ライフそれぞれのimage,textを以下のように作成します。

Manager

Canvas

| scoreUI_image

| | Text

| | scoreText

| lifeUI_image

| | Text

| | lifeText

Zombie.cs

```
Manager manager;
void Start(){
    ...
    manager = GameObject.Find("Manager").GetComponent<Manager>();
}
public void death() {
    ...
    manager.scoreUP(100);
}
void damage() {
    ...
    manager.lifeDown();
}
```

これで、ゾンビを撃つとスコアが100増え、ゾンビから攻撃されるとライフが1減るようになりました。



図4.3: 完成したUI

シーン遷移

タイトル画面を作ります。「Title」という名前でシーンを新規作成し、「Title.cs」も準備します。シーン遷移となるボタンやタイトルテキスト、イラストなどをCanvasに配置します。

Zombie.cs

```
using UnityEngine.SceneManagement;
public class Title : MonoBehaviour{
    public void startGame() {
        SceneManager.LoadScene("MainScene");
    }
}
```

スクリプトはタイトルシーン用のマネージャオブジェクトを作りアタッチし、ボタンを押すと関数を呼び出すようインスペクタで設定します。



図 4.4: タイトルとボタンを押したときの遷移

一番簡単にゲームオーバーを実装するなら、MainScene 側でライフが 0 になった時にタイトル画面に遷移させます。

Manager.cs

```
if(life==1) SceneManager.LoadScene("title");
```

これでは攻撃されたら一瞬でタイトル画面に飛ばされてしまうので、「GameOver」と表示するシーンに一度遷移させたり、ゲーム上で死亡演出を加えるのもいいでしょう。

4.3 ビルド

ゲームが完成したらビルドをします。「File>Build Settings」からシーンの設定をし、「Build」でゲームがビルドされます。

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自作する人もいるそうです。

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。

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。

また OS や Bluetooth ドライバにもよって変わることがあります。この項では筆者の thinkpad x250 内蔵の Bluetooth、Windows10 環境での設定方法を載せます。

手順を追って説明します。

1. デスクトップのツールバーの Bluetooth アイコンを右クリックし「パーソナルエリアネットワークへ参加」を選択。

^{*1} <http://touchmote.net/>
26

第5章 Wii リモコンで遊べるようにする

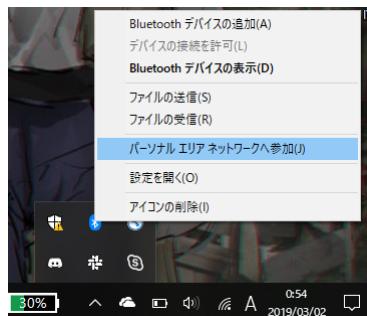


図 5.1: Wii リモコンのペアリング手順 (1)

2. デバイスの追加をクリックし、wii リモコンの 1,2 ボタンを同時に押し続ける。

この PC に追加するデバイスまたはプリンターを選びます
デバイスの選択



図 5.2: Wii リモコンのペアリング手順 (2)

3. 入力デバイスを検知したら選択 (Nintendo RVL-CNT-01 というような名前). 4.PIN コードを求められるが何も入力せずに「次へ」を選択. 5. ペアリングが終わったら TouchMote を起動し、ソフトが検知したら成功.

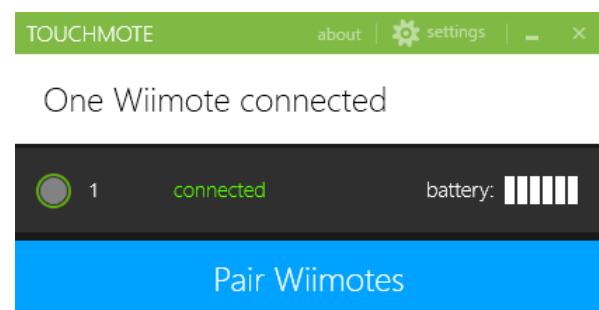


図 5.3: Wii リモコンのペアリング手順 (3)

第5章 Wii リモコンで遊べるようにする

動作確認

TouchMote で Wii リモコンの B ボタンにマウスのクリックを割り当てましょう。Wii ザッパーがあると引き金のように操作できます。

Wii のセンサーバーは面倒ですが Wii 本体に接続して起動することで使えます。このやり方では Wii 本体は電源供給のためだけに使っており不便です。非純正ではありますが、USB 接続や電池式のワイヤレスタイプもネット通販で販売されているのでそちらを試してみるのもいいでしょう。

センサーバーをディスプレイの上か下に設置し (TouchMote で設定可)，ゲームをパソコンで通常通り起動すれば Wii リモコンで遊ぶことができます。ゾンビ FPS のアーケードゲーム完成です！！！

■コラム：センサーバーが点いているかの確認

センサーバーは赤外線 LED が内蔵されており、その光を Wii リモコンで検知していることは説明しました。しかし、赤外線は我々人間には見えないので給電されて点いているかが目視で確認できません。

そこで、スマートフォンやデジカメのカメラを通して見てください。赤外線は肉眼では見えませんが、これらのカメラでは映りますので確認のためぜひ使ってみてください。中には iPhone6 など、赤外線カットフィルターで見えないものもありますので注意してください。

第 6 章

余談

6.1 ゲームをさらに面白くしよう

無事ゲームを完成させプレイした皆さんはお気づきでしょう。ゲーム性がイマイチ！だということに、ひたすら現れてくるゾンビを撃つだけで単調になってしまっています。

そこで、ゾンビに HP を設定してヘッドショットを一撃にしたり、ゾンビ犬や鳥など特殊なキャラクタを登場させてアクセントを加えるのもいいでしょう。想像が膨らんできましたか？本書では基盤となるゲームは実装しましたが、その後どう発展させていくかはあなたの自由です。「適切な難易度はどれくらいだろう」など遊ぶ人のことを考えて試行錯誤するのもゲーム制作の醍醐味ですので、ぜひチャレンジしてほしいと多います。

■コラム: UI の壁

ゲーム制作者によくあることは、ストーリーやキャラクター、オリジナル要素などシステムに凝りすぎて UI や操作性、プレイヤーが遊びやすいかという部分を軽視してしまうことです。しかし、どんなにいいゲームも UI・操作性の壁を超えないとい評価してもらえないことがあります。我々の想像する以上に視覚や直感からの印象は大きく、裏を返せばシンプルなゲームでも遊びやすさや UI の工夫を施せば高評価も狙えるということです。ぜひプログラミングの技術だけでなく、「UI/UX」の視点でより良いゲーム制作をしてみましょう。

6.2 フットペダルと高専祭

筆者が高専 3 年生の時、「高専危機」というシューティングゲームを作成しました。このゲームはモデリングした高専を舞台にしており、現れる敵兵士を倒しながらゴールを目指すゲームです。Wii リモコンによる射撃の他、障害物に隠れるアクションをフットペダ

ルで切り替えできるようにしています。フットペダルは「REVIVE USB^{*1}」という、ビンに電気スイッチを繋いで任意のキー入力に変換できるデバイスを用いました。

4年生ではフットペダルとハンドル型wiiリモコンでレースゲームも作りました。Wiiは今では古いゲーム機かもしれませんが、リモコンは高機能で可能性は無限大です。

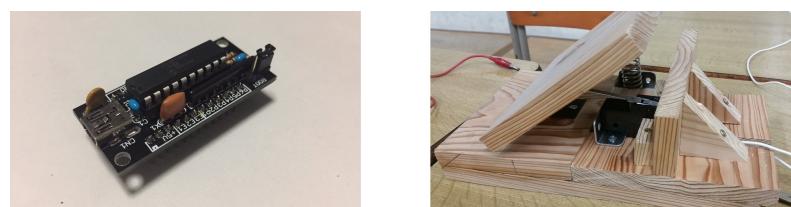


図 6.1: REVIVE USB と自作フットペダル



図 6.2: シューティングゲーム「高専危機」

^{*1} <http://bit-trade-one.co.jp/product/assemblydisk/revive-usb/>
30

第7章

あとがき

Wii リモコンで遊べる

Unity ゲーム制作

2019年3月4日 発行

著 者 トミー (@Tomy_0331)
